Introduction au framework NUXT.JS par l'exemple

Serge Tahé, décembre 2019

Ce site a été créé avec le convertisseur [Word ou ODT - > HTML] créé par l'IA Gemini 3 en janvier 2026.

1 Introduction au framework NUXT.JS

Le PDF de ce document est disponible |ICI|.

Ce document fait partie d'une série de quatre articles :

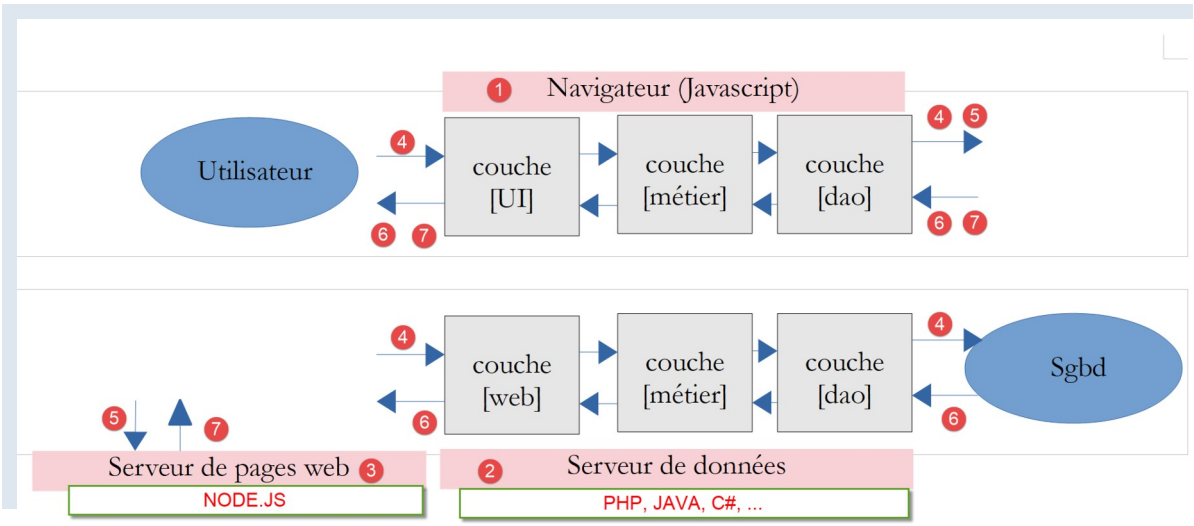
1. |Introduction au langage PHP7 par l'exemple| ;
2. |Introduction au langage ECMASCRIPT 6 par l'exemple| ;
3. |Introduction au framework VUE.JS par l'exemple| ;
4. |Introduction au framework NUXT.JS par l'exemple|. **C'est le document présent** ;

Ce sont tous des documents pour **débutants**. Les articles ont une suite logique mais **sont faiblement couplés** :

- le document [1] présente le langage PHP 7. Le lecteur seulement intéressé par le langage PHP et pas par le langage Javascript des articles suivants s'arrêtera là ;
- les documents [2-4] visent à construire un client Javascript au serveur de calcul de l'impôt développé dans le document [1] ;
- les frameworks Javascript [vue.js] et [nuxt.js] des articles 3 et 4 nécessitent de connaître le Javascript des dernières versions d'ECMASCRIPT, celles de la version 6. Le document [2] est donc destiné à ceux qui ne connaissent pas cette version de Javascript. Il fait référence au serveur de calcul de l'impôt construit dans le document [1]. Le lecteur de [2] aura alors parfois besoin de se référer au document [1] ;
- une fois ECMASCRIPT 6 maîtrisé, on peut aborder le framework VUE.JS qui permet de construire des clients Javascript s'exécutant dans un navigateur en mode SPA (Single Page Application). C'est le document [3]. Il fait référence à la fois au serveur de calcul de l'impôt construit dans le document [1] et au code du client Javascript autonome construit en [2]. Le lecteur de [3] aura alors parfois besoin de se référer aux documents [1] et [2] ;
- une fois VUE.JS maîtrisé, on peut aborder le framework NUXT.JS qui permet de construire des clients Javascript s'exécutant dans un navigateur en mode SSR (Server Side Rendered). Il fait référence à la fois au serveur de calcul de l'impôt construit dans le document [1], au code du client Javascript autonome construit en [2] ainsi qu'à l'application [vue.js] développée dans le document [3]. Le lecteur de [4] aura alors parfois besoin de se référer aux documents [1] [2] et [3] ;

Ce document poursuit le travail fait dans le document [3] avec le framework VUE.JS.

Cette section s'intéresse à l'architecture suivante :



- en `[1]`, un navigateur web affiche des pages web `[5, 7]` isues d'un serveur [3] à destination d'un utilisateur. Ces pages contiennent du Javascript implémentant un client d'un service web de données `[2]` ainsi qu'un client d'un serveur de fragments de pages web `[3]` ;
- en `[2]`, le serveur web est un serveur de données. Il peut être écrit dans n'importe quel langage. Il ne produit pas de pages web au sens classique (HTML, CSS, Javascript) sauf peut-être la 1ère fois. Mais cette 1ère page peut être obtenue d'un serveur web classique `[3]` (pas un serveur de données). Le Javascript de la page initiale

va alors générer les différentes pages web de l'application en obtenant les données **[4]** à afficher, auprès du serveur web qui agit comme un serveur de données **[2]**. Il peut également obtenir des fragments de page web **[5]** pour habiller ces données auprès du serveur de pages web **[3]** ;
- en **[4]**, l'utilisateur initie une action ;
- en **[6,7]** : il reçoit des données habillées par un fragment de page web ;

Le framework [nuxt.js] |https://fr.nuxtjs.org/| va nous permettre d'implémenter le fonctionnement suivant :

- la 1ère page de l'application est délivrée par le serveur [node.js] [3]. Par ailleurs les autres pages de l'application sont également présentes sur ce même serveur. Elles sont délivrées lorsque l'utilisateur tape leur URL à la main dans le navigateur. Ces pages embarquent une application [vue.js] (approximativement) ;
- une fois la 1ère page chargée dans le navigateur, l'application se comporte comme une application [vue.js] classique. Dans notre schéma ci-dessus, elle va alors dialoguer avec le serveur de données [2] ;

Au final, l'application se comporte comme une application [vue.js] sauf pour la 1ère page et lorsque l'utilisateur tape des URL à la main. Dans ces cas, la page est cherchée sur le serveur [3]. Lorsque qu'un moteur de recherche demande les différentes pages de l'aplication, il reçoit les pages du serveur [3]. Celles-ci ont pu être optimisées pour le SEO (Search Engine Optimization). Dans une application [vue.js], le moteur de recherche reçoit une page avec peu de signification SEO. Par exemple, dans l'application du client du serveur de calcul de l'impôt du document [3], la page reçue par le navigateur lors du démarrage de l'application était la suivante :

```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.    <head>
4.      <meta charset="utf-8">
5.      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6.      <meta name="viewport" content="width=device-width,initial-scale=1.0">
7.      <link rel="icon" href="/client-vuejs-impot/favicon.ico">
8.      <title>vuejs</title>
9.    <link href="/client-vuejs-impot/app.js" rel="preload" as="script"></head>
10.   <body>
11.     <noscript>
12.       <strong>We're sorry but vuejs doesn't work properly without JavaScript enabled. Please
    enable it to continue.</strong>
13.     </noscript>
14.     <div id="app"></div>
15.     <!-- built files will be auto injected -->
16.   <script type="text/javascript" src="/client-vuejs-impot/app.js"></script></body>
17. </html>
```

C'est l'unique page chargée par le navigateur. Toutes les autres pages de l'application sont générées dynamiquement par le Javascript sans l'aide du navigateur. Certains moteurs de recherche se contentent de cette page. D'autres vont plus loin en exécutant le Javascript contenu dans la page (ligne 9 ci-dessus). Une autre page est alors obtenue. Celle-ci peut contenir une opération asynchrone pour aller chercher les données que la page va afficher. Dans ce cas les moteurs de recherche n'attendent pas. On se retrouve alors avec une page incomplète. On se rappelle peut-être que c'est le cas de notre client [vue.js] du serveur de calcul de l'impôt : de façon asynchrone, il initialise au cours du chargement de la 1ère page, une session jSON avec le serveur de calcul de l'impôt. Dans ce cas précis, cela n'influe pas sur la page récupérée par le moteur de recherche. Pour d'autres applications, cela pourrait être pénalisant en termes SEO.

Avec [nuxt.js] on peut servir au moteur de recherche une page plus signifiante pour chacune des pages de l'application.

Les scripts de ce document sont commentés et leur exécution console reproduite. Des explications supplémentaires sont parfois fournies. Le document nécessite une lecture active : pour comprendre un script, il faut à la fois lire son code, ses commentaires et ses résultats d'exécution.

Les exemples du document sont disponibles |**ici**|.
L'application serveur PHP 7 peut être testée |**ici**|.

Serge Tahé, décembre 2019

2 L'environnement de travail

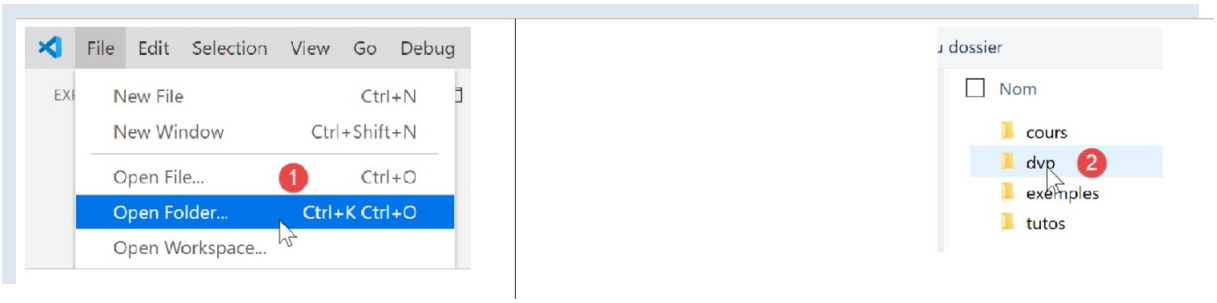
Nous utiliserons le même environnement de travail que celui présenté dans les documents :

- |Introduction au langage **ECMASCRIPT 6** par l'exemple| ;
- |Introduction au framework **VUE.JS** par l'exemple| ;

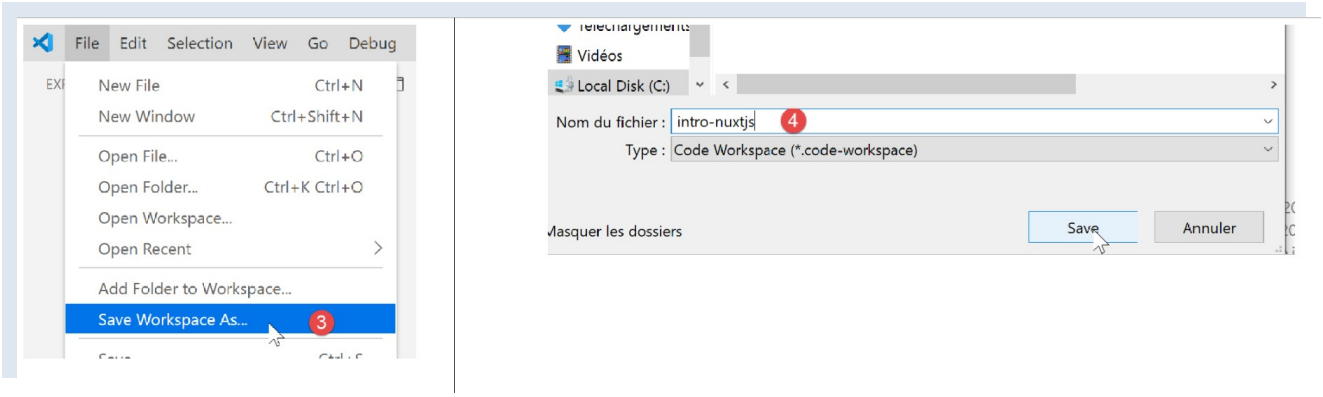
3 Une première application [nuxt.js]

3.1 Création de l'application

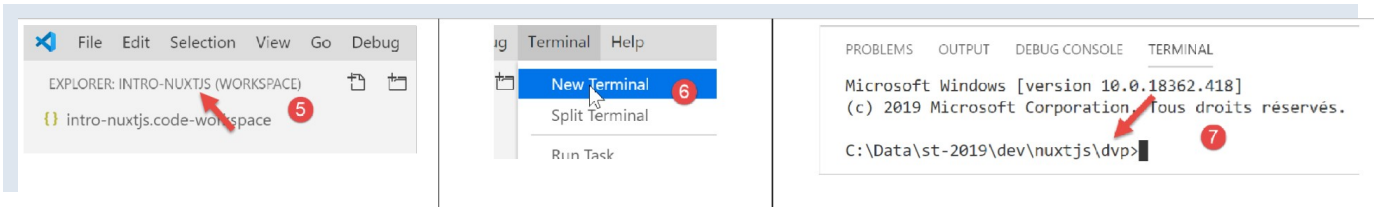
Pour nos développements [nuxt.js] nous continuons à utiliser VS Code. Nous avons créé un dossier [dvp] vide dans lequel nous allons mettre nos exemples. Puis nous ouvrons ce dossier :



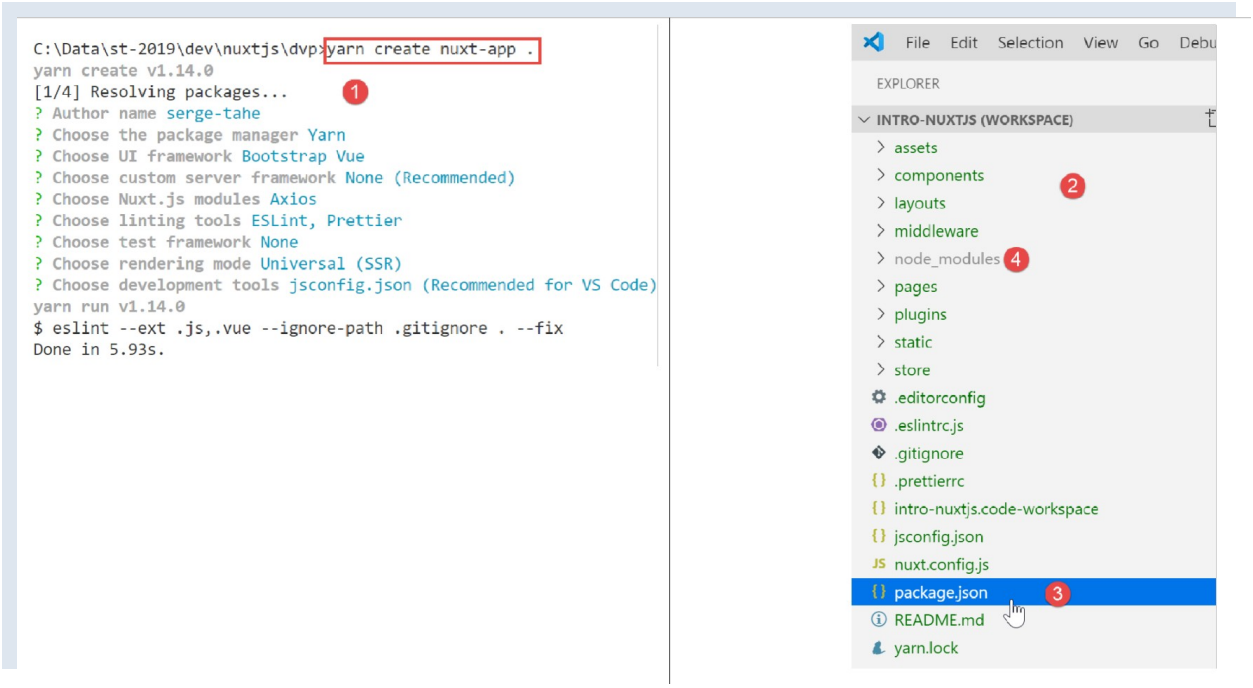
Nous sauvegardons l'espace de travail sous le nom [intro-nuxtjs] [3-5] :



Nous ouvrons un terminal [6-7] :



Jusqu'à maintenant, nous avons utilisé le gestionnaire de packages Javascript [npm]. Pour changer, nous allons ici utiliser le gestionnaire [yarn]. Celui-ci est installé, comme [npm], avec les récentes versions de [node.js]. Pour créer une première application [nuxt], nous utilisons la commande [yarn create nuxt-app <dossier>] [1]. La commande va demander un certain nombre d'informations sur le projet à générer puis, celles-ci obtenues, va le générer [2] :



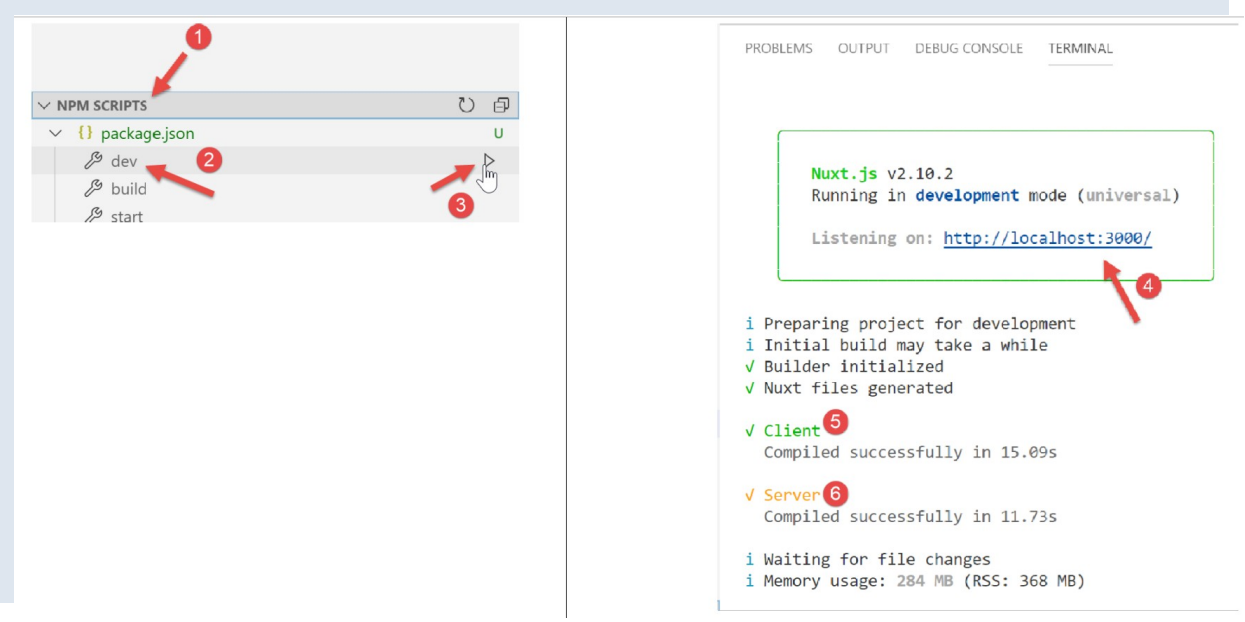
En [2], toute une arborescence de fichiers a été créée. Le fichier [package.json] donne la liste des biblothèques Javascript téléchargées dans le dossier [node-modules] [4] :

```
1.  {
2.    "name": "nuxt-intro",
3.    "version": "1.0.0",
4.    "description": "nuxt-intro",
5.    "author": "serge-tahe",
6.    "private": true,
7.    "scripts": {
8.      "dev": "nuxt",
9.      "build": "nuxt build",
10.     "start": "nuxt start",
11.     "generate": "nuxt generate",
12.     "lint": "eslint --ext .js,.vue --ignore-path .gitignore ."
13.   },
14.   "dependencies": {
15.     "nuxt": "^2.0.0",
16.     "bootstrap-vue": "^2.0.0",
17.     "bootstrap": "^4.1.3",
18.     "@nuxtjs/axios": "^5.3.6"
19.   },
20.   "devDependencies": {
21.     "@nuxtjs/eslint-config": "^1.0.1",
22.     "@nuxtjs/eslint-module": "^1.0.0",
23.     "babel-eslint": "^10.0.1",
24.     "eslint": "^6.1.0",
25.     "eslint-plugin-nuxt": ">=0.4.2",
26.     "eslint-config-prettier": "^4.1.0",
27.     "eslint-plugin-prettier": "^3.0.1",
28.     "prettier": "^1.16.4"
29.   }
30. }
```

Ce fichier reflète les réponses données à la commande [create nuxt-app] pour définir le projet créé (novembre 2019). Le lecteur peut avoir un fichier [package.json] différent :

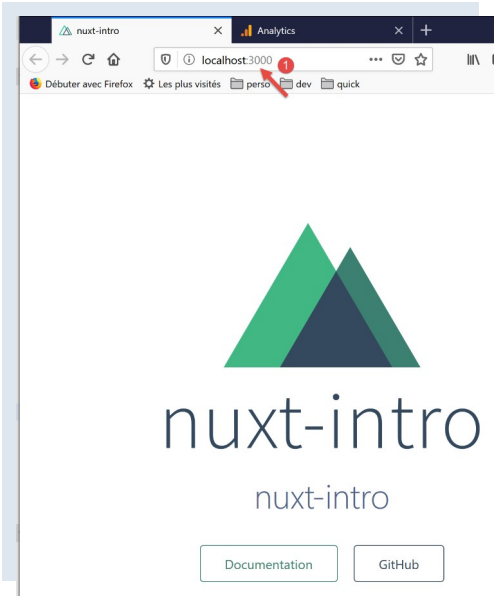
- il a pu donner des réponses différentes aux questions ;
- la commande [create nuxt-app] aura évolué depuis l'écriture de ce document : les dépendances et les versions auront changé ;

La ligne 8 du script est la commande qui lance l'application :



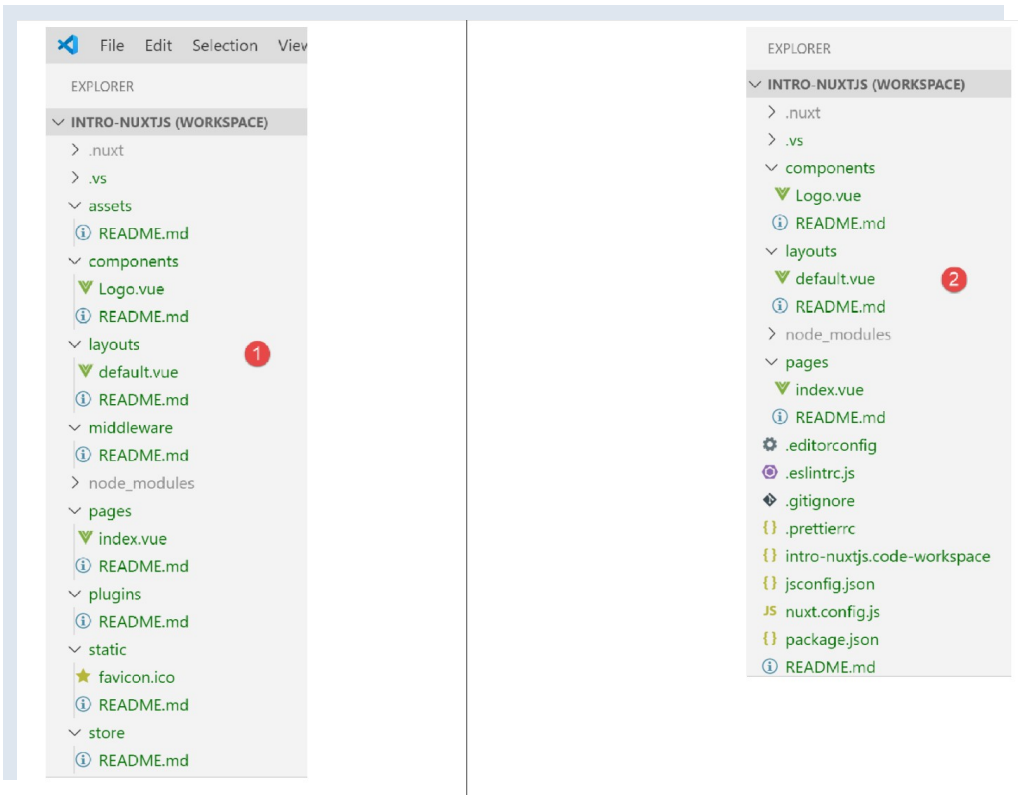
- en [4], on voit que l'application est disponible à l'URL [localhost:3000] ;
- en [5-6], on voit que l'application donne naissance à un serveur [6] et à un client (de ce serveur) [5] ;

Demandons l'URL [http://localhost:3000/] dans un navigateur :



3.2 Description de l'arborescence d'une application [nuxt]

Reprenons l'arborescence de l'application créée :



Le rôle des dossiers est le suivant :

assets	ressources non compilées de l'application (images, ...) ;
static	les fichiers de ce dossier seront disponibles à la racine de l'application. On met dans ce dossier des fichiers qu'on doit trouver à la racine de l'application comme par exemple le fichier [robots.txt] destiné aux moteurs de recherche ;
components	les composants [vue] de l'application utilisés dans les [layouts] et les [pages] ;
layouts	les composants [vue] de l'application servant de mise en page des [pages] ;
pages	les composants [vue] affichés par les différentes routes de l'application. On pourrait les appeler les vues de l'application. Les pages jouent un rôle particulier dans [nuxt] : les routes sont créées dynamiquement à partir de l'arborescence trouvée dans le dossier [pages] ;
middleware	les scripts exécutés à chaque changement de route. Ils permettent de contrôler celles-ci ;
plugins	porte un nom prêtant à confusion. Peut contenir des plugins mais également des scripts classiques. Les scripts trouvés dans ce dossier sont exécutés au démarrage de l'application ;
store	s'il contient un script [index.js] alors celui-ci définit une instance du store de [Vuex] ;

Si un dossier est vide, on peut le supprimer de l'arborescence. Ci-dessus, les dossiers [assets, static, middleware, plugins, store] peuvent être supprimés [2].

3.3 Le fichier de configuration [nuxt.config]

L'exécution de l'application est contrôlée par le fichier [nuxt.config.js] suivant :

```
1.  export default {
2.    mode: 'universal',
3.    /*
```

```
4.     ** Headers of the page
5.     */
6.    head: {
7.      title: process.env.npm_package_name || '',
8.      meta: [
9.        { charset: 'utf-8' },
10.       { name: 'viewport', content: 'width=device-width, initial-scale=1' },
11.       {
12.         hid: 'description',
13.         name: 'description',
14.         content: process.env.npm_package_description || ''
15.       }
16.     ],
17.     link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
18.   },
19.   /*
20.    ** Customize the progress-bar color
21.    */
22.   loading: { color: '#fff' },
23.   /*
24.    ** Global CSS
25.    */
26.   css: [],
27.   /*
28.    ** Plugins to load before mounting the App
29.    */
30.   plugins: [],
31.   /*
32.    ** Nuxt.js dev-modules
33.    */
34.   buildModules: [
35.     // Doc: https://github.com/nuxt-community/eslint-module
36.     '@nuxtjs/eslint-module'
37.   ],
38.   /*
39.    ** Nuxt.js modules
40.    */
41.   modules: [
42.     // Doc: https://bootstrap-vue.js.org
43.     'bootstrap-vue/nuxt',
44.     // Doc: https://axios.nuxtjs.org/usage
45.     '@nuxtjs/axios'
46.   ],
47.   /*
48.    ** Axios module configuration
49.    ** See https://axios.nuxtjs.org/options
50.    */
51.   axios: {},
52.   /*
53.    ** Build configuration
54.    */
55.   build: {
56.     /*
57.      ** You can extend webpack config here
58.      */
59.     extend(config, ctx) {}
60.   }
61. }
```

- ligne 2 : le type d'application générée :
  - [**universal**] : application client / serveur. Au chargement initial de l'application ainsi qu'à chaque rafraîchissement de page dans le navigateur, le serveur est sollicité pour délivrer la page ;
  - [**sap**] : application de type [Single Page Application] : un serveur délivre initialement la totalité de l'application. Ensuite le client opère seul, même en cas de rafraîchissement d'une page dans le navigateur ;
- lignes 6-18 : définissent l'entête HTML <head> des différentes pages de l'application :
  - ligne 7 : la balise <title> du titre des pages ;
  - lignes 8-16 : les balises <meta> ;
  - ligne 17 : les balises <link>

Dans l'application générée, la balise <head> est la suivante (code source de la page affichée dans le navigateur) :

```
1.  <title>nuxt-intro</title>
2.  <meta data-n-head="ssr" charset="utf-8">
3.  <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
4.  <meta data-n-head="ssr" data-hid="description" name="description" content="nuxt-intro">
5.  <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
6.  <link rel="preload" href="/_nuxt/runtime.js" as="script">
7.  <link rel="preload" href="/_nuxt/commons.app.js" as="script">
8.  <link rel="preload" href="/_nuxt/vendors.app.js" as="script">
9.  <link rel="preload" href="/_nuxt/app.js" as="script">
```

Maintenant, modifions le fichier [nuxt.config] de la façon suivante :

```
1.  head: {
2.      title: 'Introduction à [nuxt.js]',
3.      meta: [
4.        { charset: 'utf-8' },
5.        { name: 'viewport', content: 'width=device-width, initial-scale=1' },
6.        {
7.          hid: 'description',
8.          name: 'description',
9.          content: 'ssr routing loading asyncdata middleware plugins store'
10.       }
11.     ],
12.     link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
13. },
```

Lorsque nous réexécutons l'application, la balise <head> est devenue la suivante (code source de la page affichée dans le navigateur) :

```
1.     <head >
2.       <title>Introduction à [nuxt.js]</title>
3.       <meta data-n-head="ssr" charset="utf-8">
4.       <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
5.       <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing
    loading asyncdata middleware plugins store">
6.       <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
7.       <link rel="preload" href="/_nuxt/runtime.js" as="script">
8.       <link rel="preload" href="/_nuxt/commons.app.js" as="script">
9.       <link rel="preload" href="/_nuxt/vendors.app.js" as="script">
10. <link rel="preload" href="/_nuxt/app.js" as="script">
```

Revenons au fichier [nuxt.config] :

```
1.  export default {
2.    mode: 'universal',
3.    /*
4.     ** Headers of the page
5.     */
6.    head: {
7.      ...
8.    },
9.    /*
10.    ** Customize the progress-bar color
11.    */
12.   loading: { color: '#fff' },
13.   /*
14.    ** Global CSS
15.    */
16.   css: [],
17.   /*
18.    ** Plugins to load before mounting the App
19.    */
20.   plugins: [],
21.   /*
22.    ** Nuxt.js dev-modules
23.    */
24.   buildModules: [
25.     // Doc: https://github.com/nuxt-community/eslint-module
26.     '@nuxtjs/eslint-module'
27.   ],
28.   /*
```

```
29.     ** Nuxt.js modules
30.     */
31.   modules: [
32.     // Doc: https://bootstrap-vue.js.org
33.     'bootstrap-vue/nuxt',
34.     // Doc: https://axios.nuxtjs.org/usage
35.     '@nuxtjs/axios'
36.   ],
37.   /*
38.     ** Axios module configuration
39.     ** See https://axios.nuxtjs.org/options
40.     */
41.   axios: {},
42.   /*
43.     ** Build configuration
44.     */
45.   build: {
46.     /*
47.       ** You can extend webpack config here
48.       */
49.     extend(config, ctx) {}
50.   }
51. }
```

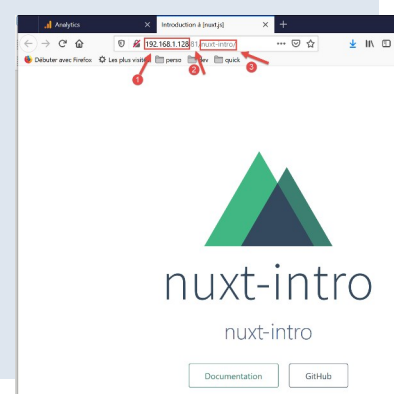
- ligne 12 : entre chaque route du client [nuxt], une barre de chargement (loading) apparaît si le changement de route prend un peu de temps. La propriété [loading] permet de paramétrer cette barre de chargement, ici la couleur de la barre ;
- ligne 16 : les fichiers [css] globaux. Ils seront automatiquement inclus dans toutes les pages de l'application ;
- lignes 24-27 : les modules Javascript nécessaires à la compilation (build) de l'application ;
- lignes 31-36 : les modules Javascript utilisés par l'application ;
- ligne 41 : paramétrage de la bibliothèque [axios] lorsque celle-ci a été sélectionnée par l'utilisateur pour les dialogues HTTP avec des serveurs tiers ;
- lignes 45-50 : paramétrage de la compilation (build) du projet ;

On peut ajouter d'autres clés au fichier de configuration. On peut notamment paramétrer le port de service (3000 par défaut) et la racine du projet (par défaut, le dossier racine du projet). C'est ce que nlous faisons maintenant en ajoutant les clés suivantes :

```
1.  // répertoire du code source
2.    srcDir: '.',
3.    router: {
4.      // URL racine des pages de l'application
5.      base: '/nuxt-intro/'
6.    },
7.    // serveur
8.    server: {
9.      // port de service - par défaut 3000
10.     port: 81,
11.     // adresses réseau écoutées - par défaut localhost=127.0.0.1
12.     host: '0.0.0.0'
13.   }
```

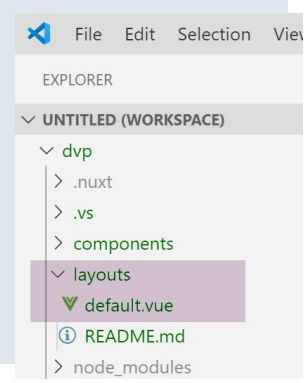
- ligne 2 : où trouver le code source du projet. On le trouve ici dans le dossier courant, ç-à-d au même niveau que le fichier [nuxt.config.js]. C'est la valeur par défaut ;
- lignes 8-13 : configurent le serveur (il ne faut pas oublier qu'une application [nuxt] de type [universal] est installée à la fois sur un serveur et un navigateur client de ce serveur) ;
- ligne 10 : les pages de l'application seront délivrées sur le port 81 du serveur ;
- ligne 12 : par défaut [localhost] (adresse réseau 127.0.0.1). Une machine peut avoir plusieurs adresses réseau si elle appartient à plusieurs réseaux. L'adresse 0.0.0.0 indique que le serveur web écoute toutes les adresses réseau de la machine ;
- lignes 3-6 : configurent le routeur de l'applicatio [nuxt] ;
- ligne 5 : les pages de l'application seront disponibles à l'URL [http://localhost:81/nuxt-intro/];

Ajoutons ces lignes au fichier [nuxt.config.js] puis exécutons le projet (script npm dev). Le résultat est le suivant :



- en [1], l'adresse de la machine sur un réseau public ;
- en [2], le port de service ;
- en [3], l'URL racine de l'application ;

3.4 Le dossier [layouts]



Le dossier [layouts] est destiné aux composants de mise en page. Par défaut, c'est le composant nommé [default.vue] qui est utilisé. Dans ce projet, celui-ci est le suivant :

```
1.  <template>
2.    <div>
3.      <nuxt />
4.    </div>
5.  </template>
6.
7.  <style>
8.  html {
9.    font-family: 'Source Sans Pro', -apple-system, BlinkMacSystemFont, 'Segoe UI',
10.     Roboto, 'Helvetica Neue', Arial, sans-serif;
11.   font-size: 16px;
12.   word-spacing: 1px;
13.   -ms-text-size-adjust: 100%;
14.   -webkit-text-size-adjust: 100%;
15.   -moz-osx-font-smoothing: grayscale;
16.   -webkit-font-smoothing: antialiased;
17.   box-sizing: border-box;
18. }
19.
20. *,
21. *:before,
22. *:after {
23.   box-sizing: border-box;
24.   margin: 0;
25. }
26.
27. .button--green {
```

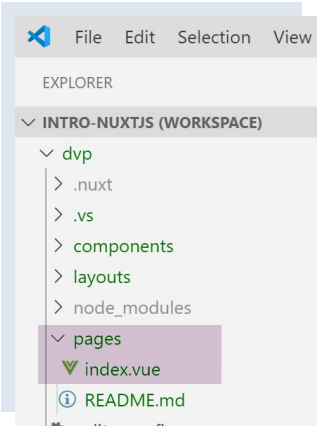
```
28.   display: inline-block;
29.   border-radius: 4px;
30.   border: 1px solid #3b8070;
31.   color: #3b8070;
32.   text-decoration: none;
33.   padding: 10px 30px;
34. }
35.
36. .button--green:hover {
37.   color: #fff;
38.   background-color: #3b8070;
39. }
40.
41. .button--grey {
42.   display: inline-block;
43.   border-radius: 4px;
44.   border: 1px solid #35495e;
45.   color: #35495e;
46.   text-decoration: none;
47.   padding: 10px 30px;
48.   margin-left: 15px;
49. }
50.
51. .button--grey:hover {
52.   color: #fff;
53.   background-color: #35495e;
54. }
55. </style>
```

Commentaires

- lignes 1-5 : le [template] du composant ;
- ligne 3 : la balise <**nuxt** /> désigne la page courante du routage ;
- lignes 7-55 : le style embarqué par le composant de mise en page. Comme celui-ci contient la page courante du routage, ce style va s'appliquer à toutes les pages routées de l'application ;

On voit que le but premier de la page [default.vue] est ici d'appliquer un style aux pages routées.

3.5 Le dossier [pages]



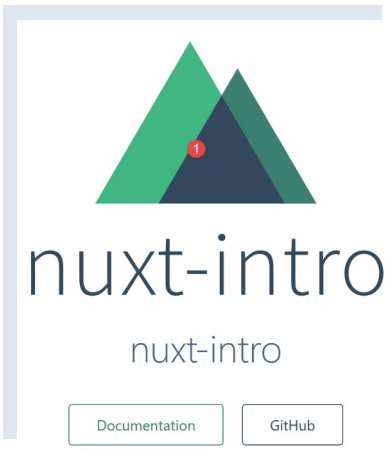
Le dossier [pages] contient les vues routées, celles que voit l'utilisateur. La page [index.vue] est la page d'accueil de l'application. Avec [nuxt.js], il n'y a pas de fichier de routage. Les routes sont déterminées à partir de la structure du dossier [pages]. Ici la présence d'un fichier [index.vue] va automatiquement créér une route appelée [index] et de chemin [/index] ramené à [/] puisqu'il s'agit de la page d'accueil. Ainsi la route suivante est créée :

```
{ name : 'index', path : '/'}
```

Le fichier [index.vue] est ici le suivant :

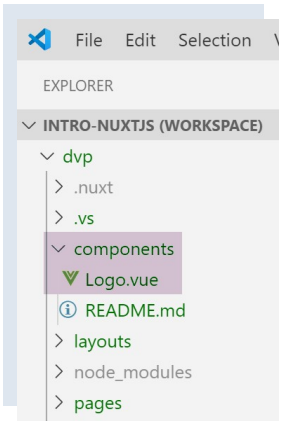
```
1.  <template>
2.    <div class="container">
3.      <div>
4.        <logo />
5.        <h1 class="title">
6.          nuxt-intro
7.        </h1>
8.        <h2 class="subtitle">
9.          nuxt-intro
10.       </h2>
11.       <div class="links">
12.         <a href="https://nuxtjs.org/" target="_blank" class="button--green">
13.           Documentation
14.         </a>
15.         <a
16.           href="https://github.com/nuxt/nuxt.js"
17.           target="_blank"
18.           class="button--grey"
19.         >
20.           GitHub
21.         </a>
22.       </div>
23.     </div>
24.   </div>
25. </template>
26.
27. <script>
28. import Logo from '~/components/Logo.vue'
29.
30. export default {
31.   components: {
32.     Logo
33.   }
34. }
35. </script>
36.
37. <style>
38. .container {
39.   margin: 0 auto;
40.   min-height: 100vh;
41.   display: flex;
42.   justify-content: center;
43.   align-items: center;
44.   text-align: center;
45. }
46.
47. .title {
48.   font-family: 'Quicksand', 'Source Sans Pro', -apple-system, BlinkMacSystemFont,
49.     'Segoe UI', Roboto, 'Helvetica Neue', Arial, sans-serif;
50.   display: block;
51.   font-weight: 300;
52.   font-size: 100px;
53.   color: #35495e;
54.   letter-spacing: 1px;
55. }
56.
57. .subtitle {
58.   font-weight: 300;
59.   font-size: 42px;
60.   color: #526488;
61.   word-spacing: 5px;
62.   padding-bottom: 15px;
63. }
64.
65. .links {
66.   padding-top: 15px;
67. }
68. </style>
```

Le [template] des lignes 1-25 affichent la vue suivante :



L'image [1] est générée par la ligne 4 du [template]. On voit donc que la page utilise un composant appelé [logo]. Celui-ci est défini aux lignes 27-35 du script de la page. Ligne 28, la notation [~] désigne la racine du projet.

3.6 Le composant [Logo]



Le composant [Logo.vue] est le suivant :

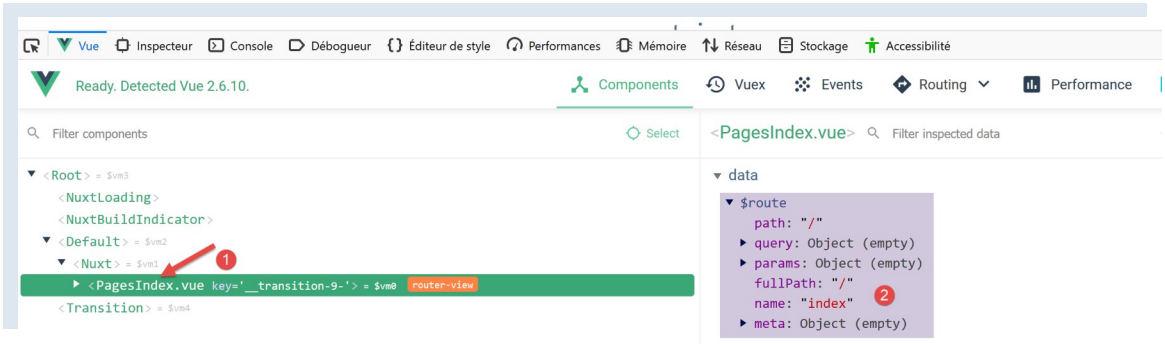
```
1.  <template>
2.    <div class="VueToNuxtLogo">
3.      <div class="Triangle Triangle--two" />
4.      <div class="Triangle Triangle--one" />
5.      <div class="Triangle Triangle--three" />
6.      <div class="Triangle Triangle--four" />
7.    </div>
8.  </template>
9.
10. <style>
11. .VueToNuxtLogo {
12.   display: inline-block;
13.   animation: turn 2s linear forwards 1s;
14.   transform: rotateX(180deg);
15.   position: relative;
16.   overflow: hidden;
17.   height: 180px;
18.   width: 245px;
19. }
20.
21. .Triangle {
22.   position: absolute;
23.   top: 0;
```

```
24.   left: 0;
25.   width: 0;
26.   height: 0;
27. }
28.
29. .Triangle--one {
30.   border-left: 105px solid transparent;
31.   border-right: 105px solid transparent;
32.   border-bottom: 180px solid #41b883;
33. }
34.
35. .Triangle--two {
36.   top: 30px;
37.   left: 35px;
38.   animation: goright 0.5s linear forwards 3.5s;
39.   border-left: 87.5px solid transparent;
40.   border-right: 87.5px solid transparent;
41.   border-bottom: 150px solid #3b8070;
42. }
43.
44. .Triangle--three {
45.   top: 60px;
46.   left: 35px;
47.   animation: goright 0.5s linear forwards 3.5s;
48.   border-left: 70px solid transparent;
49.   border-right: 70px solid transparent;
50.   border-bottom: 120px solid #35495e;
51. }
52.
53. .Triangle--four {
54.   top: 120px;
55.   left: 70px;
56.   animation: godown 0.5s linear forwards 3s;
57.   border-left: 35px solid transparent;
58.   border-right: 35px solid transparent;
59.   border-bottom: 60px solid #fff;
60. }
61.
62. @keyframes turn {
63.   100% {
64.     transform: rotateX(0deg);
65.   }
66. }
67.
68. @keyframes godown {
69.   100% {
70.     top: 180px;
71.   }
72. }
73.
74. @keyframes goright {
75.   100% {
76.     left: 70px;
77.   }
78. }
79. </style>
```

Ce composant est essentiellement constitué de styles et d'animations pour créer une image animée.

3.7 Vue DevTools

[Vue DevTools] est l'extension de navigateur qui permet d'inspecter les objets [nuxt.js] et [vue.js] dans le navigateur. Nous l'avons déjà utilisée dans le chapitre sur [vue.js]. Examinons ce que cet outil trouve lorsque la page d'accueil de notre application est affichée :



- en [1], le composant [PagesIndex] désigne la page [pages/index.vue] ;
- on voit en [2] que ce composant a une propriété [$route] qui est la route qui a amené à la page [index] ;

Comme simple exercice, affichons cette route dans la console.

3.8 Modification de la page d'accueil

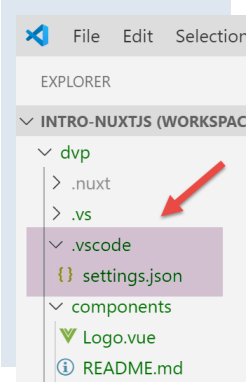
Nous allons modifier le fichier [index.vue]. Dans notre installation du projet, nous avons installé deux dépendances :

- [eslint] : qui vérifie la syntaxe des fichiers Javascript et des composants Vue. Si l'extension [ESLint] de VSCode a été installée, cette syntaxe est vérifiée lors de la frappe des textes et les erreurs sont immédiatement signalées ;
- [prettier] : qui formate les codes Javascript d'une façon standard ;

Ces dépendances sont inscrites dans le fichier [package.json] :

```
1.  "devDependencies": {
2.      "@nuxtjs/eslint-config": "^1.0.1",
3.      "@nuxtjs/eslint-module": "^1.0.0",
4.      "babel-eslint": "^10.0.1",
5.      "eslint": "^6.1.0",
6.      "eslint-config-prettier": "^4.1.0",
7.      "eslint-plugin-nuxt": ">=0.4.2",
8.      "eslint-plugin-prettier": "^3.0.1",
9.      "prettier": "^1.16.4"
10. }
```

J'ai pu remarquer (nov 2019) qu'avec l'installation faite par la commande [yarn create nuxt-app], les outils [eslint, prettier] ne fonctionnent pas lors de la frappe des textes. Les erreurs ne sont signalées qu'à la compilation. Après quelques recherches, j'ai trouvé une configuration qui marche :



On installe à la racine du projet, un dossier [.vscode] avec dedans le fichier [settings.json] suivant :

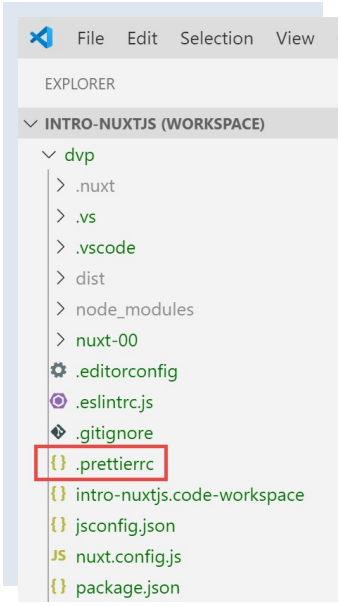
```
1.  {
2.    "eslint.validate": [
3.      {
4.        "language": "vue",
5.        "autoFix": true
6.      },
7.      {
8.        "language": "javascript",
9.        "autoFix": true
10.     }
11.   ],
12.   "eslint.autoFixOnSave": true,
13.   "editor.formatOnSave": false
14. }
```

- lignes 2-11 : indiquent que lorsque [eslint] valide les fichiers .vue et .js il doit corriger les erreurs qu'il peut corriger ;
- ligne 12 : lorsqu'un fichier est sauvegardé, [eslint] doit corriger les erreurs qu'il peut corriger ;
- ligne 13 : inhibe le formatage fait par défaut dans VSCode lors d'une sauvegarde. C'est [prettier] qui le fera ;

Avec cette configuration :

- les erreurs de syntaxe ou de formatage sont signalées dès la frappe des textes ;
- les erreurs de formatage sont automatiquement corrigées lors de la sauvegarde du fichier ;

La bibliothèque [prettier] est configurée par le fichier [.prettierrc] :



Ce fichier est par défaut le suivant :

```
1.  {
2.    "semi": false,
3.    "arrowParens": "always",
4.    "singleQuote": true
5.  }
```

- ligne 1 : pas de ; à la fin des instructions ;
- ligne 2 : si une fonction 'flèche' (arrow) a un unique paramètre, celui-ci est entouré de parenthèses ;
- ligne 3 : les chaînes de caractères sont entourées d'apostrophes (pas de guillemets) ;

Nous ajoutons les deux règles suivantes :

```
1.  {
2.    "semi": false,
3.    "arrowParens": "always",
4.    "singleQuote": true,
5.    "printWidth": 120,
6.    "endOfLine": "auto"
7.  }
```

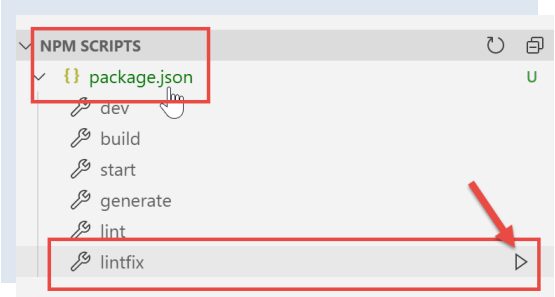
- ligne 5 : la ligne de code peut faire jusqu'à 120 caractères ;
- ligne 6 : la marque de fin de ligne peut être indifféremment CRLF (windows) ou LF (unix) ;

Enfin, le fichier [package.json] est modifié de la façon suivante :

```
1.  "scripts": {
2.      "dev": "nuxt",
3.      "build": "nuxt build",
4.      "start": "nuxt start",
5.      "generate": "nuxt generate",
6.      "lint": "eslint --ext .js,.vue --ignore-path .gitignore .",
7.      "lintfix": "eslint --fix --ext .js,.vue --ignore-path .gitignore ."
8.  },
```

- ligne 7 : nous ajoutons la commande [lintfix] qui est identique à la commande [lint] de la ligne 6 si ce n'est qu'elle a en plus le paramètre [--fix]. La commande [lint] vérifie la syntaxe et le format de tous les fichiers du projet et signale toute erreur. [lintfix] fera la même chose si ce n'est que les problèmes de formatage qui peuvent être corrigés le seront automatiquement. [lintfix] sera la commande à utiliser si la compilation échoue à cause de problèmes de formatage de fichiers ;

Ceci fait, nous modifions le fichier [index.vue] de la façon suivante :

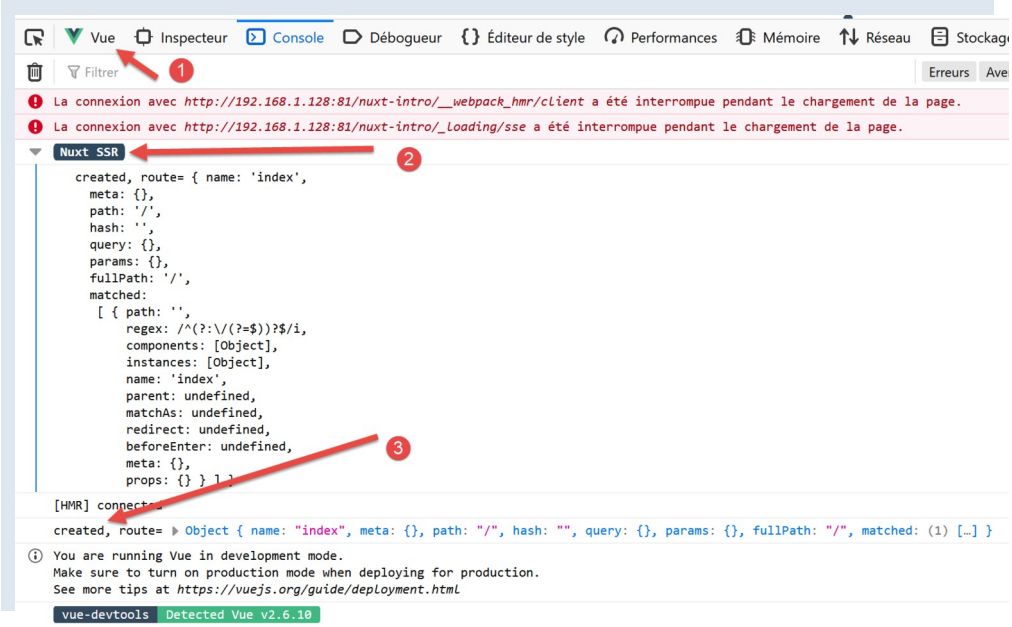



```
1.  <script>
2.  /* eslint-disable no-console */
3.  import Logo from '~/components/Logo.vue'
4.
5.  export default {
6.    components: {
7.      Logo
8.    },
9.    // cycle de vie
10.   created() {
11.     console.log('created, route=', this.$route)
12.   }
13. }
14. </script>
```

- lignes 10-12 : on ajoute la fonction [created] qui est automatiquement exécutée lorsque le composant a été créé ;
- ligne 11 : on affiche la route courante ;
- ligne 2 : un commentaire destiné à [eslint]. Sans ce commentaire, [eslint] signale une erreur ligne 11 : il ne veut pas d'instructions [console] dans les fonctions du cycle de vie. [eslint] est configurable. Nous allons garder sa configuration par défaut et nous utiliserons des commentaires tels que celui de la ligne 2 pour désactiver une règle précise de [eslint]. Nous utiliserons deux types de commentaires :
    - /* désactivation règle [eslint] */ : désactivation d'une règle pour tout le fichier ;
    - // désactivation règle [eslint] : désactivation d'une règle pour la ligne qui suit ;

Lors de la frappe, les erreurs sont signalées et une fonction [Quick Fix] disponible :

On exécute le projet :



- en [1], l'onglet [Vue] des outils de développement du navigateur (F12) ;
- en [2] et [3], l'affichage de la route ;

Pourquoi deux affichages et non un seul ?

Une application [nuxt] se décompose de deux éléments, un serveur et un client :

1. le serveur fournit les pages de l'application au démarrage de celle-ci et puis à chaque fois qu'une page est rafraîchie dans le navigateur (F5) ou bien que l'utilisateur tape une URL de l'application à la main ;
2. chaque page fournie par le navigateur contient la page demandée ainsi que le code Javascript de toute l'application qui est ensuite exécutée sur le navigateur. C'est le client. Tant qu'il n'y a pas rafraîchissement de page sur le navigateur, l'application fonctionne comme une application Vue classique en mode [sap] (Single Page Application). Dès que l'utilisateur provoque manuellement un rafraîchissement de page, celle-ci est demandée au serveur et on retourne à la phase 1 précédente.

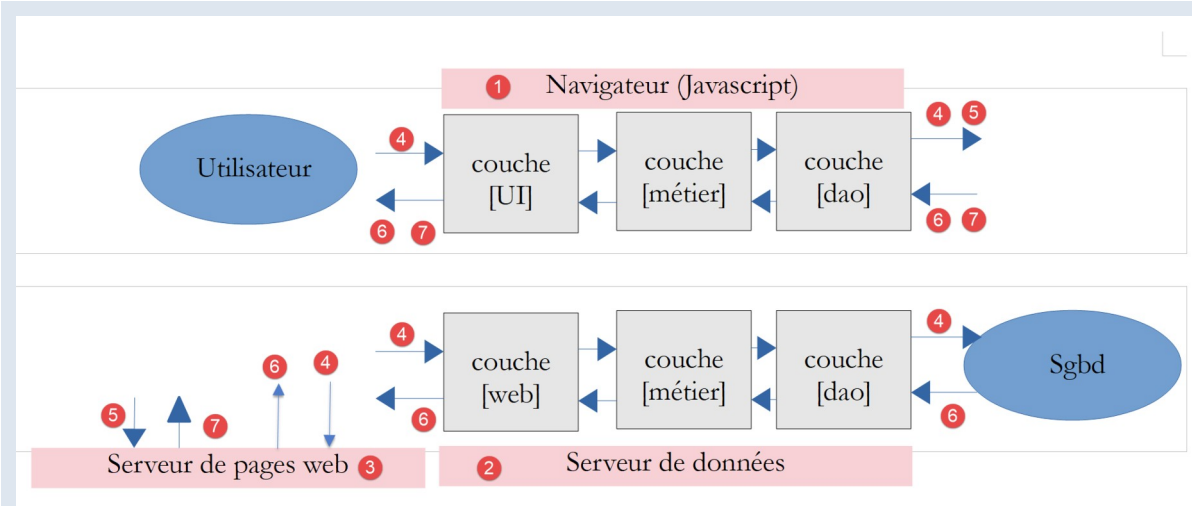
Ce qu'il faut comprendre, c'est que ce sont les **mêmes pages** du dossier [pages] qui sont fournies par le serveur ou le client. Pour cette raison, les concepteurs de [nuxt] appellent ce type de pages, des pages **isomorphiques**. Les mêmes pages [.vue] peuvent être interprétées à la fois par le client et le serveur. Prenons l'exemple de la page [index] :

```
1.  <template>
2.    <div class="container">
3.      <div>
4.        <logo />
5.        <h1 class="title">
6.          nuxt-intro
7.        </h1>
8.        <h2 class="subtitle">
9.          nuxt-intro
10.       </h2>
11.       <div class="links">
12.         <a href="https://nuxtjs.org/" target="_blank" class="button--green">
13.           Documentation
14.         </a>
15.         <a
16.           href="https://github.com/nuxt/nuxt.js"
17.           target="_blank"
18.           class="button--grey"
19.         >
20.           GitHub
21.         </a>
22.       </div>
23.     </div>
```

```
  </div>
</template>

<script>
/* eslint-disable no-console */
import Logo from '~/components/Logo.vue'

export default {
  components: {
    Logo
  },
  // cycle de vie
  created() {
    console.log('created, route=', this.$route)
  }
}
</script>
```

Comme c'est la page d'accueil, au démarrage de l'application elle est servie par le serveur. La page sur le serveur a également un cycle de vie, le même que celle d'une page [Vue] classique sauf pour les fonctions [beforeMount, monted] qui n'existent pas côté serveur. La fonction [created] est elle exécutée ce qui explique le 1er log. Cela signifie au passage que le serveur est capable d'exécuter des scripts Javascript. Ici et en général, ce serveur est un serveur [node.js]. Une fois la page créée sur le serveur, elle arrive sur le navigateur où elle subit de nouveau le cycle de vie. La fonction [created] est exécutée une seconde fois, ce qui donne le 2ième log.

L'architecture d'une application [nuxt] pourrait être la suivante :



- [1] : le navigateur qui héberge l'application [nuxt] lorsque celle-ci a été chargée sur le navigateur. C'est ce qu'on a appelé le client [nuxt] ;
- [3] : le serveur qui héberge initialement l'application [nuxt]. Celle-ci est chargée sur le navigateur [1] au démarrage de l'application et à chaque fois que l'utilisateur rafraîchit la page courante du navigateur ou tape à la main une URL de l'application. C'est là que se situe la différence de fonctionnement avec une application Vue classique. Avec celle-ci, une fois chargée sur le navigateur, le serveur n'était plus jamais sollicité par la suite. Une autre différence importante qu'on n'a pas pu voir pour l'instant est que le serveur d'une application Vue est un serveur statique, incapable d'interpréter les pages [.vue], alors que celui d'une application Nuxt de type [universal] est un serveur Javascript. Avant d'envoyer une page au navigateur, le serveur peut exécuter des scripts et aller par exemple chercher des données sur le serveur [2] ;
- [2] : est le serveur qui fournit des données soit au client [nuxt] [1], soit au serveur [nuxt] [3] ;

On peut dans le schéma ci-dessus distinguer trois sous-systèmes client / serveur :

- [1, 3] : héberge l'application [nuxt]. [3] la fournit au démarrage de l'application avec la page d'accueil et à chaque fois que l'utilisateur demande une page manuellement. [1] héberge l'application [nuxt] reçue de [3] qui fonctionne alos en mode [SAP] tant que les pages ne sont pas demandées manuellement à [3] ;
- [1, 2] : en mode [SAP], le client [nuxt] récupère des données externes auprès d'un ou plusieurs serveurs ;

- [3, 2] : lors de la génération de la page demandée par l'utilisateur, le serveur [3] peut lui aussi récupérer des données externes auprès d'un ou plusieurs serveurs ;

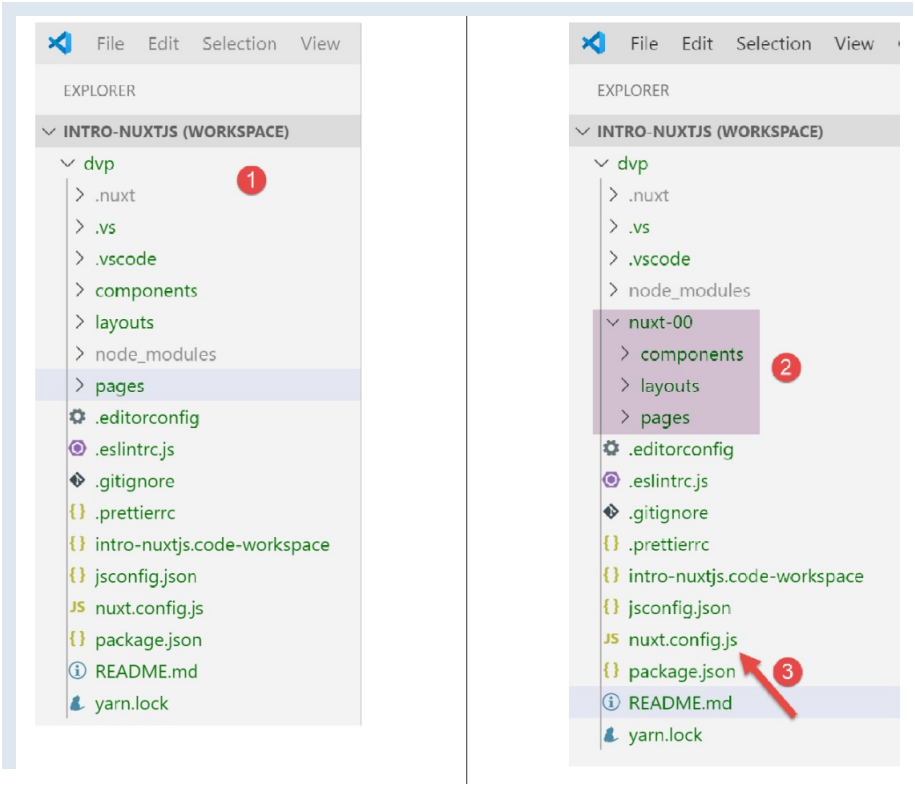
C'est donc le serveur [3] qui distingue une application [nuxt] d'une application [vue]. Ce serveur est sollicité à chaque fois que l'utilisateur demande une page manuellement. Il traite les mêmes pages [.vue] que le client [vue] [1]. C'est un serveur Javascript capable d'exécuter les scripts présents dans la page. Cela peut modifier par exemple la façon de générer la page d'accueil avec des données externes : là où une application [vue] obtient celles-ci forcément à partir du client [1], ici elles peuvent être obtenues par le serveur [3] avant que la page ne soit envoyée au client. La page d'accueil devient ainsi signifiante et peut contribuer à améliorer le SEO de l'application.

Note : en mode développement les trois entités [1, 2, 3] sont souvent sur la même machine. Ce sera le cas ici pour tous nos exemples.

3.9 Déplacement du code source de l'application dans un dossier séparé

Par la suite, nous allons créer diverses applications [nuxt] dans le même dossier [dvp]. En effet, le dossier des dépendances [node_modules] généré pour chaque projet [nuxt] peut faire plusieurs centaines de méga-octets. On va créer divers dossiers [nuxt-00, nuxt-01, ...] dans le dossier [dvp] pour contenir le code source des exemples à tester. Puis nous utiliserons le fichier de configuration [nuxt-config.js] pour indiquer où se trouve le code source du projet [dvp] qui restera l'unique projet [nuxt] de ce tutoriel.

Nous déplaçons le code source de l'application générée initialement par la commande [yarn create nuxt-app] dans un dossier [nuxt-00] :



- en [2], on a déplacé les dossiers [components, layouts, pages] dans un dossier [nuxt-00] ;
- en [3], il nous faut modifier le fichier [nuxt.config.js] ;

Nous modifions le le fichier [nuxt.config.js] de la façon suivante :

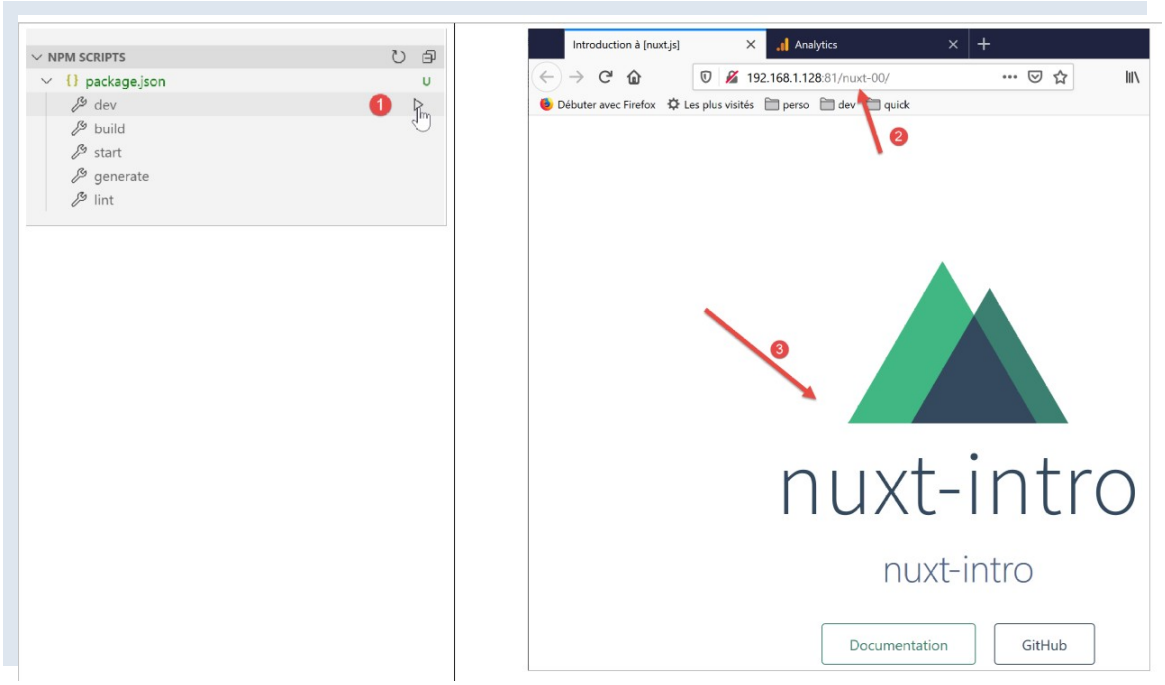
```
1.  export default {
2.    mode: 'universal',
3.    /*
4.     ** Headers of the page
5.     */
```

```
6.    ...
7.    /*
8.     ** Build configuration
9.     */
10.   build: {
11.     /*
12.      ** You can extend webpack config here
13.      */
14.     extend(config, ctx) {}
15.   },
16.   // répertoire du code source
17.   srcDir: 'nuxt-00',
18.   // routeur
19.   router: {
20.     // racine des URL de l'application
21.     base: '/nuxt-00/'
22.   },
23.   // serveur
24.   server: {
25.     // port de service, 3000 par défaut
26.     port: 81,
27.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
28.     // 0.0.0.0 = toutes les adresses réseau de la machine
29.     host: '0.0.0.0'
30.   }
31. }
```

Le fichier est modifié en deux points :

- ligne 17 : on indique que le code source du projet [dvp] est à trouver dans le dossier [nuxt-00] ;
- ligne 21 : on indique que l'URL racine de l'application est désormais [/nuxt-00/]. Ce changement n'était pas obligatoire. On pourrait ne pas mettre cette propriété et la racine des URL serait alors [/]. Ici, cela nous permettra de nous souvenir que le code source exécuté est celui du dossier [nuxt-00] ;

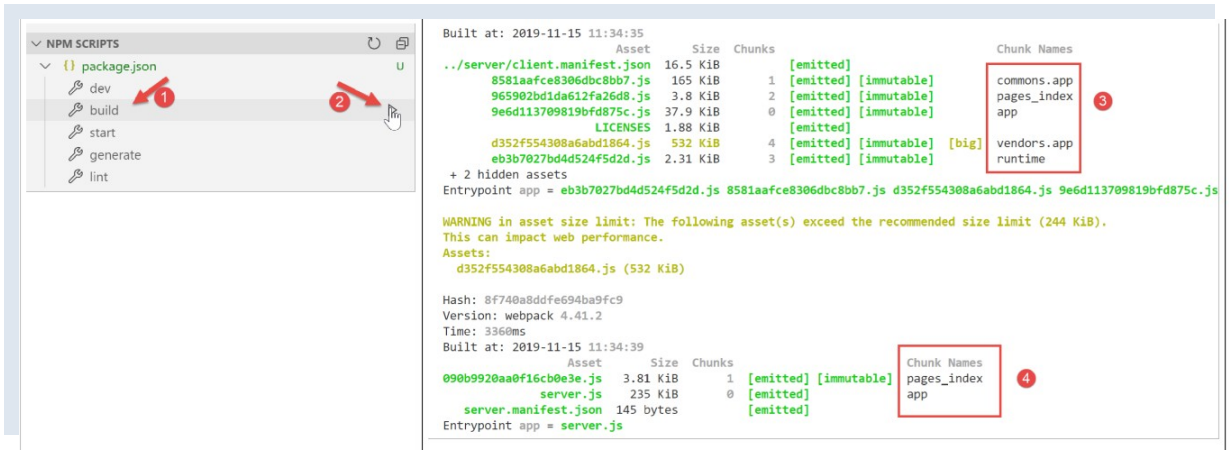
Ceci fait, le projet [dvp] est exécuté comme précédemment :



3.10 Déploiement de l'application [nuxt-00]

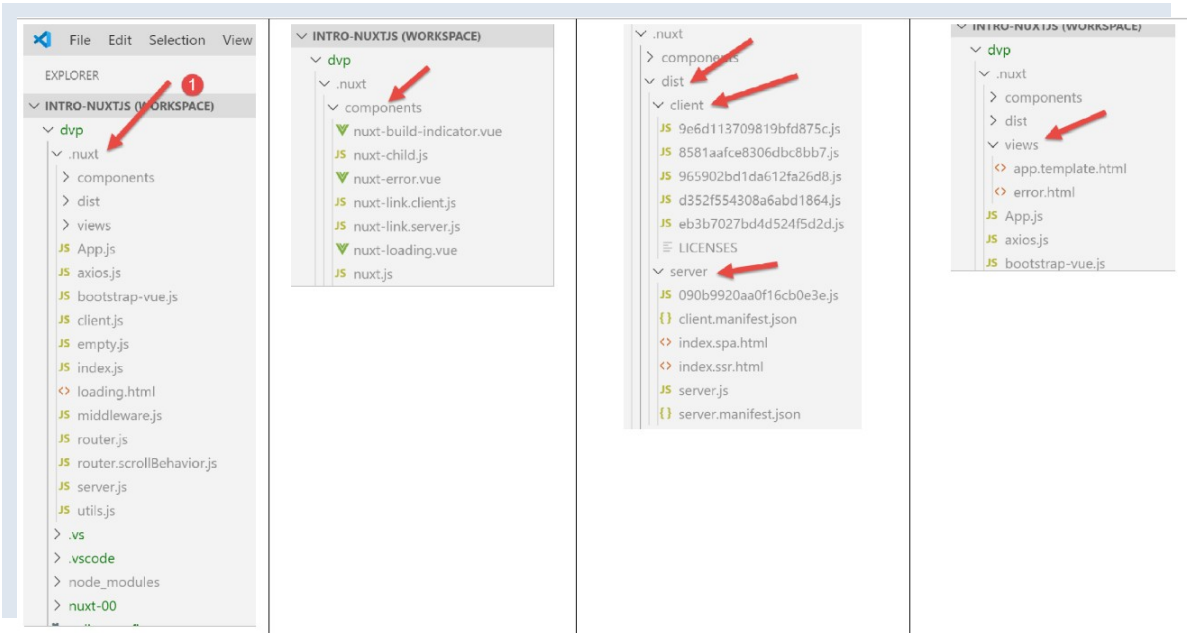
Nous allons exécuter l'application [nuxt-00] dans un environnement autre que l'environnement intégré de VSCode.

Tout d'abord nous compilons l'application :

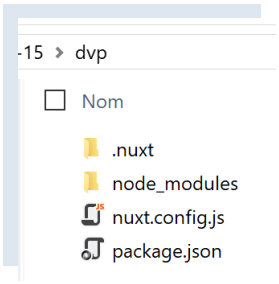


- en [3], le résultat de la compilation du client. Sera exécuté par le navigateur ;
- en [4], le résultat de la compilation du serveur. Sera exécuté par le serveur [node.js] ;

Le résultat de la compilation est placé dans le dossier [.nuxt] :



Nous copions les dossiers [.nuxt, node_modules] et les fichiers [package.json, nuxt.config.js] dans un dossier séparé :



Le fichier [package.json] est simplifié de la façon suivante :

```
1.  {
2.    "scripts": {
3.      "start": "nuxt start"
4.    }
5.  }
```

- on ne garde que le script [start] qui permet d'exécuter la version compilée du projet ;

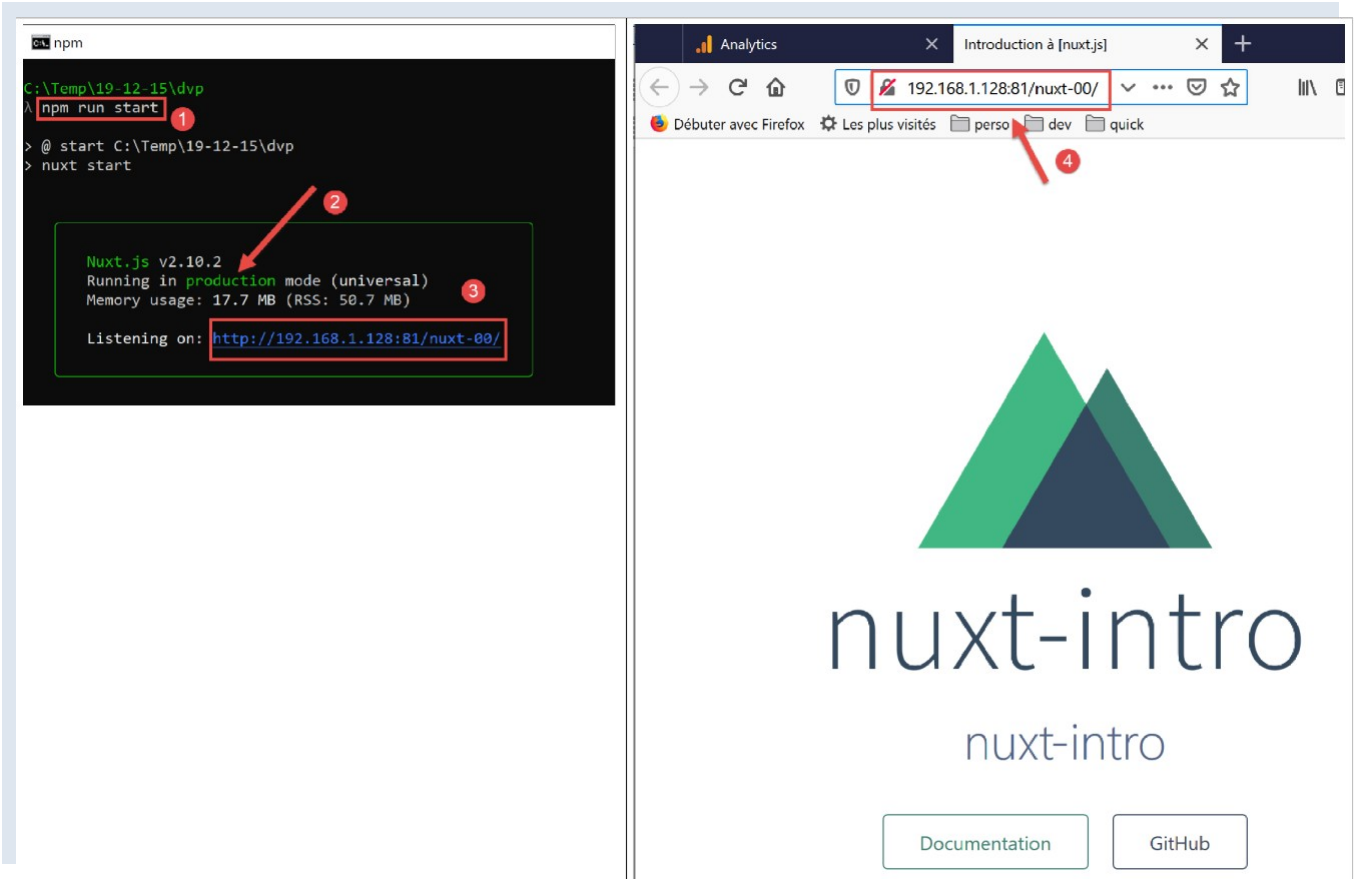
Le fichier [nuxt.config.js] est simplifié de la façon suivante :

```
1.  export default {
2.    // routeur
3.    router: {
4.      // racine des URL de l'application
5.      base: '/nuxt-00/'
6.    },
7.    // serveur
8.    server: {
9.      // port de service, 3000 par défaut
10.     port: 81,
11.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
12.     // 0.0.0.0 = toutes les adresses réseau de la machine
13.     host: '0.0.0.0'
14.   }
15. }
```

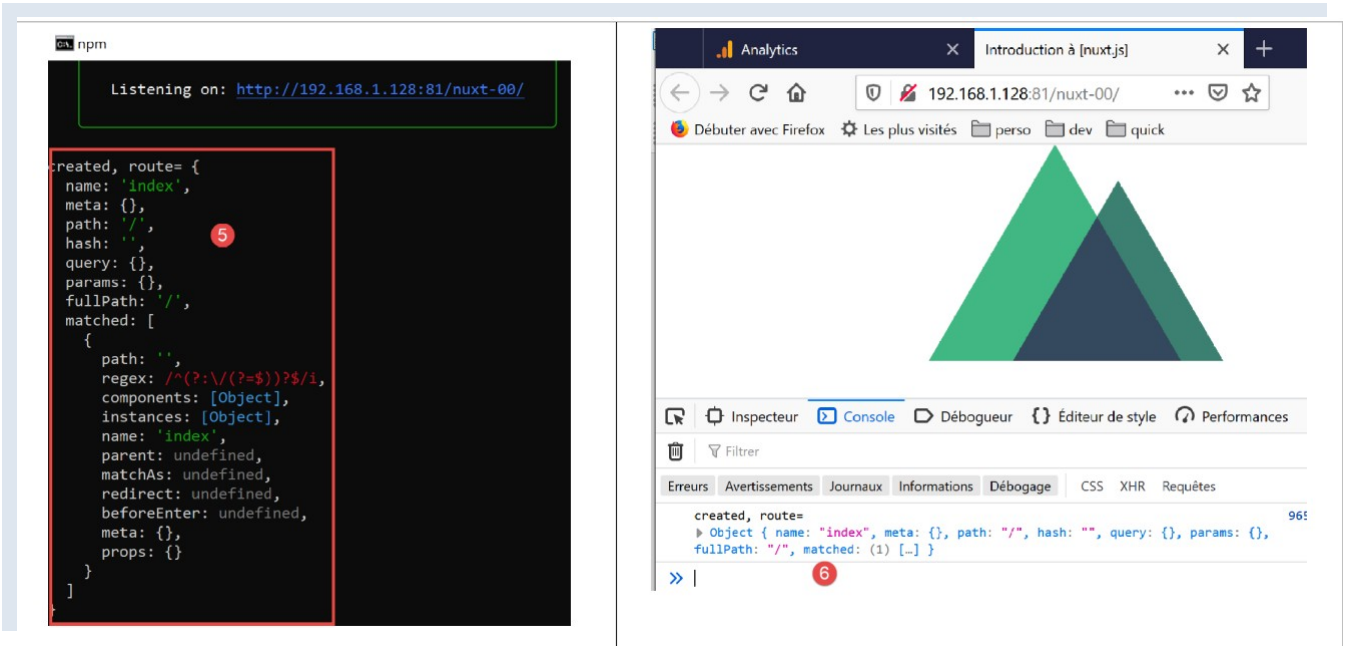
- ligne 5 : on fixe l'URL de base de l'application compilée ;
- lignes 8-14 : on définit le port de service et les adresses réseau écoutées ;

Ceci fait, on ouvre un terminal Laragon et on se positionne dans le dossier contenant la version compilée du projet. On peut ouvrir tout type de terminal mais il faut que l'exécutable [npm] soit dans le PATH du terminal. C'est le cas pour le terminal Laragon.

Ceci fait, on tape la commande [npm run start] :



En [3], on voit qu'un serveur a été lancé et qu'il écoute à l'URL [http://192.168.1.128:81/nuxt-00/]. Maintenant demandons cette URL avec un navigateur [4]. On a bien la même chose qu'auparavant. Côté terminal, des logs ont été écrits [5]. C'est le log placé dans la méthode [created] de la page [index.vue] qui a été exécutée par le serveur [node.js].



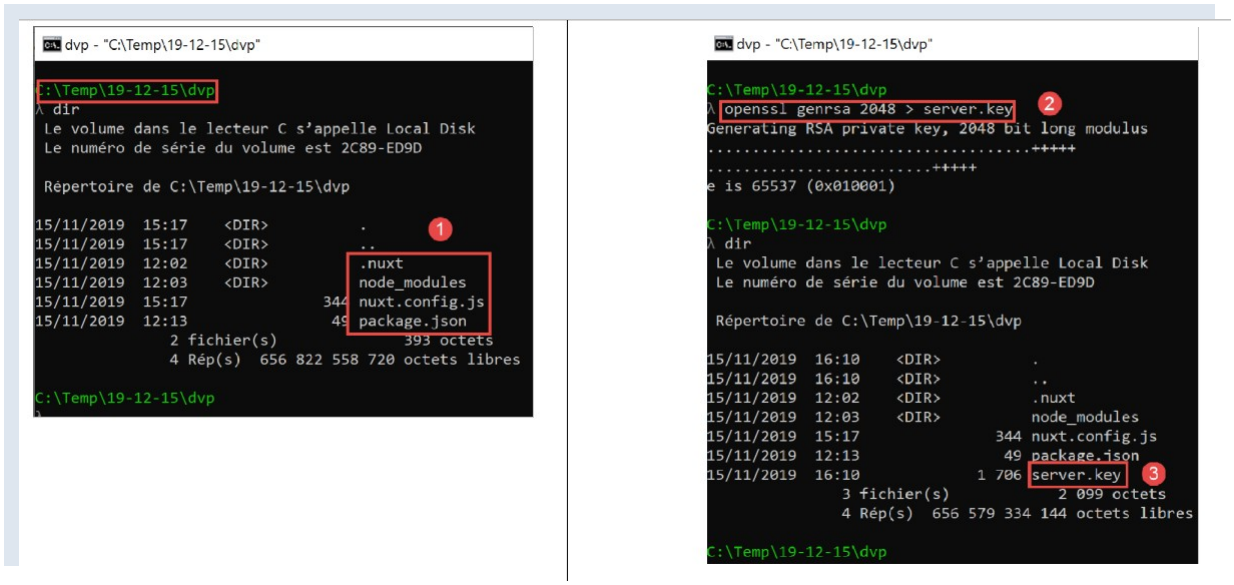
Côté navigateur [6], on retrouve également le log de la méthode [created] de la page [index.vue] mais exécutée cette fois par le client.

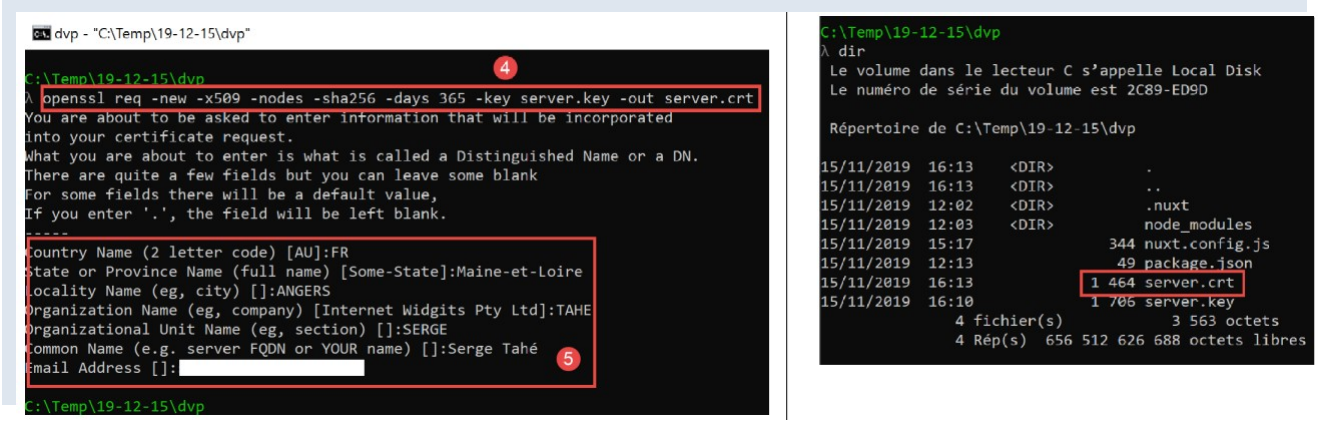
3.11 Mise en place d'un serveur sécurisé

Ci-dessus, l'URL de l'application est [**http**://192.168.1.128/nuxt-00/]. On voudrait qu'elle soit [**https**://192.168.1.128/nuxt-00/]. Il nous faut donc construire un serveur sécurisé. Nous montrons comment procéder.

Note : la méthode a été tirée de l'article [https://stackoverflow.com/questions/56966137/how-to-run-nuxt-npm-run-dev-with-https-in-localhost].

Tout d'abord nous créons une clé privée et une clé publique avec [openssl]. [openssl] est normalement installé en même temps que le serveur Laragon. Du coup, cette commande est disponible dans tout terminal Laragon. Ouvrons donc un terminal Laragon et positionnons-nous sur le dossier de l'application déployée :





- en [2], on tape la commande [openssl genrsa 2048 > server.key] ;
- en [3], un fichier [server.key] est créé ;
- en [4], on tape la commande [openssl req -new -x509 -nodes -sha256 -days 365 -key server.key -out server.crt] ;
- en [5], un fichier [server.crt] est créé ;

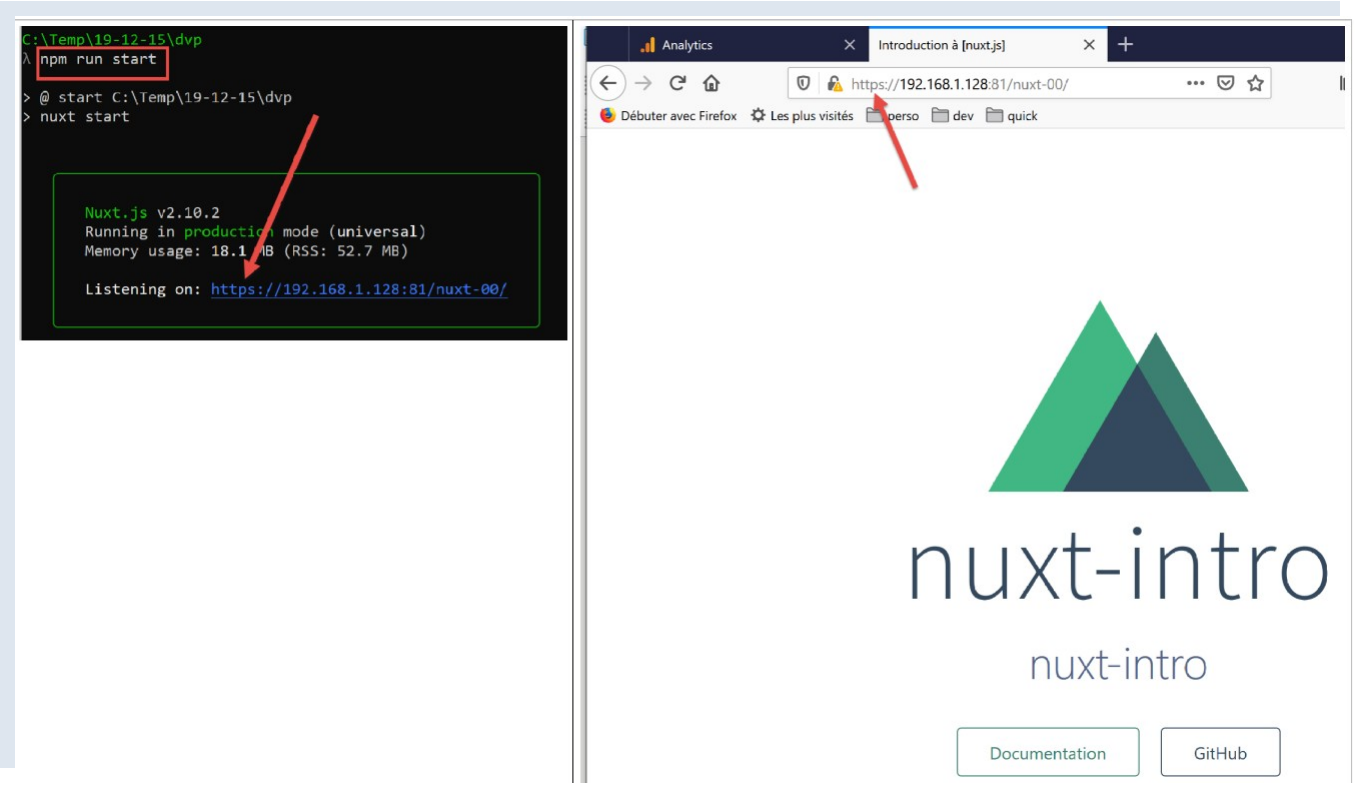
Ces deux fichiers constituent un certificat autosigné. La plupart des navigateurs ne les acceptent qu'après approbation de l'utilisateur ayant demandé la page.

Les fichiers [server.key, server.crt] doivent être maintenant utilisés par l'application web. Pour cela le fichier [nuxt.config.js] doit être modifié de la façon suivante :

```
1.  import path from 'path'
2.  import fs from 'fs'
3.
4.  export default {
5.    // routeur
6.    router: {
7.      // racine des URL de l'application
8.      base: '/nuxt-00/'
9.    },
10.   // serveur
11.   server: {
12.     // port de service, 3000 par défaut
13.     port: 81,
14.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
15.     // 0.0.0.0 = toutes les adresses réseau de la machine
16.     host: '0.0.0.0',
17.     // certificat autosigné
18.     https: {
19.       key: fs.readFileSync(path.resolve(__dirname, 'server.key')),
20.       cert: fs.readFileSync(path.resolve(__dirname, 'server.crt'))
21.     }
22.   }
23. }
```

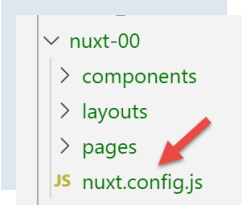
Ce sont les lignes 18-21 qui mettent en place le protocole [https].

Maintenant réexécutons l'application :



3.12 Fin du premier exemple

Le premier exemple est désormais terminé. Il nous a appris beaucoup de concepts de [nuxt]. Nous allons maintenant développer d'autres exemples que nous placerons dans des dossiers [nuxt-01, nuxt-02, ...]. Comme ces exemples utiliseront un fichier [nuxt.config.js] différent, nous sauvegarderons dans chacun de ces dossiers, le fichier [nuxt.config.js] qui a servi à les exécuter :

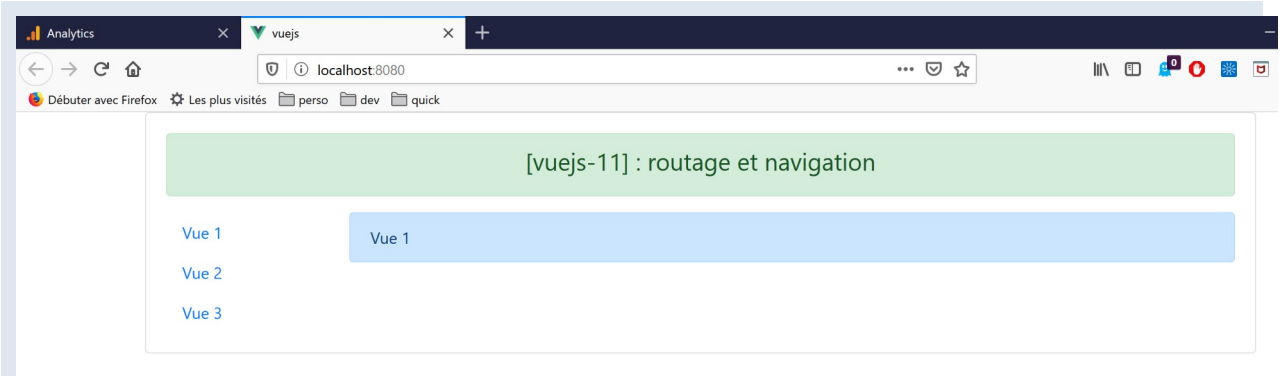

nuxt-00
components
layouts
pages
JS
nuxt.config.js

4 Exemple [nuxt-01] : routage et navigation

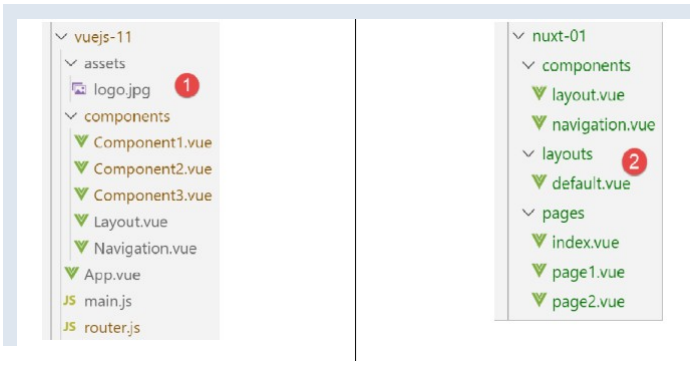
Nous allons construire une série d'exemples simples pour découvrir progressivement le fonctionnement d'une application [nuxt]. Nous allons commencer par porter l'application [vuejs-11] du document |Introduction au framework **VUE.JS** par l'exemple|, pour découvrir tout d'abord ce qui différencie l'organisation du code d'une application [nuxt] de celle du code d'une application [vue].

4.1 Arborescence du projet

Le projet [vuejs-11] était un projet de navigation entre vues :



L'arborescence du code source du projet [vuejs-11] était le suivant :



- [main.js] était le script exécuté lors du démarrage de l'application [vue] ;
- [router.js] fixait les règles de routage ;
- [App.vue] était la vue structurante de l'application. Elle organisait la mise en page des différentes vues ;
- [Component1, Component2, Component3, Layout, Navigation] étaient les composants utilisés dans les différentes vues de l'application ;

Dans le portage de l'application [vue] [1] vers une application [nuxt] [2] :

- les scripts exécutés au démarrage de l'application doivent être déclarés dans la clé [plugins] du fichier [nuxt.config.js]. Par ailleurs, il est possible de séparer les scripts destinés au serveur [nuxt] de ceux destinés au client [nuxt] ;
- la vue [App.vue] doit être installée dans le dossier [layouts] et être renommée [default.vue] ;
- les composants [Component1, Component2, Component3] qui sont les cibles du routage doivent migrer dans le dossier [pages]. L'un d'eux, celui qui sert de page d'accueil, doit être renommé [index.vue]. Nous avons ici renommé les fichiers :
  - [Component1] --> [index] : affiche le texte [Home] ;
  - [Component2] --> [page1] : affiche le texte [Page 1] ;
  - [Component3] --> [page2] : affiche le texte [Page 2] ;

  [nuxt] utilise le contenu du dossier [pages] pour générer dynamiquement les routes suivantes :

```
1.  { name :'index', 'path' :'/'}
2.  { name :'page1', 'path' :'/page1'}
3.  { name :'page2', 'path' :'/page2'}
```

Du coup, le fichier [router.js] utilisé dans le projet [vue] devient inutile dans le projet [nuxt].

Le fichier de configuration [nuxt.config.js] sera le suivant :

```
1.  export default {
2.    mode: 'universal',
3.    /*
4.     ** Headers of the page
5.     */
6.    head: {
7.      title: 'Introduction à [nuxt.js]',
8.      meta: [
9.        { charset: 'utf-8' },
10.       { name: 'viewport', content: 'width=device-width, initial-scale=1' },
11.       {
12.         hid: 'description',
13.         name: 'description',
14.         content: 'ssr routing loading asyncdata middleware plugins store'
15.       }
16.     ],
17.     link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
18.   },
19.   /*
20.    ** Customize the progress-bar color
21.    */
22.   loading: { color: '#fff' },
23.   /*
24.    ** Global CSS
25.    */
26.   css: [],
27.   /*
28.    ** Plugins to load before mounting the App
29.    */
30.   plugins: [],
31.   /*
32.    ** Nuxt.js dev-modules
33.    */
34.   buildModules: [
35.     // Doc: https://github.com/nuxt-community/eslint-module
36.     '@nuxtjs/eslint-module'
37.   ],
38.   /*
39.    ** Nuxt.js modules
40.    */
41.   modules: [
42.     // Doc: https://bootstrap-vue.js.org
43.     'bootstrap-vue/nuxt',
44.     // Doc: https://axios.nuxtjs.org/usage
45.     '@nuxtjs/axios'
46.   ],
47.   /*
48.    ** Axios module configuration
49.    ** See https://axios.nuxtjs.org/options
50.    */
51.   axios: {},
52.   /*
53.    ** Build configuration
54.    */
55.   build: {
56.     /*
57.      ** You can extend webpack config here
58.      */
59.     extend(config, ctx) {}
60.   },
61.   // répertoire du code source
62.   srcDir: 'nuxt-01',
63.   // routeur
64.   router: {
```

```
65.     // racine des URL de l'application
66.     base: '/nuxt-01/'
67.   },
68.   // serveur
69.   server: {
70.     // port de service, 3000 par défaut
71.     port: 81,
72.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
73.     // 0.0.0.0 = toutes les adresses réseau de la machine
74.     host: '0.0.0.0'
75.   }
76. }
```

- ligne 62 : on indique le dossier qui contient le code source du projet [dvp] ;
- ligne 66 : : on indique l'URL racine de l'application [dvp] (on peut mettre ce qu'on veut) ;
- ligne 43 : notons que la bibliothèque [bootstrap-vue] est référencée dans la configuration ;

4.2 Portage du fichier [main.js]

Le fichier [main.js] du projet [vuejs-11] était le suivant :

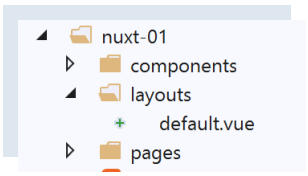
```
1.  // imports
2.  import Vue from 'vue'
3.  import App from './App.vue'
4.
5.  // plugins
6.  import BootstrapVue from 'bootstrap-vue'
7.  Vue.use(BootstrapVue);
8.
9.  // bootstrap
10. import 'bootstrap/dist/css/bootstrap.css'
11. import 'bootstrap-vue/dist/bootstrap-vue.css'
12.
13. // routeur
14. import monRouteur from './router'
15.
16. // configuration
17. Vue.config.productionTip = false
18.
19. // instanciation projet [App]
20. new Vue({
21.   name: "app",
22.   // vue principale
23.   render: h => h(App),
24.   // routeur
25.   router: monRouteur,
26. }).$mount('#app')
```

En-dehors des [imports], le code fait les choses suivantes :

- lignes 5-11 : utilisation de la bibliothèque [bootstrap-vue]. Ce travail est désormais fait par le module [`bootstrap-vue/nuxt`] de la ligne 43 du fichier de configuration [nuxt.config.js] ;
- lignes 14 et 25 : utilisation du fichier de routage [router.js]. Ce travail est désormais fait automatiquement par l'application [nuxt] à partir de l'arborescence du dossier [pages] ;
- lignes 20-26 : instanciation de la vue principale de l'application. Dans une application [nuxt], c'est la vue [layouts/default.vue] qui sert de vue principale ;

Le fichier [main.js] n'a désormais plus de raison d'être. S'il en avait eu une, on l'aurait déclaré dans la clé [plugins] de la ligne 30 du fichier de configuration [nuxt.config.js] ;

4.3 La vue principale [default.vue]

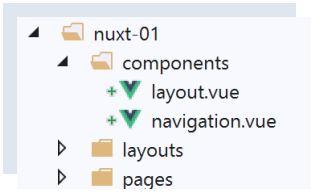


La vue principale [layouts / default.vue] est la suivante :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <!-- un message -->
5.        <b-alert show variant="success" align="center">
6.          <h4>[nuxt-01] : routage et navigation</h4>
7.        </b-alert>
8.        <!-- la vue courante du routage -->
9.        <nuxt />
10.     </b-card>
11.   </div>
12. </template>
13.
14. <script>
15. export default {
16.   name: 'App'
17. }
18. </script>
```

- ligne 9, dans le projet [vuejs-11], on avait la balise <**router-view** /> au lieu de la balise <**nuxt** /> utilisée ici. Les deux semblent utilisables. Je les ai essayées toutes les deux sans voir de changement. J'ai gardé la balise <**nuxt** /> qui est celle conseillée. Elle affiche la vue courante, ç-à-d la page cible du routage courant ;

4.4 Les composants



Par rapport au projet [vuejs-11], les composants [layout, navigation] ne changent pas :

[**components / layout.vue**]

```
1.  <!-- disposition des vues -->
2.  <template>
3.    <!-- ligne -->
4.    <div>
5.      <b-row>
6.        <!-- zone à deux colonnes -->
7.        <b-col v-if="left" cols="2">
8.          <slot name="left" />
9.        </b-col>
10.       <!-- zone à dix colonnes -->
11.       <b-col v-if="right" cols="10">
12.         <slot name="right" />
13.       </b-col>
14.     </b-row>
15.   </div>
16. </template>
17.
18. <script>
19. export default {
```

```
20.   // paramètres
21.   props: {
22.     left: {
23.       type: Boolean
24.     },
25.     right: {
26.       type: Boolean
27.     }
28.   }
29. }
30. </script>
```

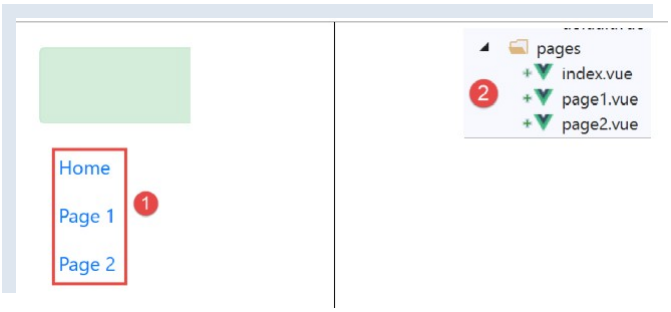
Ce composant sert à structurer les pages de l'application en deux colonnes :

- lignes 7-9 : la colonne de gauche sur 2 colonnes Bootstrap ;
- lignes 11-13 : la colonne de droite sur 10 colonnes Bootstrap ;

[**navigation.vue**]

```
1.  <template>
2.    <!-- menu Bootstrap à trois options -->
3.    <b-nav vertical>
4.      <b-nav-item to="/" exact exact-active-class="active">
5.        Home
6.      </b-nav-item>
7.      <b-nav-item to="/page1" exact exact-active-class="active">
8.        Page 1
9.      </b-nav-item>
10.     <b-nav-item to="/page2" exact exact-active-class="active">
11.       Page 2
12.     </b-nav-item>
13.   </b-nav>
14. </template>
```

Ce composant affiche trois liens de navigation :



Pour savoir quoi mettre comme valeur aux attributs [to] des lignes 4, 7 et 10, il faut regarder le dossier [pages] [2] :

- la page [index] aura l'URL [/] ;
- la page [page1] aura l'URL [/page1] ;
- la page [page2] aura l'URL [/page2] ;

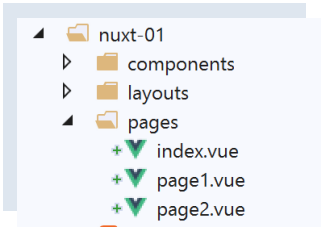
Le composant [navigation] peut être également écrit de la façon suivante :

```
1.  <template>
2.    <!-- menu Bootstrap à trois options -->
3.    <b-nav vertical>
4.      <nuxt-link to="/" exact exact-active-class="active">
5.        Home
6.      </nuxt-link>
7.      <nuxt-link to="/page1" exact exact-active-class="active">
8.        Page 1
9.      </nuxt-link>
10.     <nuxt-link to="/page2" exact exact-active-class="active">
11.       Page 2
12.     </nuxt-link>
13.   </b-nav>
```

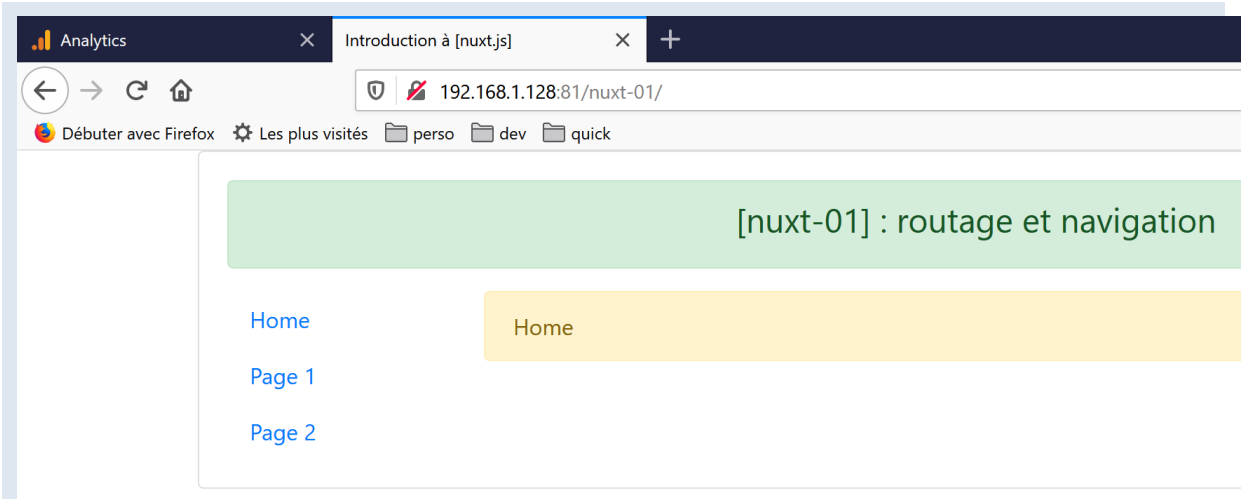
```
14. </template>
```

La balise <**b-nav-item**> est remplacée par la balise <**nuxt-link**> qui désigne un lien de routage. A l'exécution, je n'ai pas vu de grande différence, rien qui pourrait faire pencher la balance vers une balise plutôt que l'autre.

4.5 Les pages



La page [index.vue] affiche la vue suivante :



Le code de la page est le suivant :

```
1.  <!-- page principale -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="warning">
8.        Home
9.      </b-alert>
10.   </Layout>
11. </template>
12.
13. <script>
14. /* eslint-disable no-undef */
15. /* eslint-disable no-console */
16. /* eslint-disable nuxt/no-env-in-hooks */
17.
18. import Navigation from '@/components/navigation'
19. import Layout from '@/components/layout'
20.
21. export default {
22.   name: 'Home',
23.   // composants utilisés
24.   components: {
```

```
25.     Layout,
26.     Navigation
27.   },
28.   // cycle de vie
29.   beforeCreate(...args) {
30.     console.log('[home beforeCreate]', 'process.server=', process.server,
31.  'process.client=', process.client, "nombre d'arguments=", args.length)
32.   },
33.   created(...args) {
34.     console.log('[home created]', 'process.server=', process.server,
35.  'process.client=', process.client, "nombre d'arguments=", args.length)
36.   },
37.   beforeMount(...args) {
38.     console.log('[home beforeMount]', 'process.server=', process.server,
39.  'process.client=', process.client, "nombre d'arguments=", args.length)
40.   },
41.   mounted(...args) {
42.     console.log('[home mounted]', 'process.server=', process.server,
43.  'process.client=', process.client, "nombre d'arguments=", args.length)
44.   }
45. }
46. </script>
```

- ligne 5 : le composant de navigation est placé en colonne de gauche ;
- lignes 7-9 : une alerte est placée dans la colonne de droite ;

Dans la partie <script>, nous mettons du code dans les fonctions du cycle de vie de la page [beforeCreate, created, beforeMount, beforeMounted]. Nous voulons savoir lesquelles sont exécutées par le serveur []nuxt et lesquelles par le client [nuxt]. On rappelle deux choses :

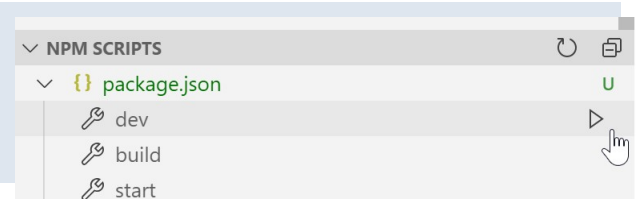
- lorsqu'une page est demandée soit au démarrage de l'application, cas de la page [index], soit manuellement par l'utilisateur qui rafraîchit la page du navigateur ou tape une URL à la main, elle est délivrée d'abord par le serveur [nuxt]. Celui-ci interprète le code ci-dessus et exécute le Javascript qu'il contient ;
- lorsque la page envoyée par le serveur [nuxt] arrive sur le navigateur, elle arrive avec le code du client [nuxt]. Celui-ci interprète de nouveau la page ci-dessus ;
- par des logs, on veut savoir qui fait quoi pour mieux comprendre ce processus ;
- lignes 30-31 : on utilise dans la fonction un objet global [process] qui existe aussi bien sur le serveur que sur le client :
  - [process.server] est vrai si le code est exécuté par le serveur, faux sinon ;
  - [process.client] est vrai si le code est exécuté par le client, faux sinon ;
  - parce que la variable [process] est non déclarée dans le code, on est obligés de mettre la ligne 14 pour [eslint]. La ligne [16] est nécessaire parce que sinon [eslint] déclare un autre type d'erreur à cause de la variable [process]. La ligne 15 est elle nécessaire pour permettre l'utilisation de [console] dans les fonctions du cycle de vie ;
- ligne 29 : on veut savoir également si les fonctions du cycle de vie reçoivent des arguments. On va découvrir en effet que [nuxt] transmet des informations à certaines fonctions. On veut savoir si les fonctions du cycle de vie en font partie ;
- on répète le même code pour les quatre fonctions ;

4.6 Le fichier [nuxt.config.js]

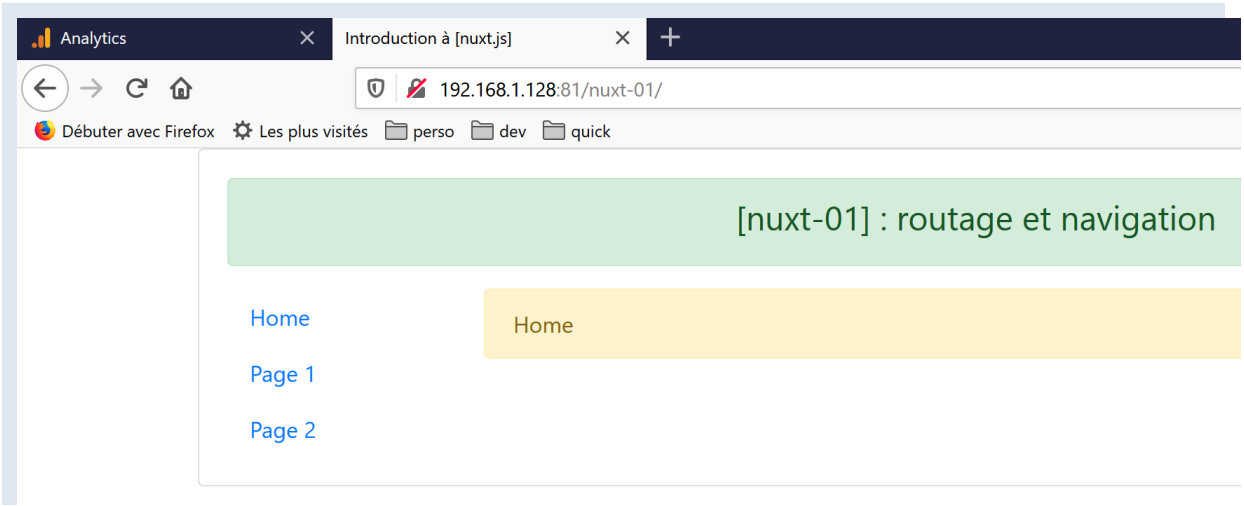
C'est lui qui contrôle l'exécution du projet [dvp]. Il a été décrit page 32.

4.7 Exécution du projet

Nous exécutons le projet :



La page affichée est la suivante :



Une fois installée sur le navigateur, l'application [nuxt] devient une application [vue] classique. Nous ne commenterons donc pas le fonctionnement client de l'application [nuxt-01]. Cela a été fait dans le projet [vuejs-11] du document | Introduction au framework **VUE.JS** par l'exemple|.

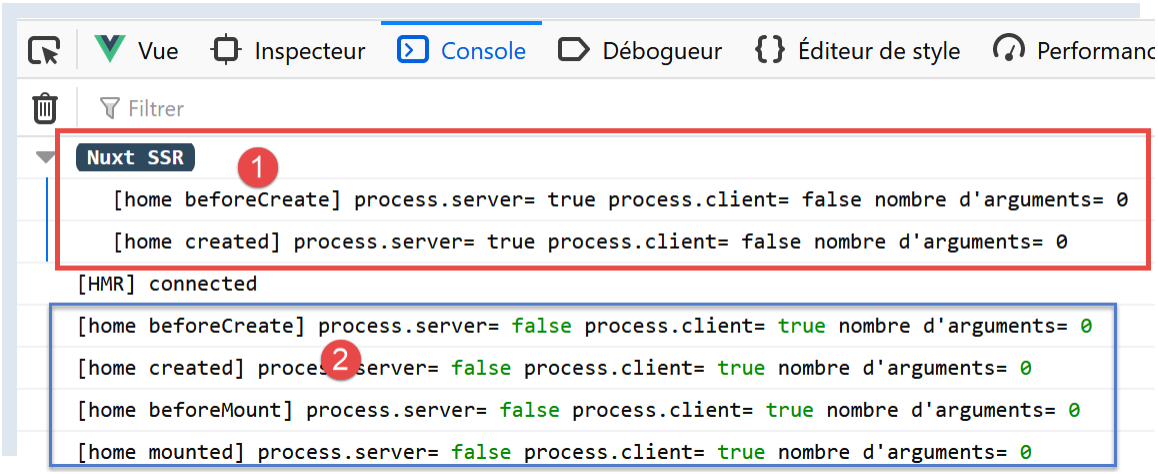
L'application [nuxt] ne diffère de l'application [vue] qu'à deux moments :

- le démarrage initial de l'application qui fournit la page d'accueil ;
- à chaque fois que l'utilisateur provoque d'une manière ou une autre le rafraîchissement du navigateur ;

Dans ces deux cas :

- la page demandée est fournie par le serveur ;
- la page reçue est traitée par le client ;

Regardons les logs du démarrage de l'application (F12 sur le navigateur) :



- en [1], les logs du serveur (process.server=true). Ils apparaissent précédés de la mention [Nuxt SSR] (SSR= Server Side Rendered) ;
- en [2], les logs du client sur le navigateur (process.client=true) ;

De ces logs, on peut déduire que :

- le **serveur** exécute les fonctions [beforeCreate, created] du cycle de vie ;
- le **client** exécute les fonctions [beforeCreate, created, beforeMount, mounted] du cycle de vie ;
- le serveur a traité la page **avant** le client ;
- dans les deux cas, aucune des fonctions exécutées ne reçoit d'arguments ;

Maintenant regardons le code source de la page reçue (option [Code source de la page] dans le navigateur) :

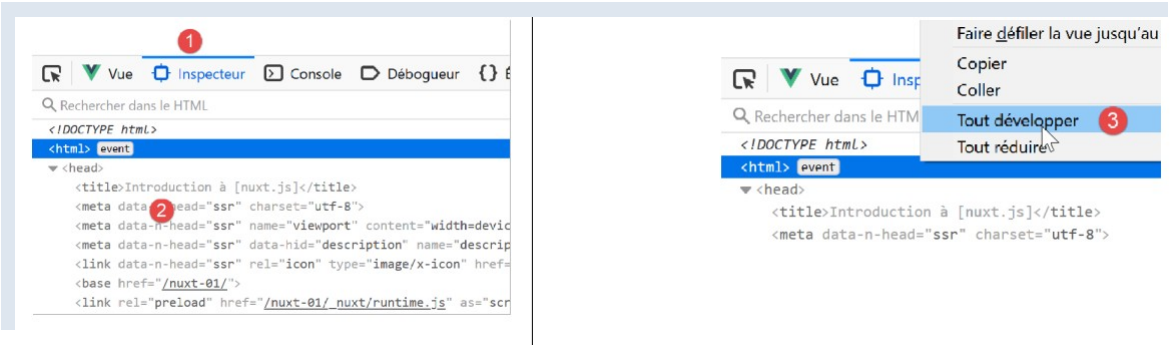
```
1.  <!doctype html>
2.  <html data-n-head-ssr>
3.  <head>
4.    <title>Introduction à [nuxt.js]</title>
5.    <meta data-n-head="ssr" charset="utf-8">
6.    <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
7.    <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing
    loading asyncdata middleware plugins store">
8.    <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
9.    <base href="/nuxt-01/">
10.   ....
11.   <link rel="preload" href="/nuxt-01/_nuxt/runtime.js" as="script">
12.   <link rel="preload" href="/nuxt-01/_nuxt/commons.app.js" as="script">
13.   <link rel="preload" href="/nuxt-01/_nuxt/vendors.app.js" as="script">
14.   <link rel="preload" href="/nuxt-01/_nuxt/app.js" as="script">
15. </head>
16. <body>
17.   <div data-server-rendered="true" id="__nuxt">
18.     <div id="__layout">
19.       <div class="container">
20.         <div class="card">
21.           <div class="card-body">
22.             <div role="alert" aria-live="polite" aria-atomic="true" align="center" class="alert
    alert-success">
23.               <h4>[nuxt-01] : routage et navigation</h4>
24.             </div>
25.             <div>
26.               <div class="row">
27.                 <div class="col-2">
28.                   <ul class="nav flex-column">
29.                     <li class="nav-item">
30.                       <a href="/nuxt-01/" target="_self" class="nav-link active nuxt-link-
    active">
31.                         Home
32.                       </a>
33.                     </li>
34.                     <li class="nav-item">
35.                       <a href="/nuxt-01/page1" target="_self" class="nav-link">
36.                         Page 1
37.                       </a>
38.                     </li>
39.                     <li class="nav-item">
40.                       <a href="/nuxt-01/page2" target="_self" class="nav-link">
41.                         Page 2
42.                       </a>
43.                     </li>
44.                   </ul>
45.                 </div> <div class="col-10">
46.                   <div role="alert" aria-live="polite" aria-atomic="true" class="alert alert-
    warning">
47.                     Home
48.                   </div>
49.                 </div>
50.               </div>
51.             </div>
52.           </div>
53.         </div>
54.       </div>
55.     </div>
56.   </div>
57.   <script>
58.     window.__NUXT__ = (function (a, b, c, d, e, f, g, h, i, j) {
59.       return {
60.         layout: "default", data: [{}], error: null, serverRendered: true,
61.         logs: [
```

```
62.           { date: new Date(1574069600078), args: [a, b, c, d, e, f, g, "(repeated 1 times)"],
    type: h, level: i, tag: j },
63.           { date: new Date(1574070938091), args: [a, b, c, d, e, f, g], type: h, level: i,
    tag: j }
64.         ]
65.       }
66.     }("[home beforeCreate]", "process.server=", "true", "process.client=", "false", "nombre
    d'arguments=", "0", "log", 2, ""));
67.   </script>
68.   <script src="/nuxt-01/_nuxt/runtime.js" defer></script>
69.   <script src="/nuxt-01/_nuxt/commons.app.js" defer></script>
70.   <script src="/nuxt-01/_nuxt/vendors.app.js" defer></script>
71.   <script src="/nuxt-01/_nuxt/app.js" defer></script>
72. </body>
73. </html>
```

Commentaires

- la première chose qui peut être remarquée est que le code HTML **reçu** reflète correctement ce que voit l'utilisateur. Ce n'était pas le cas des applications [vue] pour lesquelles le code source affiché était le code source d'un fichier HTML quasi vide. C'était ce qu'avait reçu le navigateur. Ensuite le client [vue] prenait la main et construisait la page attendue par l'utilisateur. Il fallait alors aller dans l'onglet [inspecteur] des outils de développement du navigateur (F12) pour découvrir le code HTML de la page affichée ;
- lignes 57-67 : c'est le script qui a affiché les logs tagués [Nuxt SSR]. Ces logs ont été produits côté serveur et les résultats ont été embarqués dans un script inclus dans la page envoyée ;
- lignes 68-71 : les scripts qui forment le client exécuté côté navigateur ;

Les scripts des lignes 68-71 sont exécutés et transforment la page reçue. Pour connaître la page finalement affichée pour l'utilisateur, il faut aller dans l'onglet [inspecteur] des outils de développement du navigateur (F12) :



Lorsqu'on développe la balise <html> [3], on a le contenu suivant :

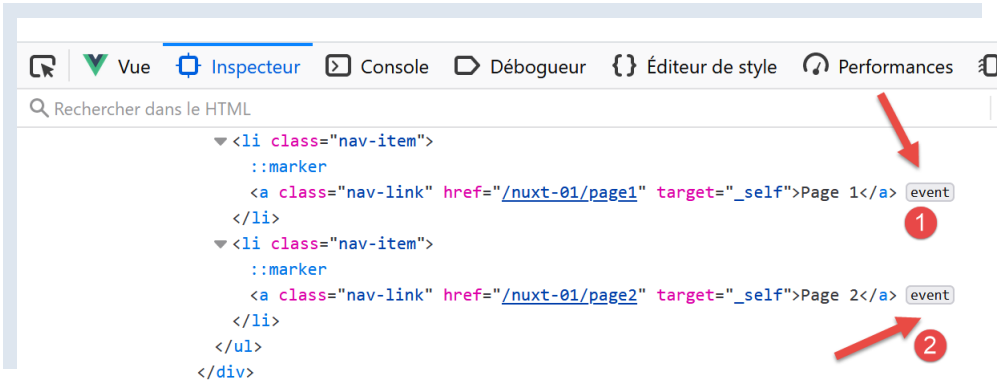
```
1.  <head>
2.    <title>Introduction à [nuxt.js]</title>
3.    <meta data-n-head="ssr" charset="utf-8">
4.    <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
5.    <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing
    loading asyncdata middleware plugins store">
6.    <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
7.    <base href="/nuxt-01/">
8.    ...
9.    <link rel="preload" href="/nuxt-01/_nuxt/runtime.js" as="script">
10.   <link rel="preload" href="/nuxt-01/_nuxt/commons.app.js" as="script">
11.   <link rel="preload" href="/nuxt-01/_nuxt/vendors.app.js" as="script">
12.   <link rel="preload" href="/nuxt-01/_nuxt/app.js" as="script">
13.
14.   <script charset="utf-8" src="/nuxt-01/_nuxt/pages_index.js"></script>
15.   <script charset="utf-8" src="/nuxt-01/_nuxt/pages_page1.js"></script>
16.   <script charset="utf-8" src="/nuxt-01/_nuxt/pages_page2.js"></script>
17. </head>
18. <body>
19.   <div id="__nuxt">
20.     <div id="__layout">
21.       <div class="container">
22.         <div class="card">
```

```
23.           <div class="card-body">
24.             <div role="alert" aria-live="polite" aria-atomic="true" class="alert alert-success"
    align="center">
25.               <h4>[nuxt-01] : routage et navigation</h4>
26.             </div>
27.             <div>
28.               <div class="row">
29.                 <div class="col-2">
30.                   <ul class="nav flex-column">
31.                     <li class="nav-item">
32.                       <a href="/nuxt-01/" target="_self" class="nav-link active nuxt-link-
    active">
33.                         Home
34.                       </a>
35.                     </li>
36.                     <li class="nav-item">
37.                       <a href="/nuxt-01/page1" target="_self" class="nav-link">
38.                         Page 1
39.                       </a>
40.                     </li>
41.                     <li class="nav-item">
42.                       <a href="/nuxt-01/page2" target="_self" class="nav-link">
43.                         Page 2
44.                       </a>
45.                     </li>
46.                   </ul>
47.                 </div>
48.                 <div class="col-10">
49.                   <div role="alert" aria-live="polite" aria-atomic="true" class="alert alert-
    warning">
50.                     Home
51.                   </div>
52.                 </div>
53.               </div>
54.             </div>
55.           </div>
56.         </div>
57.       </div>
58.     </div>
59.   </div>
60.   <script>
61.     window.__NUXT__ = (function (a, b, c, d, e, f, g, h, i) {
62.       return {
63.         layout: "default", data: [{}], error: null, serverRendered: true,
64.         logs: [
65.           { date: new Date(1574068674481), args: ["[home beforeCreate]", a, b, c, d, e, f],
    type: g, level: h, tag: i },
66.           { date: new Date(1574068674482), args: ["[home created]", a, b, c, d, e, f], type:
    g, level: h, tag: i }
67.         ]
68.       }
69.     }("process.server=", "true", "process.client=", "false", "nombre d'arguments=", "0", "log",
    2, ""));
70.   </script>
71.   <script src="/nuxt-01/_nuxt/runtime.js" defer=""></script>
72.   <script src="/nuxt-01/_nuxt/commons.app.js" defer=""></script>
73.   <script src="/nuxt-01/_nuxt/vendors.app.js" defer=""></script>
74.   <script src="/nuxt-01/_nuxt/app.js" defer=""></script>
75.
76.   <iframe id="mc-sidebar-container" ...></iframe>
77.   <iframe id="mc-topbar-container"...>  </iframe>
78.   <iframe id="mc-toast-container" ...></iframe>
79.   <iframe id="mc-download-overlay-container"...></iframe>
80. </body>
```

Commentaires

- à première vue, la page affichée lignes 19-59, semble être la même que la page reçue ;
- lignes 14-16 : trois nouveaux scripts apparaissent, un pour chacune des pages de l'application ;
- lignes 76-79 : quatre [iframe] apparaissent ;

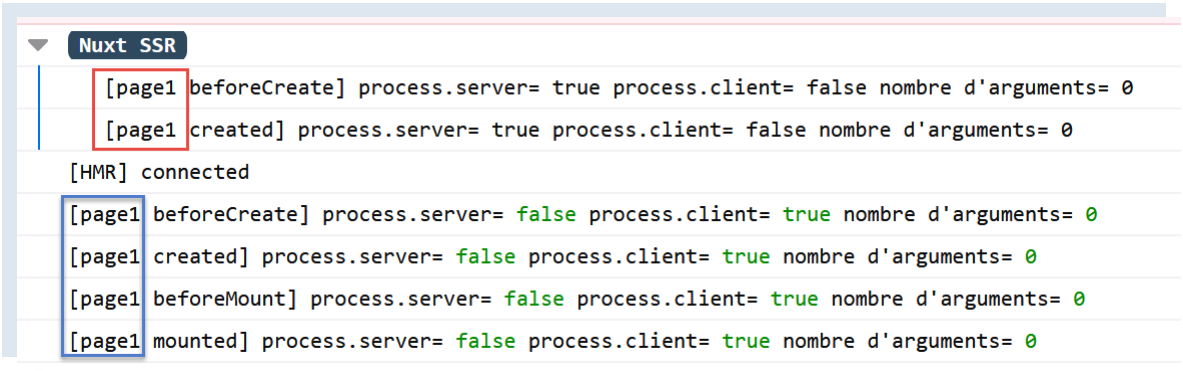
Lignes 33, 37 et 42, les liens posent problème. Ils semblent être des liens normaux qui lorsqu'on les clique vont faire une requête vers le serveur. Or à l'exécution, on voit que ce n'est pas vrai : il n'y a pas de requête vers le serveur. Pour comprendre pourquoi, il faut retourner dans l'onglet [inspecteur] du navigateur :



On voit qu'en [1, 2] des événements ont été attachés aux liens. Ce sont les scripts des lignes 71-74 qui ont attaché des gestionnaires d'événements aux liens. Donc :

- la page affichée par le client est visuellement identique à celle envoyée par le serveur ;
- un comportement dynamique a été ajouté à la page par le client ;

Maintenant demandons la page [page1] en tapant l'URL à la main [http://192.168.1.128:81/nuxt-01/page1]. Les logs deviennent les suivants :



On obtient les mêmes résultats que pour la page [index] mais pour [page1]. Le code source de la page reçue est lui le suivant :

```
1.  <body>
2.    <div data-server-rendered="true" id="__nuxt">
3.      <div id="__layout">
4.        <div class="container">
5.          <div class="card">
6.            <div class="card-body">
7.              <div role="alert" aria-live="polite" aria-atomic="true" align="center" class="alert
    alert-success"><h4>[nuxt-01] : routage et navigation</h4></div> <div>
8.                <div class="row">
9.                  <div class="col-2">
10.                   <ul class="nav flex-column">
11.                     <li class="nav-item">
12.                       <a href="/nuxt-01/" target="_self" class="nav-link">
13.                         Home
14.                       </a>
15.                     </li>
16.                     <li class="nav-item">
17.                       <a href="/nuxt-01/page1" target="_self" class="nav-link active nuxt-link-
    active">
18.                         Page 1
```

```
19.                       </a>
20.                     </li>
21.                     <li class="nav-item">
22.                       <a href="/nuxt-01/page2" target="_self" class="nav-link">
23.                         Page 2
24.                       </a>
25.                     </li>
26.                   </ul>
27.                 </div> <div class="col-10">
28.                   <div role="alert" aria-live="polite" aria-atomic="true" class="alert alert-
    primary">
29.
30.                     Page 1
31.                   </div>
32.                 </div>
33.               </div>
34.             </div>
35.           </div>
36.         </div>
37.       </div>
38.     </div>
39.   </div>
40.   <script>window.__NUXT__ = { layout: "default", data: [{}], error: null, serverRendered: true,
    logs: [{ date: new Date(1573917721122), args: ["[page1 beforeCreate]", "process.server=",
    "true", "process.client=", "false", "nombre d'arguments=", "0"], type: "log", level: 2, tag: ""
    }] };</script>
41.   <script src="/nuxt-01/_nuxt/runtime.js" defer></script>
42.   <script src="/nuxt-01/_nuxt/commons.app.js" defer></script>
43.   <script src="/nuxt-01/_nuxt/vendors.app.js" defer></script>
44.   <script src="/nuxt-01/_nuxt/app.js" defer></script>
45. </body>
```

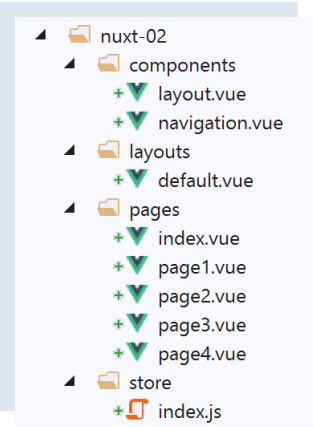
On obtient le même type de page que la page [index] mais avec l'alerte de la vue [Page 1] (ligne 30). Lignes 41-44, le code du client a été renvoyé avec la page. Au final, demander une URL à la main est **identique** à redémarrer l'application. Simplement la page affichée n'est pas forcément la page d'accueil, c'est celle qui a été demandée. Une fois la page reçue, c'est le client qui prend la main. Le serveur ne sera plus sollicité à moins que l'utilisateur n'en décide autrement.

5 Exemple [nuxt-02] : pages serveur et client

Dans ce projet, nous montrons :

- que la page construite par le client peut être visuellement différente de celle reçue du serveur. On a alors un changement rapide de page, perceptible par l'utilisateur, et qui est donc nuisible à l'ergonomie de l'application. C'est donc une option à éviter ;
- une solution pour que la page client recrée la même page que celle envoyée par le serveur ;

Le projet [nuxt-02] est obtenue initialement par recopie du projet [nuxt-01].



Un dossier [store] est ajouté au projet ainsi que deux nouvelles pages. Nous y reviendrons.

5.1 La page [index]

5.1.1 Le code de la page

Le code de la page [index] devient le suivant :

```
1.  <!-- page principale -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="warning"> Home - value= {{ value }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-undef */
13. /* eslint-disable no-console */
14. /* eslint-disable nuxt/no-env-in-hooks */
15.
16. import Navigation from '@/components/navigation'
17. import Layout from '@/components/layout'
18.
19. export default {
20.   name: 'Home',
21.   // composants utilisés
22.   components: {
23.     Layout,
24.     Navigation
25.   },
26.   data() {
27.     return {
28.       value: 0
29.     }
30.   },
```

```
31.   // cycle de vie
32.   beforeCreate() {
33.     // client et serveur
34.     console.log('[home beforeCreate]')
35.   },
36.   created() {
37.     // client et serveur
38.     console.log('[home created]')
39.     // serveur seulement
40.     if (process.server) {
41.       this.value = 10
42.     }
43.     // client et serveur
44.     console.log('value=', this.value)
45.   },
46.   beforeMount() {
47.     // client seulement
48.     console.log('[home beforeMount]')
49.   },
50.   mounted() {
51.     // client seulement
52.     console.log('[home mounted]')
53.   }
54. }
55. </script>
```

Commentaires

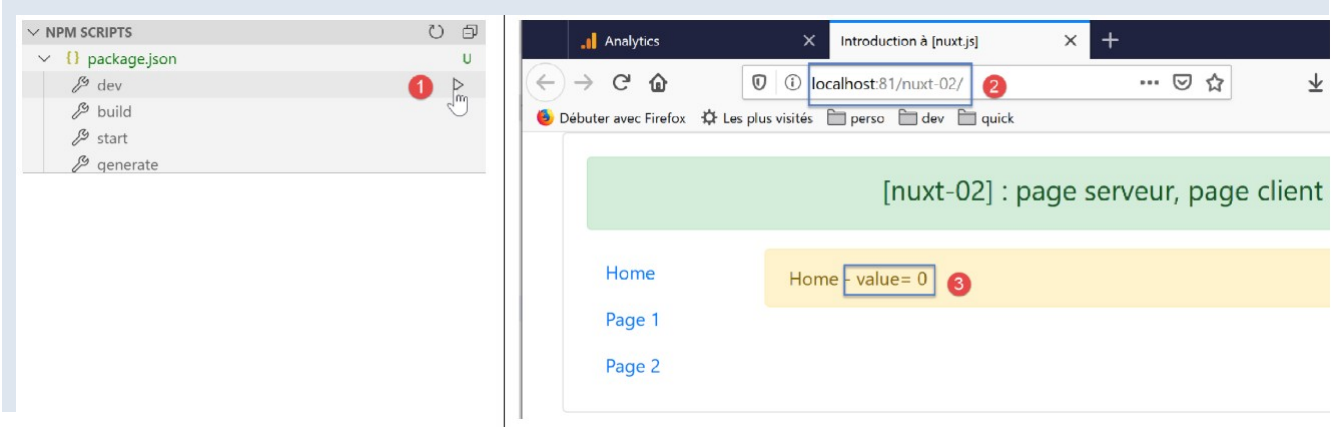
- ligne 7 : la page [index] va afficher la valeur de sa propriété [value] (ligne 28) ;
- lignes 36-45 : il faut se rappeler ici que la fonction [created] est exécutée à la fois côté serveur et côté client. Lignes 40-42, le serveur fera passer à 10 la valeur de la propriété [value]. Le client, lui, ne touche pas à cette valeur. On veut simplement savoir si cette valeur est conservée par le client. On va découvrir que non ;

5.1.2 Exécution

Nous modifions le fichier [/nuxt.config.js] pour exécuter le projet [nuxt-02] :

```
1.  ...
2.    // répertoire du code source
3.    srcDir: 'nuxt-02',
4.    // routeur
5.    router: {
6.      // racine des URL de l'application
7.      base: '/nuxt-02/'
8.    },
9.    // serveur
10.   server: {
11.     // port de service, 3000 par défaut
12.     port: 81,
13.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
14.     // 0.0.0.0 = toutes les adresses réseau de la machine
15.     host: 'localhost'
16.   }
17. ...
```

Nous exécutons le projet [1] :



La page [index] est alors affichée [2-3]. Elle affiche la valeur [10] pendant quelques instants puis affiche la valeur [0]. Que s'est-il passé ?

étape 1

C'est le serveur qui s'exécute le premier. Il exécute le code de la page [index] :

```
1.  export default {
2.    name: 'Home',
3.    // composants utilisés
4.    components: {
5.      Layout,
6.      Navigation
7.    },
8.    data() {
9.      return {
10.       value: 0
11.     }
12.   },
13.   // cycle de vie
14.   beforeCreate() {
15.     // client et serveur
16.     console.log('[home beforeCreate]')
17.   },
18.   created() {
19.     // client et serveur
20.     console.log('[home created]')
21.     // serveur seulement
22.     if (process.server) {
23.       this.value = 10
24.     }
25.     // client et serveur
26.     console.log('value=', this.value)
27.   },
28.   beforeMount() {
29.     // client seulement
30.     console.log('[home beforeMount]')
31.   },
32.   mounted() {
33.     // client seulement
34.     console.log('[home mounted]')
35.   }
36. }
```

- à cause de la ligne 23, la propriété [value] de la ligne 10 prend la valeur 10 ;

On peut le vérifier en regardant le code source de la page reçue par le navigateur (option [code source] dans le navigateur) :

```
1.  <!doctype html>
2.  <html data-n-head-ssr>
3.  <head>
```

```
4.    <title>Introduction à [nuxt.js]</title>
5.    <meta data-n-head="ssr" charset="utf-8">
6.    <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
7.    <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing
    loading asyncdata middleware plugins store">
8.    <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
9.    <base href="/nuxt-02/">
10.   <link rel="preload" href="/nuxt-02/_nuxt/runtime.js" as="script">
11.   <link rel="preload" href="/nuxt-02/_nuxt/commons.app.js" as="script">
12.   <link rel="preload" href="/nuxt-02/_nuxt/vendors.app.js" as="script">
13.   <link rel="preload" href="/nuxt-02/_nuxt/app.js" as="script">
14.   ....
15. </head>
16. <body>
17.   <div data-server-rendered="true" id="__nuxt">
18.     <div id="__layout">
19.       <div class="container">
20.         <div class="card">
21.           <div class="card-body">
22.             <div role="alert" aria-live="polite" aria-atomic="true" align="center" class="alert
    alert-success">
23.               <h4>[nuxt-02] : page serveur, page client</h4>
24.             </div> <div>
25.               <div class="row">
26.                 <div class="col-2">
27.                   <ul class="nav flex-column">
28.                     <li class="nav-item">
29.                       <a href="/nuxt-02/" target="_self" class="nav-link active nuxt-link-
    active">
30.                         Home
31.                       </a>
32.                     </li>
33.                     <li class="nav-item">
34.                       <a href="/nuxt-02/page1" target="_self" class="nav-link">
35.                         Page 1
36.                       </a>
37.                     </li>
38.                     <li class="nav-item">
39.                       <a href="/nuxt-02/page2" target="_self" class="nav-link">
40.                         Page 2
41.                       </a>
42.                     </li>
43.                   </ul>
44.                 </div> <div class="col-10">
45.                   <div role="alert" aria-live="polite" aria-atomic="true" class="alert alert-
    warning">
46.                     Home - value= 10
47.                   </div>
48.                 </div>
49.               </div>
50.             </div>
51.           </div>
52.         </div>
53.       </div>
54.     </div>
55.   </div>
56.   <script>window.__NUXT__ = ....;</script>
57.   <script src="/nuxt-02/_nuxt/runtime.js" defer></script>
58.   <script src="/nuxt-02/_nuxt/commons.app.js" defer></script>
59.   <script src="/nuxt-02/_nuxt/vendors.app.js" defer></script>
60.   <script src="/nuxt-02/_nuxt/app.js" defer></script>
61. </body>
62. </html>
```

- ligne 46 : dans la page reçue, [value] avait la valeur 10 ;

étape 2

On sait qu'après réception de la page, les scripts des lignes 57-60 prennent la main et transforment le comportement de la page reçue, voire les informations affichées comme ici. Ces scripts forment le client qui lui aussi exécute le code de la page [index], le même code que le serveur :

```
1.  export default {
```

```
2.    name: 'Home',
3.    // composants utilisés
4.    components: {
5.      Layout,
6.      Navigation
7.    },
8.    data() {
9.      return {
10.       value: 0
11.     }
12.   },
13.   // cycle de vie
14.   beforeCreate() {
15.     // client et serveur
16.     console.log('[home beforeCreate]')
17.   },
18.   created() {
19.     // client et serveur
20.     console.log('[home created]')
21.     // serveur seulement
22.     if (process.server) {
23.       this.value = 10
24.     }
25.     // client et serveur
26.     console.log('value=', this.value)
27.   },
28.   beforeMount() {
29.     // client seulement
30.     console.log('[home beforeMount]')
31.   },
32.   mounted() {
33.     // client seulement
34.     console.log('[home mounted]')
35.   }
36. }
```

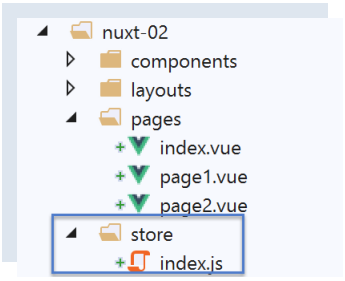
- pour comprendre ce qui se passe, il faut comprendre que le client [nuxt] n'exécutera pas les lignes 22-24 (process.server=false) ;
- dans une application [vue] classique la propriété [value] de la ligne 10 reste à 0. C'est pourquoi, une fois que le client est passé sur la page reçue, la valeur affichée devient [0] ;

La valeur générée par le serveur [nuxt] pour la propriété [value] n'a servi à rien.

5.2 La page [page1]

5.2.1 Le store [Vuex]

Nous avons ajouté un dossier [store] au projet [nuxt-02] :



La présence de ce dossier fait qu'automatiquement [nuxt] va implémenter un store [Vuex]. C'est le fichier [index.js] qui implémente ce store. Ici, le fichier [index.js] est le suivant :

```
1.  export const state = () => ({
2.    counter: 0
3.  })
4.
5.  export const mutations = {
```

```
6.    increment(state, inc) {
7.      state.counter += inc
8.    }
9.  }
```

[nuxt] implémente un store [Vuex] à partir du contenu de [index.js] :

- lignes 1-3 : définition de l'état [state] du store. Cet état est retourné par une **fonction**. Ici, l'état n'a qu'une propriété, le compteur de la ligne 2. La fonction exportée doit s'appeler [state] ;
- lignes 5-9 : les opération possibles sur l'état du store. On les appelle des [mutations]. Ici, la mutation [increment] permet d'incrémenter la propriété [counter] d'une quantité [inc]. L'objet exporté doit s'appeler [mutations] ;

Le [store] Vuex implémenté par [nuxt] est disponible à différents endroits. Dans les vues, il est disponible dans la propriété [**this.$store**].

5.2.2 Le code de la page

Comme la page [index], la page [page1] va afficher une valeur, celle du compteur du store Vuex :

```
1.  <!-- page 1 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="primary"> Page 1 - value = {{ value }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13.
14. import Navigation from '@/components/navigation'
15. import Layout from '@/components/layout'
16.
17. export default {
18.   name: 'Page1',
19.   // composants utilisés
20.   components: {
21.     Layout,
22.     Navigation
23.   },
24.   data() {
25.     return {
26.       value: 0
27.     }
28.   },
29.   // cycle de vie
30.   beforeCreate() {
31.     // client et serveur
32.     console.log('[home beforeCreate]')
33.   },
34.   created() {
35.     // client et serveur
36.     console.log('[home created]')
37.     // serveur seulement
38.     if (process.server) {
39.       this.$store.commit('increment', 25)
40.     }
41.     // client et serveur
42.     this.value = this.$store.state.counter
43.     console.log('value=', this.value)
44.   },
45.   beforeMount() {
46.     // client seulement
47.     console.log('[home beforeMount]')
48.   },
49.   mounted() {
50.     // client seulement
51.     console.log('[home mounted]')
```

```
52.   }
53. }
54. </script>
```

Commentaires

- lignes 38-40 : le serveur va incrémenter le compteur de 25 ;
- ligne 42 : aussi bien le serveur que le client vont afficher la valeur du compteur ;
- ligne 7 : la valeur du compteur est affichée ;

En lisant ce code, il faut comprendre deux choses :

- le code exécuté est le même pour le serveur que le client ;
- l'objet [this] n'est lui pas le même : il y a une version [this] côté serveur et une autre côté [client] ;

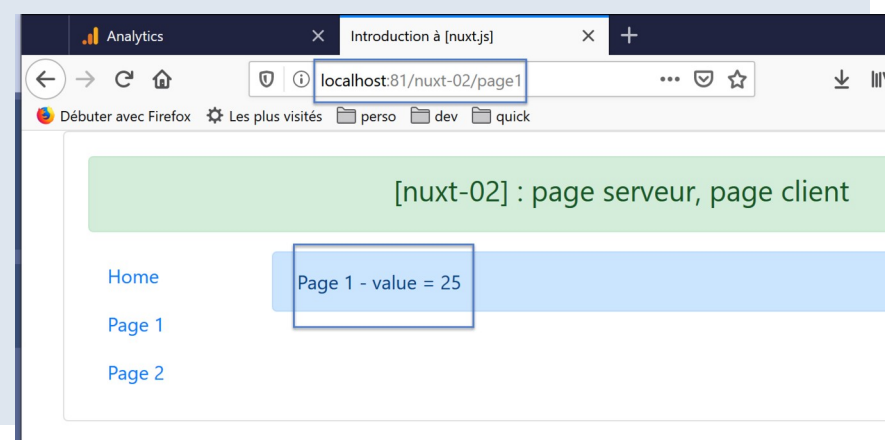
Nous cherchons à savoir si le [this.$store] du serveur est le même que le [this.$store] du client. Comme c'est le serveur qui s'exécute en premier (au démarrage de l'application), cela revient à se poser la question : est-ce que le [store] initialisé par le serveur est transmis au client ?

5.2.3 Exécution

On exécute le projet [nuxt-02] et on tape [localhost:81/nuxt-02/**page1**] à la main pour que le serveur soit sollicité. Comme au démarrage pour la page [index] :

- le serveur exécute la page [page1.vue] ;
- envoie la page générée au navigateur. Celle-ci est affichée ;
- les scripts client embarqués dans la page envoyée prennent la main et exécutent à nouveau la page [page1.vue] ;
- la page affichée est alors modifiée ;

Le résultat final est le suivant :



Cette fois-ci, la valeur affichée est bien celle fixée par le serveur et visuellement, on ne voit pas la page 'tressauter' à cause d'un changement par le client de la valeur affichée par le serveur. Cette fois-ci que s'est-il passé ?

Le serveur a exécuté la page [page1] suivante :

```
1.  ...
2.
3.  <script>
4.  /* eslint-disable no-console */
5.
6.  import Navigation from '@/components/navigation'
7.  import Layout from '@/components/layout'
8.
9.  export default {
10.   name: 'Page1',
11.   // composants utilisés
```

```
12.   components: {
13.     Layout,
14.     Navigation
15.   },
16.   data() {
17.     return {
18.       value: 0
19.     }
20.   },
21.   // cycle de vie
22.   beforeCreate() {
23.     // client et serveur
24.     console.log('[page1 beforeCreate]')
25.   },
26.   created() {
27.     // client et serveur
28.     console.log('[page1 created]')
29.     // serveur seulement
30.     if (process.server) {
31.       this.$store.commit('increment', 25)
32.     }
33.     // client et serveur
34.     this.value = this.$store.state.counter
35.     console.log('value=', this.value)
36.   },
37.   beforeMount() {
38.     // client seulement
39.     console.log('[page1 beforeMount]')
40.   },
41.   mounted() {
42.     // client seulement
43.     console.log('[page1 mounted]')
44.   }
45. }
46. </script>
```

- les lignes 30-32 ont été exécutées sans erreur. Ce qui signifie que côté serveur également, [this.$store] désigne le store [Vuex]. La ligne 31 a fait passer le compteur du store à 25 ;
- ceci fait, la page a été envoyée au client ;

Si on regarde la page reçue par le client, on trouve les éléments suivants :

```
1.  <!doctype html>
2.  <html data-n-head-ssr>
3.  <head>
4.    <title>Introduction à [nuxt.js]</title>
5.    <meta data-n-head="ssr" charset="utf-8">
6.    <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
7.    <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing
    loading asyncdata middleware plugins store">
8.    <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
9.    <base href="/nuxt-02/">
10.   <link rel="preload" href="/nuxt-02/_nuxt/runtime.js" as="script">
11.   <link rel="preload" href="/nuxt-02/_nuxt/commons.app.js" as="script">
12.   <link rel="preload" href="/nuxt-02/_nuxt/vendors.app.js" as="script">
13.   <link rel="preload" href="/nuxt-02/_nuxt/app.js" as="script">
14.   ...
15. </head>
16. <body>
17.   <div data-server-rendered="true" id="__nuxt">
18.     <div id="__layout">
19.       <div class="container">
20.         <div class="card">
21.           <div class="card-body">
22.             <div role="alert" aria-live="polite" aria-atomic="true" align="center" class="alert
    alert-success">
23.               <h4>[nuxt-02] : page serveur, page client</h4>
24.             </div>
25.             <div>
26.               <div class="row">
27.                 <div class="col-2">
28.                   <ul class="nav flex-column">
29.                     <li class="nav-item">
```

```
30.                       <a href="/nuxt-02/" target="_self" class="nav-link">
31.                         Home
32.                       </a>
33.                     </li>
34.                     <li class="nav-item">
35.                       <a href="/nuxt-02/page1" target="_self" class="nav-link active nuxt-link-
    active">
36.                         Page 1
37.                       </a>
38.                     </li>
39.                     <li class="nav-item">
40.                       <a href="/nuxt-02/page2" target="_self" class="nav-link">
41.                         Page 2
42.                       </a>
43.                     </li>
44.                   </ul>
45.                 </div> <div class="col-10">
46.                   <div role="alert" aria-live="polite" aria-atomic="true" class="alert alert-
    primary">
47.                     Page 1 - value = 25
48.                   </div>
49.                 </div>
50.               </div>
51.             </div>
52.           </div>
53.         </div>
54.       </div>
55.     </div>
56.   </div>
57.   <script>
58.   window.__NUXT__ = (function (a, b, c) {
59.       return {
60.         layout: "default", data: [{}], error: null, state: { counter: 25 }, serverRendered:
    true,
61.         logs: [
62.           { date: new Date(1574085336802), args: ["[home beforeCreate]"], type: a, level: b,
    tag: c },
63.           { date: new Date(1574085336839), args: ["[home created]"], type: a, level: b, tag: c
    },
64.           { date: new Date(1574085336869), args: ["value=", "25"], type: a, level: b, tag: c }
65.         ]
66.       }
67.     }("log", 2, ""));</script>
68.   <script src="/nuxt-02/_nuxt/runtime.js" defer></script>
69.   <script src="/nuxt-02/_nuxt/commons.app.js" defer></script>
70.   <script src="/nuxt-02/_nuxt/vendors.app.js" defer></script>
71.   <script src="/nuxt-02/_nuxt/app.js" defer></script>
72. </body>
73. </html>
```

- ligne 47 : la valeur envoyée par le serveur ;
- ligne 60 : on constate que l'état du store [Vuex] a été embarqué dans la page. Cela va permettre au client qui va s'exécuter après réception de la page, de reconstituer un nouveau store [Vuex] avec 25 comme valeur initiale du compteur ;

Après réception et affichage de la page reçue du serveur, le client prend la main et exécute à son tour la page [page1] :

```
1.  ...
2.
3.  <script>
4.  /* eslint-disable no-console */
5.
6.  import Navigation from '@/components/navigation'
7.  import Layout from '@/components/layout'
8.
9.  export default {
10.   name: 'Page1',
11.   // composants utilisés
12.   components: {
13.     Layout,
14.     Navigation
15.   },
```

```
16.   data() {
17.     return {
18.       value: 0
19.     }
20.   },
21.   // cycle de vie
22.   beforeCreate() {
23.     // client et serveur
24.     console.log('[page1 beforeCreate]')
25.   },
26.   created() {
27.     // client et serveur
28.     console.log('[page1 created]')
29.     // serveur seulement
30.     if (process.server) {
31.       this.$store.commit('increment', 25)
32.     }
33.     // client et serveur
34.     this.value = this.$store.state.counter
35.     console.log('value=', this.value)
36.   },
37.   beforeMount() {
38.     // client seulement
39.     console.log('[page1 beforeMount]')
40.   },
41.   mounted() {
42.     // client seulement
43.     console.log('[page1 mounted]')
44.   }
45. }
46. </script>
```

- ligne 34 : la propriété [value] de la ligne 18 reçoit à son tour la valeur 25 du compteur ;

Le store de [nuxt] permet donc au serveur de transmettre des informations au client lors du chargement initial de la page, lorsque celle-ci est cherchée sur le serveur. On rappelle qu'une fois cette page obtenue, le serveur n'est plus sollicité et l'application fonctionne comme une application [vue] classique, en mode SAP.

5.3 La page [page2]

Dans la page [page2], nous montrons une autre façon pour que :

- le serveur inclut des informations calculées dans la page ;
- le client ne modifie pas celles-ci ;

5.3.1 Le code de la page

Le code de la page [page2] évolue de la façon suivante :

```
1.  <!-- page2 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-alert slot="right" show variant="secondary"> Page 2 - value = {{ value }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13. /* eslint-disable nuxt/no-timing-in-fetch-data */
14.
15. import Navigation from '@/components/navigation'
16. import Layout from '@/components/layout'
17.
18. export default {
19.   name: 'Page2',
20.   // composants utilisés
21.   components: {
22.     Layout,
```

```
23.     Navigation
24.   },
25.   asyncData(context) {
26.     // qui exécute ce code ?
27.     console.log('asyncData, client=', process.client, 'serveur=', process.server)
28.     // seulement pour le serveur
29.     if (process.server) {
30.       // on retourne une promesse
31.       return new Promise(function(resolve, reject) {
32.         // on a normalement ici une fonction asynchrone
33.         // on la simule avec une attente d'une seconde
34.         setTimeout(() => {
35.           // ce résultat sera inclus dans les propriétés de [data]
36.           resolve({ value: 87 })
37.           // log
38.           console.log('asynData terminée')
39.         }, 1000)
40.       })
41.     }
42.   },
43.   // cycle de vie
44.   beforeCreate() {
45.     // client et serveur
46.     console.log('[page2 beforeCreate]')
47.   },
48.   created() {
49.     // client et serveur
50.     console.log('[page2 created]')
51.   },
52.   beforeMount() {
53.     // client seulement
54.     console.log('[page2 beforeMount]')
55.   },
56.   mounted() {
57.     // client seulement
58.     console.log('[page2 mounted]')
59.   }
60. }
61. </script>
```

- ligne 7 : la page affiche la valeur d'une propriété nommée [value] ;
- la propriété [value] n'existe pas comme élément d'un objet rendu par la fonction [data]. Ici cette fonction n'existe pas. La propriété [value] est créée dynamiquement par la ligne 36 ;
- ligne 25 : la fonction [asyncData] est une fonction [nuxt]. Comme son nom l'indique, c'est normalement une fonction asynchrone. Son rôle habituel est d'aller chercher des données externes. [nuxt] assure que la page n'est pas envoyée au navigateur client avant que la fonction [asyncData] n'ait rendu ses données asynchrones ;
- la fonction [asyncData] reçoit comme paramètre le contexte [nuxt]. Cet objet est très dense et donne accès à beaucoup d'informations sur l'application [nuxt. Nous le découvrirons dans les sections à venir ;
- ligne 31 : on implémente la fonction [asyncData] avec une [Promise] (cf document |Introduction au langage **ECMASCRIPT 6** par l'exemple|). Le constructeur de cette classe accepte comme paramètre une fonction asynchrone qui :
  - signale un succès en rendant des données avec la fonction [resolve]. L'objet rendu par cette fonction est automatiquement inclus dans les propriétés [data] de la page ;
  - signale un échec en rendant une erreur avec la fonction [reject] ;
- ligne 34 : on simule une fonction asynchrone avec la fonction [setTimeout]. Cette fonction rend l'objet [`{ value: 87 }`] (ligne 36) au bout d'une seconde (ligne 31) grâce à la fonction [resolve] qui signale un succès de la [Promise]. L'objet rendu par la fonction asynchrone est inclus automatiquement dans les propriétés [data] de la page. Et c'est donc cette propriété que la ligne 7 affiche ;
- ligne 27 : nous allons découvrir que la fonction [asyncData] est exécutée par le serveur mais pas par le client ;
- ligne 29 : l'initialisation de la propriété [value] est faite par le serveur ;

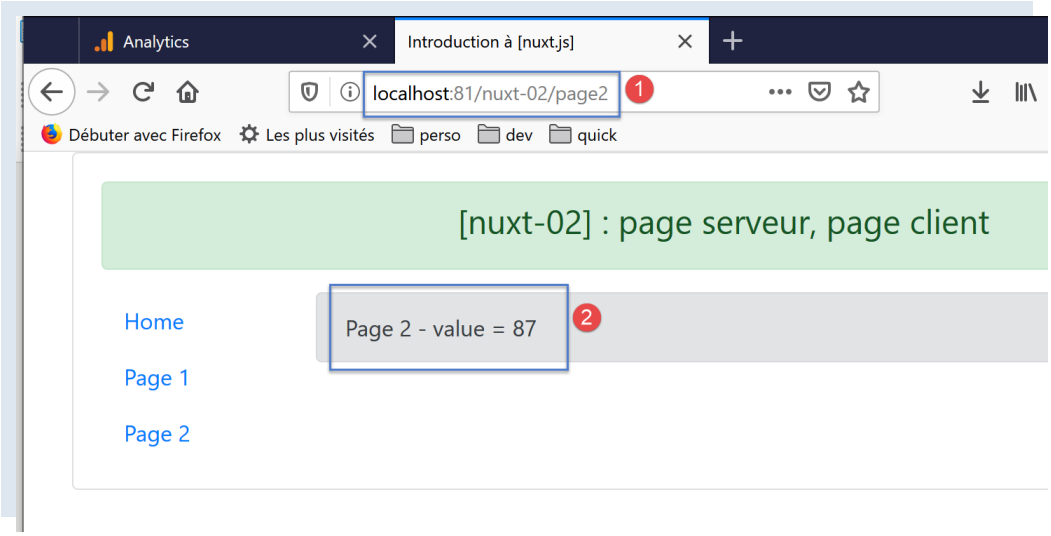
Note : l'objet [this] n'est pas connu dans la fonction [asyncData] car l'objet encapsulant le composant [vue] n'a pas encore été créé ;

5.3.2 Exécution

On exécute le projet [nuxt-02] et on tape [localhost:81/nuxt-02/**page2**] à la main pour que le serveur soit sollicité. Comme au démarrage pour la page [index] :

- le serveur exécute la page [page2.vue] ;
- envoie la page générée au navigateur. Celle-ci est affichée ;
- les scripts client embarqués dans la page envoyée prennent la main et exécutent à nouveau la page [page2.vue] ;
- la page affichée est alors modifiée ;

Le résultat final est le suivant :



Cette fois-ci, la valeur affichée est bien celle fixée par le serveur et visuellement, on ne voit pas la page 'tressauter' à cause d'un changement par le client de la valeur affichée par le serveur. Cette fois-ci que s'est-il passé ?

Le serveur a exécuté la page [page2] suivante :

```
1.  <!-- page2 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-alert slot="right" show variant="secondary"> Page 2 - value = {{ value }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13. /* eslint-disable nuxt/no-timing-in-fetch-data */
14.
15. import Navigation from '@/components/navigation'
16. import Layout from '@/components/layout'
17.
18. export default {
19.   name: 'Page2',
20.   // composants utilisés
21.   components: {
22.     Layout,
23.     Navigation
24.   },
25.   asyncData(context) {
26.     // qui exécute ce code ?
27.     console.log('asyncData, client=', process.client, 'serveur=', process.server)
28.     // seulement pour le serveur
29.     if (process.server) {
30.       // on retourne une promesse
```

```
31.       return new Promise(function(resolve, reject) {
32.         // on a normalement ici une fonction asynchrone
33.         // on la simule avec une attente d'une seconde
34.         setTimeout(() => {
35.           // ce résultat sera inclus dans les propriétés de [data]
36.           resolve({ value: 87 })
37.           // log
38.           console.log('asynData terminée')
39.         }, 1000)
40.       })
41.     }
42.   },
43.   // cycle de vie
44.   beforeCreate() {
45.     // client et serveur
46.     console.log('[page2 beforeCreate]')
47.   },
48.   created() {
49.     // client et serveur
50.     console.log('[page2 created]')
51.   },
52.   beforeMount() {
53.     // client seulement
54.     console.log('[page2 beforeMount]')
55.   },
56.   mounted() {
57.     // client seulement
58.     console.log('[page2 mounted]')
59.   }
60. }
61. </script>
```

C'est la ligne 36 qui a fixé la valeur affichée par la ligne 7. C'est donc ce qu'a reçu le navigateur client. Très exactement il reçoit la page suivante :

```
1.  <!doctype html>
2.  <html data-n-head-ssr>
3.  <head>
4.    <title>Introduction à [nuxt.js]</title>
5.    <meta data-n-head="ssr" charset="utf-8">
6.    <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
7.    <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing
    loading asyncdata middleware plugins store">
8.    <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
9.    <base href="/nuxt-02/">
10.   <link rel="preload" href="/nuxt-02/_nuxt/runtime.js" as="script">
11.   <link rel="preload" href="/nuxt-02/_nuxt/commons.app.js" as="script">
12.   <link rel="preload" href="/nuxt-02/_nuxt/vendors.app.js" as="script">
13.   <link rel="preload" href="/nuxt-02/_nuxt/app.js" as="script">
14.   ...
15. </head>
16. <body>
17.   <div data-server-rendered="true" id="__nuxt">
18.     <div id="__layout">
19.       <div class="container">
20.         <div class="card">
21.           <div class="card-body">
22.             <div role="alert" aria-live="polite" aria-atomic="true" align="center" class="alert
    alert-success">
23.               <h4>[nuxt-02] : page serveur, page client</h4>
24.             </div>
25.             <div>
26.               <div class="row">
27.                 <div class="col-2">
28.                   <ul class="nav flex-column">
29.                     <li class="nav-item">
30.                       <a href="/nuxt-02/" target="_self" class="nav-link">
31.                         Home
32.                       </a>
33.                     </li>
34.                     <li class="nav-item">
35.                       <a href="/nuxt-02/page1" target="_self" class="nav-link">
36.                         Page 1
37.                       </a>
```

```
38.                     </li>
39.                     <li class="nav-item">
40.                       <a href="/nuxt-02/page2" target="_self" class="nav-link active nuxt-link-
    active">
41.                         Page 2
42.                       </a>
43.                     </li>
44.                   </ul>
45.                 </div>
46.                 <div class="col-10">
47.                   <div role="alert" aria-live="polite" aria-atomic="true" class="alert alert-
    secondary">
48.                     Page 2 - value = 87
49.                   </div>
50.                 </div>
51.               </div>
52.             </div>
53.           </div>
54.         </div>
55.       </div>
56.     </div>
57.   </div>
58.   <script>
59.   window.__NUXT__ = (function (a, b, c) {
60.       return {
61.         layout: "default", data: [{ value: 87 }], error: null, state: { counter: 0 },
    serverRendered: true,
62.         logs: [
63.           { date: new Date(1574096608555), args: ["asyncData, client=", "false", "serveur=",
    "true"], type: a, level: b, tag: c },
64.           { date: new Date(1574096608575), args: ["[page2 beforeCreate]"], type: a, level: b,
    tag: c },
65.           { date: new Date(1574096608599), args: ["[page2 created]"], type: a, level: b, tag:
    c }
66.         ]
67.       }
68.     }("log", 2, ""));</script>
69.   <script src="/nuxt-02/_nuxt/runtime.js" defer></script>
70.   <script src="/nuxt-02/_nuxt/commons.app.js" defer></script>
71.   <script src="/nuxt-02/_nuxt/vendors.app.js" defer></script>
72.   <script src="/nuxt-02/_nuxt/app.js" defer></script>
73. </body>
74. </html>
```

- ligne 48 : on voit que la valeur dans la page reçue est 87 ;
- ligne 61 : dans la réponse du serveur, on voit deux objets : [data] et [state] :
  - [state] est l'état du store [Vuex]. Celui-ci a été instancié à partir du contenu du dossier [store] de l'application [nuxt-02] ;
  - [data] contient les propriétés créées par le serveur grâce à la fonction [asyncData]. On retrouve la propriété [value : 87] créée par le serveur. Les scripts du client vont intégrer cette propriété dans celles de la page [page2] ;

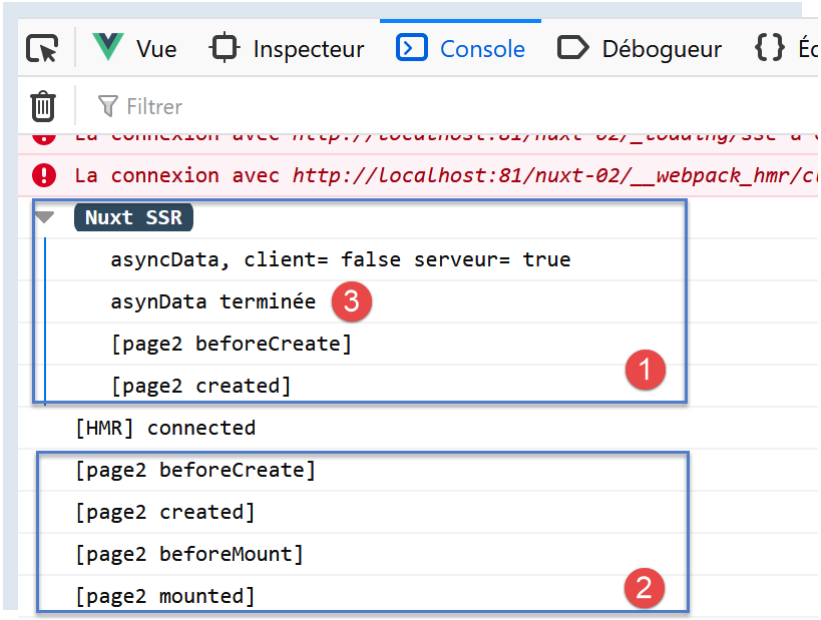
Revenons au code de la page [page2] :

```
1.  <!-- page2 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-alert slot="right" show variant="secondary"> Page 2 - value = {{ value }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13. /* eslint-disable nuxt/no-timing-in-fetch-data */
14.
15. import Navigation from '@/components/navigation'
16. import Layout from '@/components/layout'
17.
18. export default {
```

```
19.   name: 'Page2',
20.   // composants utilisés
21.   components: {
22.     Layout,
23.     Navigation
24.   },
25.   asyncData(context) {
26.     // qui exécute ce code ?
27.     console.log('asyncData, client=', process.client, 'serveur=', process.server)
28.     // seulement pour le serveur
29.     if (process.server) {
30.       // on retourne une promesse
31.       return new Promise(function(resolve, reject) {
32.         // on a normalement ici une fonction asynchrone
33.         // on la simule avec une attente d'une seconde
34.         setTimeout(() => {
35.           // ce résultat sera inclus dans les propriétés de [data]
36.           resolve({ value: 87 })
37.           // log
38.           console.log('asynData terminée')
39.         }, 1000)
40.       })
41.     }
42.   },
43.   // cycle de vie
44.   beforeCreate() {
45.     // client et serveur
46.     console.log('[page2 beforeCreate]')
47.   },
48.   created() {
49.     // client et serveur
50.     console.log('[page2 created]')
51.   },
52.   beforeMount() {
53.     // client seulement
54.     console.log('[page2 beforeMount]')
55.   },
56.   mounted() {
57.     // client seulement
58.     console.log('[page2 mounted]')
59.   }
60. }
61. </script>
```

- la ligne 7 utilise la propriété [value]. Or la page ne définit aucune propriété nommée [value]. Cependant les scripts du client ont créé automatiquement cette propriété grâce à l'objet [`data: [{ value: 87 }]`] reçue du serveur ;

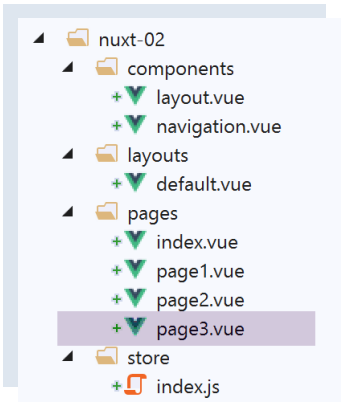
Les logs montrent par ailleurs que la fonction [asyncData] n'a pas été exécutée par le client :



La fonction [asyncData] a été exécutée par le serveur [1] mais pas par le client [2]. Par ailleurs, on peut noter que les fonctions du cycle de vie ne sont pas exécutées par le serveur avant la fin de la fonction [asyncData]. On peut augmenter la durée de l'attente au sein de la fonction [asyncData] pour le vérifier.

5.4 La page [page3]

Nous ajoutons une nouvelle page [page3] à notre application :



5.4.1 Le composant [navigation]

Le composant [vavigation] est modifié pour permettre la navigation vers la nouvelle page :

```
1.  <template>
2.    <!-- menu Bootstrap à trois options -->
3.    <b-nav vertical>
4.      <b-nav-item to="/" exact exact-active-class="active">
5.        Home
6.      </b-nav-item>
7.      <b-nav-item to="/page1" exact exact-active-class="active">
8.        Page 1
9.      </b-nav-item>
10.     <b-nav-item to="/page2" exact exact-active-class="active">
11.       Page 2
12.     </b-nav-item>
```

```
    <b-nav-item to="/page3" exact exact-active-class="active">
      Page 3
    </b-nav-item>
  </b-nav>
</template>
```

5.4.2 Le code de [page3]

Le code de la page [page3] est le suivant :

```
<!-- page3 -->
<template>
  <Layout :left="true" :right="true">
    <!-- navigation -->
    <Navigation slot="left" />
    <!-- message -->
    <b-alert slot="right" show variant="secondary"> Page 3 - value = {{ value }} </b-alert>
  </Layout>
</template>

<script>
/* eslint-disable no-console */
/* eslint-disable nuxt/no-timing-in-fetch-data */

import Navigation from '@/components/navigation'
import Layout from '@/components/layout'

export default {
  name: 'Page3',
  // composants utilisés
  components: {
    Layout,
    Navigation
  },
  data() {
    return {
      value: 0
    }
  },
  fetch(context) {
    // qui exécute ce code ?
    console.log('fetch, client=', process.client, 'serveur=', process.server)
    // seulement pour le serveur
    if (process.server) {
      // on retourne une promesse
      return new Promise(function(resolve, reject) {
        // on a normalement ici une fonction asynchrone
        // on la simule avec une attente d'une seconde
        setTimeout(() => {
          // succès
          resolve()
        }, 1000)
      }).then(() => {
        // on modifie le store
        context.store.commit('increment', 28)
      })
    }
  },
  // cycle de vie
  beforeCreate() {
    // client et serveur
    console.log('[page3 beforeCreate]')
  },
  created() {
    // client et serveur
    this.value = this.$store.state.counter
    console.log('[page3 created], value=', this.value)
  },
  beforeMount() {
    // client seulement
    console.log('[page3 beforeMount]')
  },
  mounted() {
    // client seulement
```

```
65.     console.log('[page3 mounted]')
66.   }
67. }
68. </script>
```

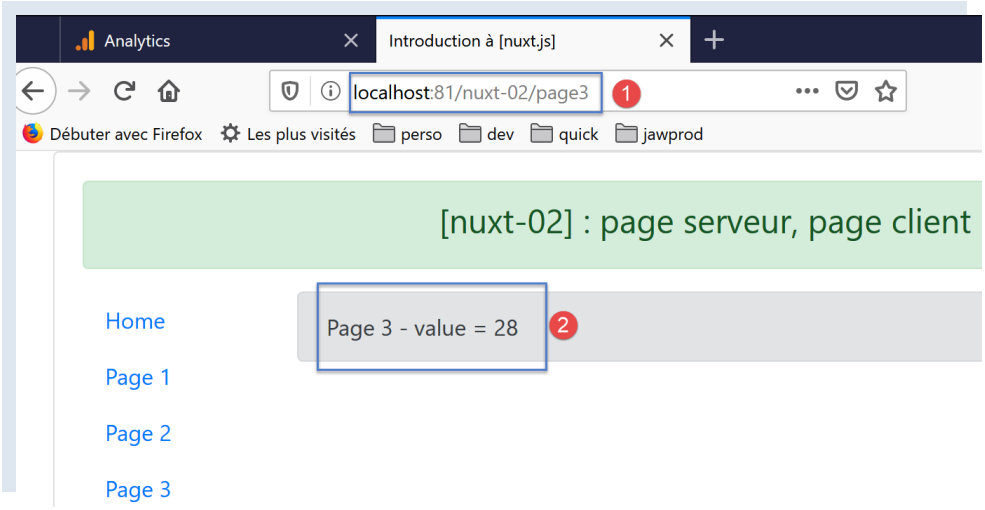
- ligne 30 : la fonction [fetch] a un comportement analogue à celui de la fonction [asyncData] :
  - elle est exécutée avant les fonctions du cycle de vie ;
  - l'objet [this] n'est pas connu dans cette fonction ;
  - son fonctionnement est asynchrone ;
  - le cycle de vie ne commence pas tant que la fonction asynchrone n'a pas rendu son résultat ;
  - le résultat est ici rendu par la méthode [then] de la [Promise], ligne 43 ;
  - fonction [fetch] reçoit le paramètre [context]. Celui-ci représente le contexte [nuxt] du moment ;
- ligne 30 : parmi ses nombreuses propriétés, l'objet [context] a une propriété [store] qui représente le store [Vuex] de l'application ;
- ligne 41 : artificiellement, on signale le succès de la [Promise] au bout d'une seconde (cf. document | Introduction au langage **ECMASCRIPT 6** par l'exemple |) ;
- ligne 45 : la méthode [then] est alors exécutée. On y incrémente le compteur du [store] ;

5.4.3 Exécution

On exécute le projet [nuxt-02] et on tape [localhost:81/nuxt-02/**page3**] à la main pour que le serveur soit sollicité. Comme au démarrage pour la page [index] :

- le serveur exécute la page [page3.vue] ;
- envoie la page générée au navigateur. Celle-ci est affichée ;
- les scripts client embarqués dans la page envoyée prennent la main et exécutent à nouveau la page [page3.vue] ;
- la page affichée est alors modifiée ;

Le résultat final est le suivant :



La valeur affichée est bien celle fixée par le serveur et visuellement, on ne voit pas la page 'tressauter' à cause d'un changement par le client de la valeur affichée par le serveur. Cette fois-ci que s'est-il passé ?

Le serveur a exécuté la page [page3] suivante :

```
1.  <!-- page3 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-alert slot="right" show variant="secondary"> Page 3 - value = {{ value }} </b-alert>
8.    </Layout>
9.  </template>
10.
```

```
11. <script>
12. /* eslint-disable no-console */
13. /* eslint-disable nuxt/no-timing-in-fetch-data */
14.
15. import Navigation from '@/components/navigation'
16. import Layout from '@/components/layout'
17.
18. export default {
19.   name: 'Page3',
20.   // composants utilisés
21.   components: {
22.     Layout,
23.     Navigation
24.   },
25.   data() {
26.     return {
27.       value: 0
28.     }
29.   },
30.   fetch(context) {
31.     // qui exécute ce code ?
32.     console.log('fetch, client=', process.client, 'serveur=', process.server)
33.     // seulement pour le serveur
34.     if (process.server) {
35.       // on retourne une promesse
36.       return new Promise(function(resolve, reject) {
37.         // on a normalement ici une fonction asynchrone
38.         // on la simule avec une attente d'une seconde
39.         setTimeout(() => {
40.           // succès
41.           resolve()
42.         }, 1000)
43.       }).then(() => {
44.         // on modifie le store
45.         context.store.commit('increment', 28)
46.         // log
47.         console.log('fetch commit terminé')
48.       })
49.     }
50.   },
51.   // cycle de vie
52.   beforeCreate() {
53.     // client et serveur
54.     console.log('[page3 beforeCreate]')
55.   },
56.   created() {
57.     // client et serveur
58.     this.value = this.$store.state.counter
59.     console.log('[page3 created], value=', this.value)
60.   },
61.   beforeMount() {
62.     // client seulement
63.     console.log('[page3 beforeMount]')
64.   },
65.   mounted() {
66.     // client seulement
67.     console.log('[page3 mounted]')
68.   }
69. }
70. </script>
```

- ligne 45 : la fonction asynchrone [fetch] est la première des fonctions ci-dessus à s'exécuter. Elle reçoit en paramètre, un objet appelé [context] qui est le contexte [nuxt] du moment. Parmi les très nombreuses propriétés de cet objet, la propriété [context.store] représente le store [Vuex] ;
- ligne 45 : dans la fonction asynchrone [fetch], le serveur fixe le compteur du store à 28 ;
- ligne 56 : lorsque la fonction [created] s'exécute, [nuxt] garantit que la fonction asynchrone [fetch] a terminé son travail ;
- ligne 58 : la valeur du compteur du store est affectée à la propriété [value] de la ligne 27 ;
- ligne 7 : affichage de la valeur de [value], donc du compteur du store ;

Le navigateur client reçoit la page suivante :

```
<!doctype html>
<html data-n-head-ssr>
<head>
  <title>Introduction à [nuxt.js]</title>
  <meta data-n-head="ssr" charset="utf-8">
  <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
  <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing loading asyncdata middleware plugins store">
  <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
  <base href="/nuxt-02/">
  <link rel="preload" href="/nuxt-02/_nuxt/runtime.js" as="script">
  <link rel="preload" href="/nuxt-02/_nuxt/commons.app.js" as="script">
  <link rel="preload" href="/nuxt-02/_nuxt/vendors.app.js" as="script">
  <link rel="preload" href="/nuxt-02/_nuxt/app.js" as="script">
  ...
</head>
<body>
  <div data-server-rendered="true" id="__nuxt">
    <div id="__layout">
      <div class="container">
        <div class="card">
          <div class="card-body">
            <div role="alert" aria-live="polite" aria-atomic="true" align="center" class="alert alert-success">
              <h4>[nuxt-02] : page serveur, page client</h4>
            </div>
            <div>
              <div class="row">
                <div class="col-2">
                  <ul class="nav flex-column">
                    <li class="nav-item">
                      <a href="/nuxt-02/" target="_self" class="nav-link">
                        Home
                      </a>
                    </li>
                    <li class="nav-item">
                      <a href="/nuxt-02/page1" target="_self" class="nav-link">
                        Page 1
                      </a>
                    </li>
                    <li class="nav-item">
                      <a href="/nuxt-02/page2" target="_self" class="nav-link">
                        Page 2
                      </a>
                    </li>
                    <li class="nav-item">
                      <a href="/nuxt-02/page3" target="_self" class="nav-link active nuxt-link-active">
                        Page 3
                      </a>
                    </li>
                  </ul>
                </div> <div class="col-10">
                  <div role="alert" aria-live="polite" aria-atomic="true" class="alert alert-secondary">
                    Page 3 - value = 28
                  </div>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
  <script>
  window.__NUXT__ = (function (a, b, c) {
      return {
        layout: "default", data: [{}], error: null, state: { counter: 28 }, serverRendered: true,
        logs: [
          { date: new Date(1574169916025), args: ["fetch, client=", "false", "serveur=", "true"], type: a, level: b, tag: c },
```

```
68.           { date: new Date(1574169917038), args: ["fetch commit terminé"], type: a, level: b,
    tag: c },
69.           { date: new Date(1574169917137), args: ["[page3 beforeCreate]"], type: a, level: b,
    tag: c },
70.           { date: new Date(1574169917167), args: ["[page3 created], value=", "28"], type: a,
    level: b, tag: c }
71.         ]
72.       }
73.     }("log", 2, ""));</script>
74.   <script src="/nuxt-02/_nuxt/runtime.js" defer></script>
75.   <script src="/nuxt-02/_nuxt/commons.app.js" defer></script>
76.   <script src="/nuxt-02/_nuxt/vendors.app.js" defer></script>
77.   <script src="/nuxt-02/_nuxt/app.js" defer></script>
78. </body>
79. </html>
```

- ligne 52 : on voit que la valeur dans la page reçue est 28 ;
- ligne 65 : dans la réponse du serveur, on voit que le serveur a envoyé au client l'état [state] du store [Vuex]. Grâce à cette information, les scripts client vont pouvoir reconstituer un store [Vuex] ;

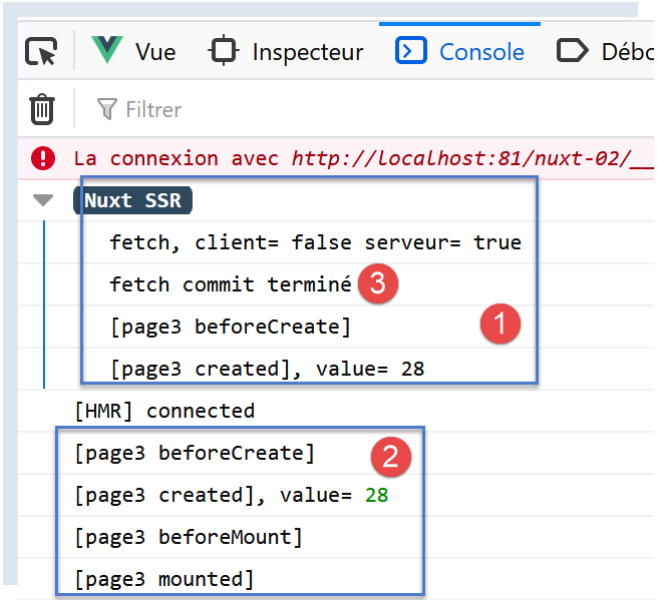
Les scripts client vont à leur tour exécuter le code de la page [page3] :

```
1.  <!-- page3 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-alert slot="right" show variant="secondary"> Page 3 - value = {{ value }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13. /* eslint-disable nuxt/no-timing-in-fetch-data */
14.
15. import Navigation from '@/components/navigation'
16. import Layout from '@/components/layout'
17.
18. export default {
19.   name: 'Page3',
20.   // composants utilisés
21.   components: {
22.     Layout,
23.     Navigation
24.   },
25.   data() {
26.     return {
27.       value: 0
28.     }
29.   },
30.   fetch(context) {
31.     // qui exécute ce code ?
32.     console.log('fetch, client=', process.client, 'serveur=', process.server)
33.     // seulement pour le serveur
34.     if (process.server) {
35.       // on retourne une promesse
36.       return new Promise(function(resolve, reject) {
37.         // on a normalement ici une fonction asynchrone
38.         // on la simule avec une attente d'une seconde
39.         setTimeout(() => {
40.           // succès
41.           resolve()
42.         }, 1000)
43.       }).then(() => {
44.         // on modifie le store
45.         context.store.commit('increment', 28)
46.         // log
47.         console.log('fetch commit terminé')
48.       })
49.     }
50.   },
51.   // cycle de vie
```

```
52.   beforeCreate() {
53.     // client et serveur
54.     console.log('[page3 beforeCreate]')
55.   },
56.   created() {
57.     // client et serveur
58.     this.value = this.$store.state.counter
59.     console.log('[page3 created], value=', this.value)
60.   },
61.   beforeMount() {
62.     // client seulement
63.     console.log('[page3 beforeMount]')
64.   },
65.   mounted() {
66.     // client seulement
67.     console.log('[page3 mounted]')
68.   }
69. }
70. </script>
```

- line 58 : la fonction [created] exécutée par le client affecte la valeur du compteur à la propriété [value] de la ligne 27 ;
- la ligne 7 affiche cette valeur. Puisque c'est la même que celle envoyée par le serveur, on ne voit pas la page 'tressauter' à cause d'une modification ;

Les logs montrent par ailleurs que la fonction [fetch] n'a pas été exécutée par le client :

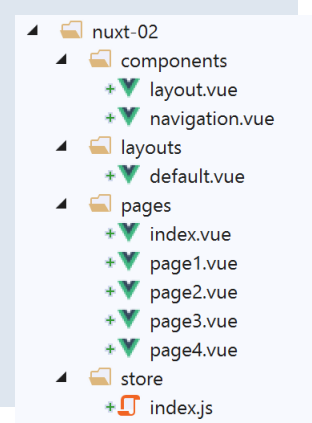


La fonction [fetch] a été exécutée par le serveur [1] mais pas par le client [2]. Par ailleurs, on peut noter que les fonctions du cycle de vie ne sont pas exécutées par le serveur avant la fin de la fonction [fetch] [3]. On peut augmenter la durée de l'attente au sein de la fonction [fetch] pour le vérifier.

Les pages [page1] et [page3] ont montré deux méthodes utilisant le store [Vuex] pour transmettre une information du serveur au client. On peut se demander si elles sont équivalentes. Nous construisons une page [page4] pour le vérifier.

5.5 La page [page4]

Nous ajoutons une nouvelle page [page4] à notre application :



5.5.1 Le composant [navigation]

Le composant [navigation] est modifié pour permettre la navigation vers la nouvelle page :

```
1.  <template>
2.    <!-- menu Bootstrap à cinq options -->
3.    <b-nav vertical>
4.      <b-nav-item to="/" exact exact-active-class="active">
5.        Home
6.      </b-nav-item>
7.      <b-nav-item to="/page1" exact exact-active-class="active">
8.        Page 1
9.      </b-nav-item>
10.     <b-nav-item to="/page2" exact exact-active-class="active">
11.       Page 2
12.     </b-nav-item>
13.     <b-nav-item to="/page3" exact exact-active-class="active">
14.       Page 3
15.     </b-nav-item>
16.     <b-nav-item to="/page4" exact exact-active-class="active">
17.       Page 4
18.     </b-nav-item>
19.   </b-nav>
20. </template>
```

5.5.2 Le code de [page4]

Le code de la page [page4] est le suivant :

```
1.  <!-- page4 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-alert slot="right" show variant="secondary"> Page 4 - value = {{ value }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13.
14. import Navigation from '@/components/navigation'
15. import Layout from '@/components/layout'
16.
17. export default {
18.   name: 'Page4',
19.   // composants utilisés
20.   components: {
21.     Layout,
22.     Navigation
23.   },
```

```
24.   data() {
25.     return {
26.       value: 0
27.     }
28.   },
29.   // cycle de vie
30.   async beforeCreate() {
31.     // client et serveur
32.     console.log('[page4 beforeCreate]')
33.     // seulement pour le serveur
34.     if (process.server) {
35.       // on exécute la fonction asynchrone
36.       const valeur = await new Promise(function(resolve, reject) {
37.         // on a normalement ici une fonction asynchrone
38.         // on la simule avec une attente de 10 secondes
39.         setTimeout(() => {
40.           // succès - on rend la valeur du compteur
41.           resolve(52)
42.         }, 10000)
43.       })
44.       // on modifie le store
45.       this.$store.commit('increment', valeur)
46.       // log
47.       console.log('[page4 beforeCreate], fonction asynchrone terminée, compteur=', this.
    $store.state.counter)
48.     }
49.   },
50.   created() {
51.     // client et serveur
52.     this.value = this.$store.state.counter
53.     console.log('[page4 created], value=', this.value)
54.   },
55.   beforeMount() {
56.     // client seulement
57.     console.log('[page4 beforeMount]')
58.   },
59.   mounted() {
60.     // client seulement
61.     console.log('[page4 mounted]')
62.   }
63. }
64. </script>
```

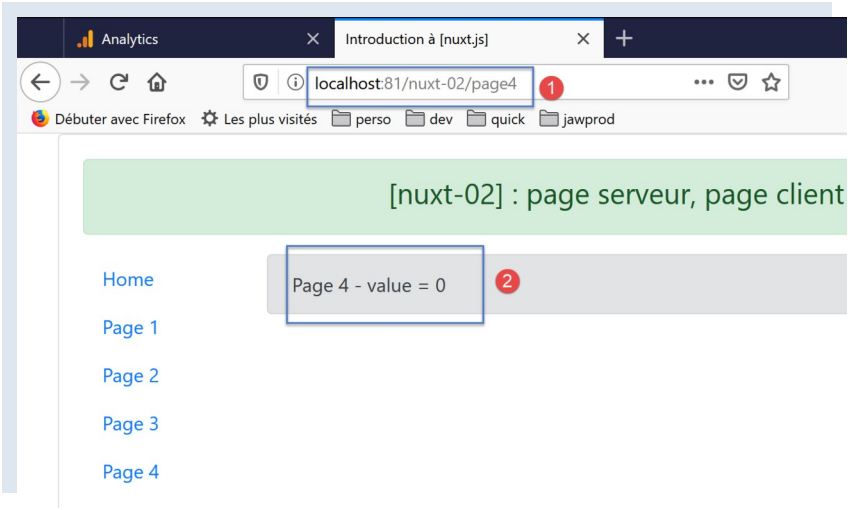
- ligne 30 : ce qui était fait auparavant dans la fonction [fetch] est désormais fait dans la méthode [beforeCreate]. On utilise le couple async (ligne 30) / await (ligne 36) pour attendre la fin de la fonction asynchrone ;
- ligne 36 : on récupère le résultat de la fonction asynchrone rendu ligne 41 après 10 secondes (ligne 42) ;
- lignes 50-54 : dans la méthode [created] exécutée aussi bien côté serveur que côté client, le compteur est affecté à la propriété [value] de la page ;

5.5.3 Exécution

On exécute le projet [nuxt-02] et on tape [localhost:81/nuxt-02/**page4**] à la main pour que le serveur soit sollicité. Comme au démarrage pour la page [index] :

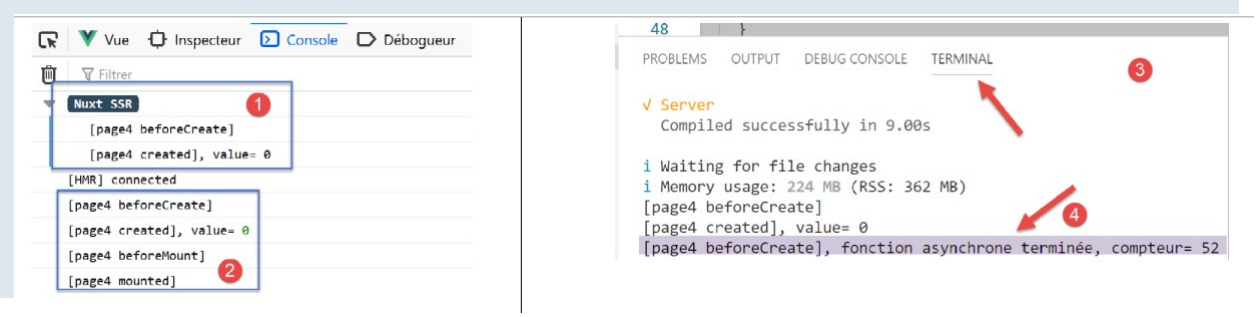
- le serveur exécute la page [page4.vue] ;
- envoie la page générée au navigateur. Celle-ci est affichée ;
- les scripts client embarqués dans la page envoyée prennent la main et exécutent à nouveau la page [page4.vue] ;
- la page affichée est alors modifiée ;

Le résultat final est le suivant :



Contrairement à ce qui était attendu, la valeur affichée en [2] n'est pas 52. Que s'est-il passé ?

Les logs sont les suivants :



On peut remarquer qu'en [1] le log de fin de l'action asynchrone n'a pas été affiché. La fonction [created] qui affiche la valeur du compteur, affiche 0. Tout cela laisse penser que [nuxt] n'a pas attendu la fin de l'action asynchrone.

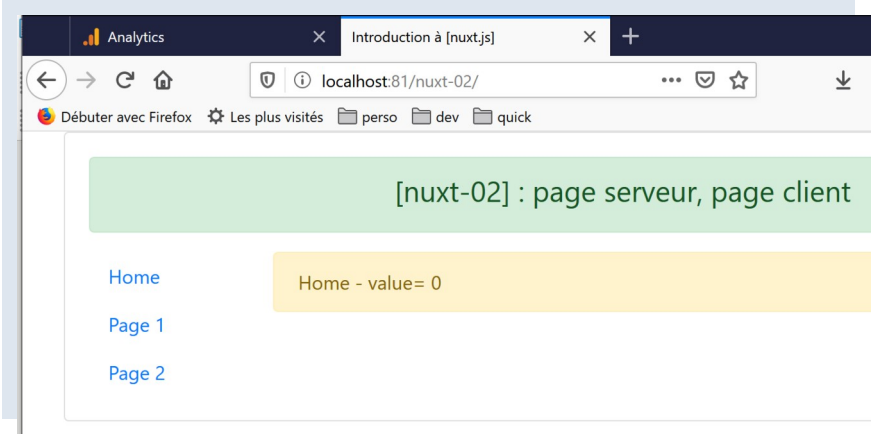
Si on retourne dans le terminal de VSCode qui a servi au lancement de l'application, on trouve les logs [3-4]. On voit que la fonction asynchrone a bien été exécutée côté serveur.

Au final, la fonction [beforeCreate] a bien été exécutée totalement côté serveur, mais [nuxt] n'a pas attendu la fin de son exécution pour envoyer la page au navigateur client alors qu'il attend bien la fin de la fonction [fetch]. C'est donc cette méthode qu'il faut utiliser si on veut que le serveur initialise un store [Vuex].

5.6 Navigation dans l'application [vue]

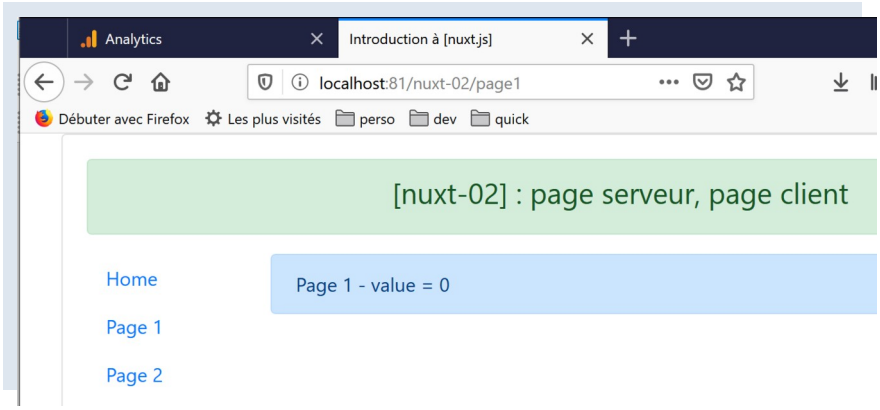
Nous avons montré ce qui se passait lorsque chacune des pages [index, page1, page2, page3, page4] était chargée initialement par le serveur. Dans la pratique, ce n'est pas ce qui se passe : dans un fonctionnement normal, seule la page [index] est cherchée sur le serveur. Voyons les trois pages dans ce cas là :

page [index]



Nous avons déjà expliqué ce résultat au paragraphe lien.

Maintenant cliquons sur le lien [Page 1] :

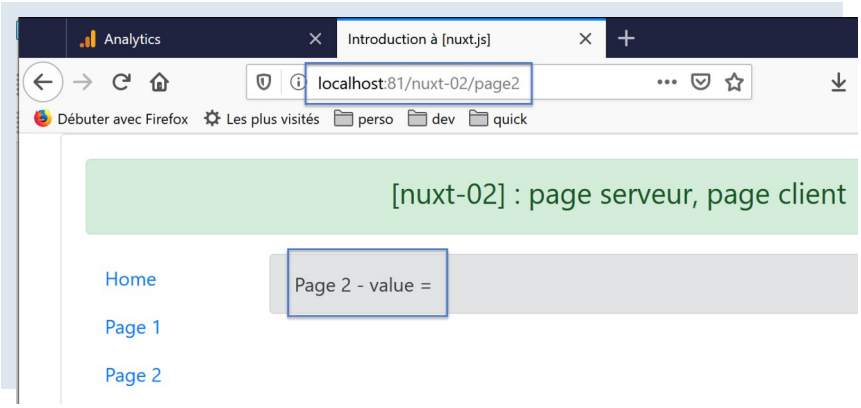


La valeur affichée est 0. C'était 25 lorsque la page était d'abord demandée au serveur en tapant son URL à la main. L'explication est simple. Le code exécuté est le suivant :

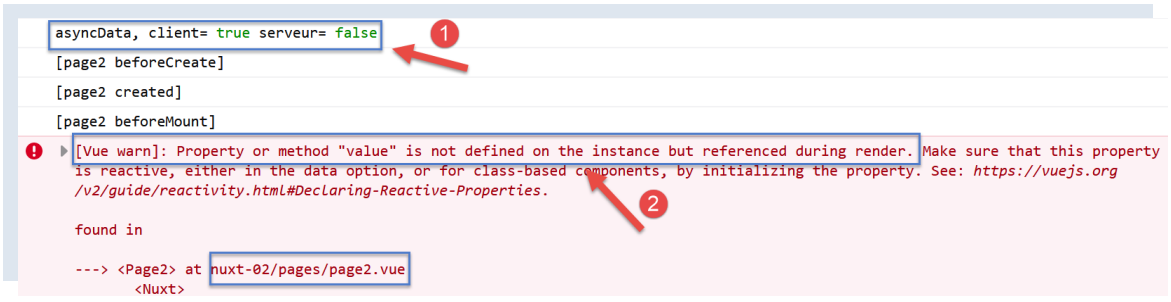
```
1.    created() {
2.      // client et serveur
3.      console.log('[page1 created]')
4.      // serveur seulement
5.      if (process.server) {
6.        this.$store.commit('increment', 25)
7.      }
8.      // client et serveur
9.      this.value = this.$store.state.counter
10.     console.log('value=', this.value)
11. },
```

C'est la ligne 6 qui mettait le compteur à 25. Comme la page n'a pas été demandée au serveur, les lignes 5-7 n'ont pas été exécutées et le compteur du store [Vuex] est resté à 0.

Maintenant, cliquons sur le lien [Page 2] :



Cette fois-ci, aucune valeur n'est affichée et on a de plus un avertissement dans les logs de la console :



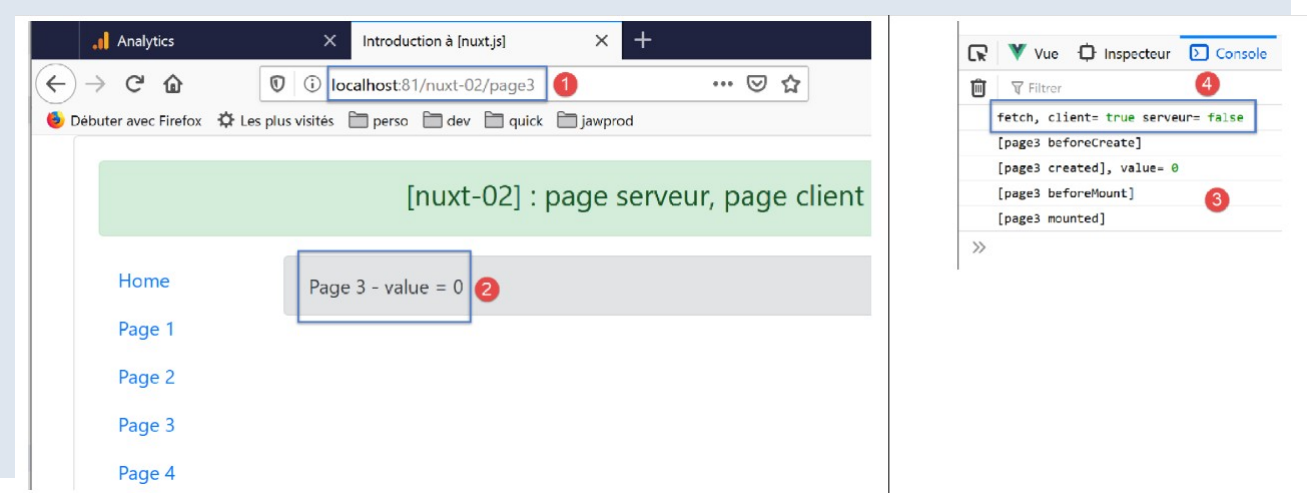
- en [1], on découvre que la fonction [asyncData] a été exécutée par le client. C'est toujours ainsi :
  - elle est exécutée par le serveur si la page est demandée au serveur. dans ce cas, elle n'est pas exécutée par le client ;
  - puis à chaque fois que la page est la **cible** de la route courante du client ;
- en [2] : [nuxt] émet un avertissement parce que dans le template de la page, il y a une expression réactive `{{ value }}` alors que la page n'a pas de propriété [value] ;

Rappelons le code exécuté par le client :

```
1.  asyncData() {
2.      // qui exécute ce code ?
3.      console.log('asyncData, client=', process.client, 'serveur=', process.server)
4.      // seulement pour le serveur
5.      if (process.server) {
6.        // on retourne une promesse
7.        return new Promise(function(resolve, reject) {
8.          // on a normalement ici une fonction asynchrone - pas ici donc
9.          // ce résultat sera inclus dans les propriétés de [data]
10.         resolve({ value: 87 })
11.       })
12.     }
13.   },
```

- la ligne 3 a permis de voir que la fonction [asyncData] a été exécutée par le client avant même les fonctions du cycle de vie ;
- la ligne 5 a empêché l'exécution du reste du code qui créait la propriété [value] car ce code est exécuté par le client ;

Maintenant passons à la page [page3] :



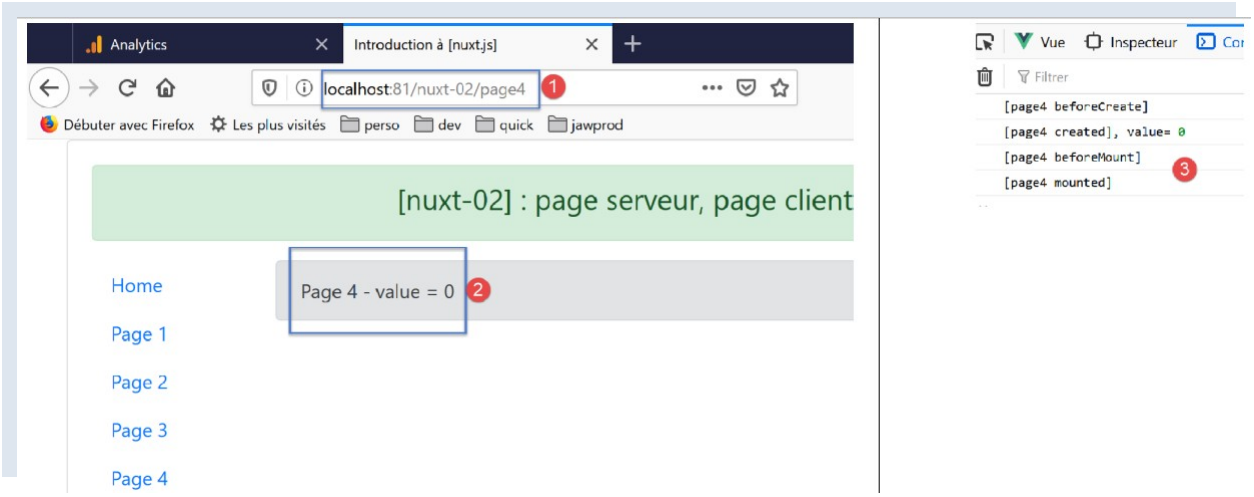
- en [2], on avait 28 lorsque la page était fournie par le serveur. Ici ce n'est pas le cas ;
- en [4], on voit que la fonction [fetch] a été exécutée côté client ;

Examinons le code exécuté par le client :

```
1.  ...
2.  fetch(context) {
3.      // qui exécute ce code ?
4.      console.log('fetch, client=', process.client, 'serveur=', process.server)
5.      // seulement pour le serveur
6.      if (process.server) {
7.        // on retourne une promesse
8.        return new Promise(function(resolve, reject) {
9.          // on a normalement ici une fonction asynchrone
10.         // on la simule avec une attente d'une seconde
11.         setTimeout(() => {
12.           // succès
13.           resolve()
14.         }, 1000)
15.       }).then(() => {
16.         // on modifie le store
17.         context.store.commit('increment', 28)
18.         // log
19.         console.log('fetch commit terminé')
20.       })
21.     }
22. },
23. ...
```

- le client exécute la méthode [fetch], ligne 2 ;
- les lignes 6-21 ne sont pas exécutées car la condition [process.server] est fausse. Donc la ligne 17 qui met le compteur à 28 n'est pas exécuté. Il reste à zéro. C'est pourquoi le client affiche 0 au lieu de 28 ;

Passons maintenant à la page [page4]. On a le résultat suivant :



- en [2], la valeur du compteur ;
- en [3], les logs du client ;

Le code exécuté par le client est le suivant :

```
1.  ...
2.  data() {
3.      return {
4.        value: 0
5.      }
6.    },
7.    // cycle de vie
8.    async beforeCreate() {
9.      // client et serveur
10.     console.log('[page4 beforeCreate]')
11.     // seulement pour le serveur
12.     if (process.server) {
13.       // on exécute la fonction asynchrone
14.       const valeur = await new Promise(function(resolve, reject) {
15.         // on a normalement ici une fonction asynchrone
16.         // on la simule avec une attente de 10 secondes
17.         setTimeout(() => {
18.           // succès - on rend la valeur du compteur
19.           resolve(52)
20.         }, 10000)
21.       })
22.       // on modifie le store
23.       this.$store.commit('increment', valeur)
24.       // log
25.       console.log('[page4 beforeCreate], fonction asynchrone terminée, compteur=', this.
    $store.state.counter)
26.     }
27.   },
28.   created() {
29.     // client et serveur
30.     this.value = this.$store.state.counter
31.     console.log('[page4 created], value=', this.value)
32.   },
33. ...
```

- les lignes 12-26 ne sont pas exécutées par le client, car la condition [process.server] de la ligne 12 est fausse. Donc le compteur du store [Vuex] vaut 0 (sa valeur initiale dans le store) et c'est cette valeur qu'affiche la page ;

On peut se demander ce qui se passe lorsqu'on met en commentaires le [if] des lignes 12 et 26. Voici la réponse :

- le client exécute cette fois les lignes 14-25 mais [nuxt] n'attend pas la fin de la fonction asynchrone (comme pour le serveur) et laisse donc le compteur à 0 ;
- au bout de 10 s, la fonction asynchrone se termine et le compteur est mis à 52 ligne 23 ;

- lorsqu'on navigue de nouveau vers la page [page4], c'est alors la valeur 52 qui est affichée ;

5.7 Résumé

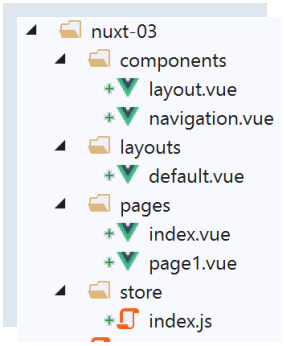
De nos différents essais, on retiendra les points suivants :

- si la page [index] doit contenir des données externes, celles-ci peuvent être cherchées par le serveur avec la fonction [asyncData] ;
- si le serveur doit initialiser un store [Vuex] avec des données externes au chargement de la page [index], il le fera dans la fonction [fetch] ;
- la page générée par le serveur et la page générée par le client doivent être **identiques** si on veut éviter l'effet de 'tressautement' provoqué par le fait que la page du client vient visuellement remplacer la page envoyée par le serveur et initialement affichée ;

Nous allons découvrir d'autres aspects de [nuxt] au travers d'un nouvel exemple.

6 Exemple [nuxt-03] : nuxtServerInit

Le projet [nuxt-03] vise à présenter une fonction du store [Vuex] appelée [nuxtServerInit]. Elle permet au serveur d'initialiser le store [Vuex] comme le fait la fonction [fetch]. Mais contrairement à la fonction [fetch], la fonction [nuxtServerInit] n'est jamais exécutée par le client.



Le projet [nuxt-03] est initialement obtenu par recopie du projet [nuxt-01] duquel on supprime la page [page2] du dossier [pages] et du composant [navigation]. Le dossier [store] est obtenu par recopie du dossier [nuxt-02/store].

6.1 Le store [Vuex]

Le store [Vuex] sera implémenté par le fichier [store/index.js] suivant :

```
1.  /* eslint-disable no-console */
2.  export const state = () => ({
3.    // compteur
4.    counter: 0
5.  })
6.
7.  export const mutations = {
8.    // incrémentation du compteur d'une valeur [inc]
9.    increment(state, inc) {
10.     state.counter += inc
11.   }
12. }
13.
14. export const actions = {
15.   async nuxtServerInit(store, context) {
16.     // qui exécute ce code ?
17.     console.log('nuxtServerInit, client=', process.client, 'serveur=', process.server)
18.     // on attend la fin d'une promesse
19.     await new Promise(function(resolve, reject) {
20.       // on a normalement ici une fonction asynchrone
21.       // on la simule avec une attente d'une seconde
22.       setTimeout(() => {
23.         // succès
24.         resolve()
25.       }, 1000)
26.     })
27.     // on modifie le store
28.     store.commit('increment', 34)
29.     // log
30.     console.log('nuxtServerInit commit terminé')
31.   }
32. }
```

- lignes 1-12 : sont analogues à ce qu'elles étaient dans le projet [nuxt-02] ;
- lignes 14-32 : on exporte un objet [actions]. C'est un terme réservé du store de [Vuex] ;
- ligne 15 : on définit la fonction [nuxtServerInit]. Celle-ci sera exécutée au démarrage de l'application par le serveur. Son rôle usuel est d'initialiser un store [Vuex] à l'aide de données externes obtenues avec une fonction asynchrone. [nuxt] attend que celle-ci rende ses résultats avant d'entamer le cycle de vie de la page demandée. La fonction reçoit deux paramètres :

    - le store [Vuex] à initialiser ;
    - le contexte [nuxt] du moment ;
- lignes 19-26 : on attend la fin de l'action asynchrone, ici une attente artificielle d'une seconde (ligne 15) ;
- ligne 28 : on donne au compteur la valeur 34 ;
- lignes 17 et 30 : des logs pour suivre le déroulement de l'exécution de la fonction [nuxtServerInit] ;

6.2 La page [index]

La page [index] sera la suivante :

```
1.  <!-- page [index] -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="warning"> Home - value= {{ value }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-undef */
13. /* eslint-disable no-console */
14. /* eslint-disable nuxt/no-env-in-hooks */
15.
16. import Layout from '@/components/layout'
17. import Navigation from '@/components/navigation'
18. export default {
19.   name: 'Home',
20.   // composants utilisés
21.   components: {
22.     Layout,
23.     Navigation
24.   },
25.   data() {
26.     return {
27.       value: 0
28.     }
29.   },
30.   // cycle de vie
31.   beforeCreate() {
32.     // client et serveur
33.     console.log('[home beforeCreate]')
34.   },
35.   created() {
36.     // client et serveur
37.     this.value = this.$store.state.counter
38.     console.log('[home created], value=', this.value)
39.   },
40.   beforeMount() {
41.     // client seulement
42.     console.log('[home beforeMount]')
43.   },
44.   mounted() {
45.     // client seulement
46.     console.log('[home mounted]')
47.   }
48. }
49. </script>
```

- ligne 37 : la valeur du compteur initialisé par la fonction [nuxtServerInit] est affectée à la propriété [value] de la ligne 27. Cette valeur est affichée par la ligne 7 ;
- la ligne 37 sera exécutée aussi bien par le serveur que par le client. Dans les deux cas, la propriété [value] recevra la même valeur ce qui assure l'identité de la page générée par le serveur avec celle générée par le client ;

6.3 La page [page1]

La page [page1] est obtenue par recopie de la page [index]. On modifie ensuite son texte pour remplacer [home] par [page1] :

```
<!-- page [page1]] -->
<template>
  <Layout :left="true" :right="true">
    <!-- navigation -->
    <Navigation slot="left" />
    <!-- message-->
    <b-alert slot="right" show variant="warning"> Page1 - value= {{ value }} </b-alert>
  </Layout>
</template>

<script>
/* eslint-disable no-undef */
/* eslint-disable no-console */
/* eslint-disable nuxt/no-env-in-hooks */

import Layout from '@/components/layout'
import Navigation from '@/components/navigation'

export default {
  name: 'Page1',
  // composants utilisés
  components: {
    Layout,
    Navigation
  },
  data() {
    return {
      value: 0
    }
  },
  // cycle de vie
  beforeCreate() {
    // client et serveur
    console.log('[page1 beforeCreate]')
  },
  created() {
    // client et serveur
    this.value = this.$store.state.counter
    console.log('[page1 created], value=', this.value)
  },
  beforeMount() {
    // client seulement
    console.log('[page1 beforeMount]')
  },
  mounted() {
    // client seulement
    console.log('[page1 mounted]')
  }
}
</script>
```

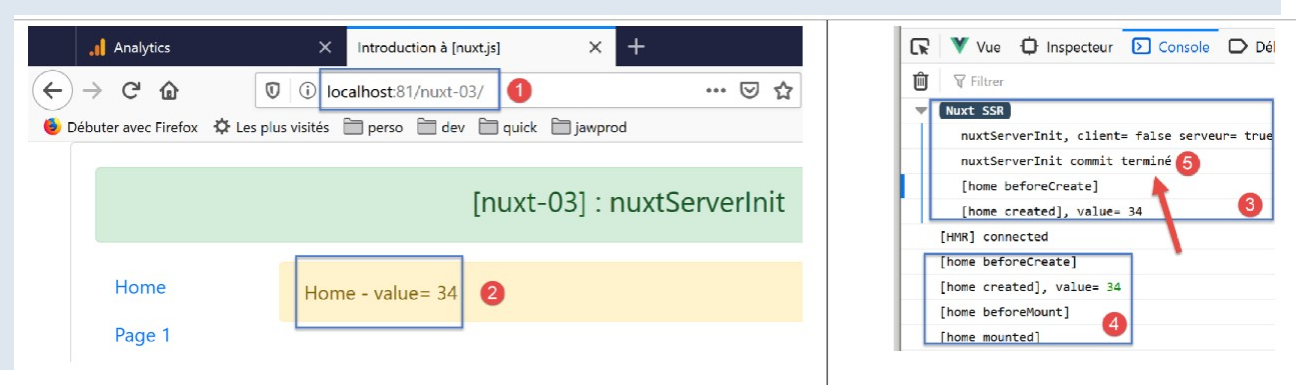
Cette page n'est là que pour rendre possible la navigation entre deux pages.

6.4 Exécution

Le fichier [nuxt.config.js] est modifié de la façon suivante :

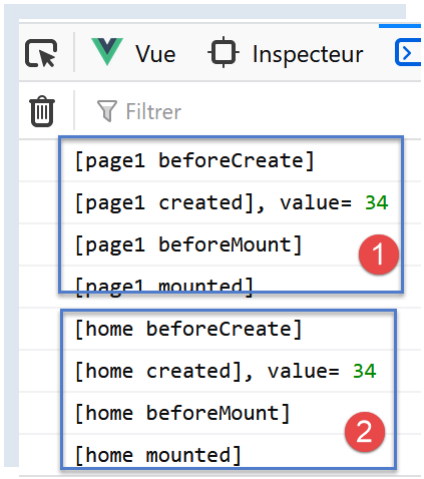
```
// répertoire du code source
  srcDir: 'nuxt-03',
  // routeur
  router: {
    // racine des URL de l'application
    base: '/nuxt-03/'
  },
  // serveur
  server: {
    // port de service, 3000 par défaut
    port: 81,
    // adresses réseau écoutées, par défaut localhost : 127.0.0.1
    // 0.0.0.0 = toutes les adresses réseau de la machine
    host: 'localhost'
}
```

La page affichée à l'exécution est alors la suivante :



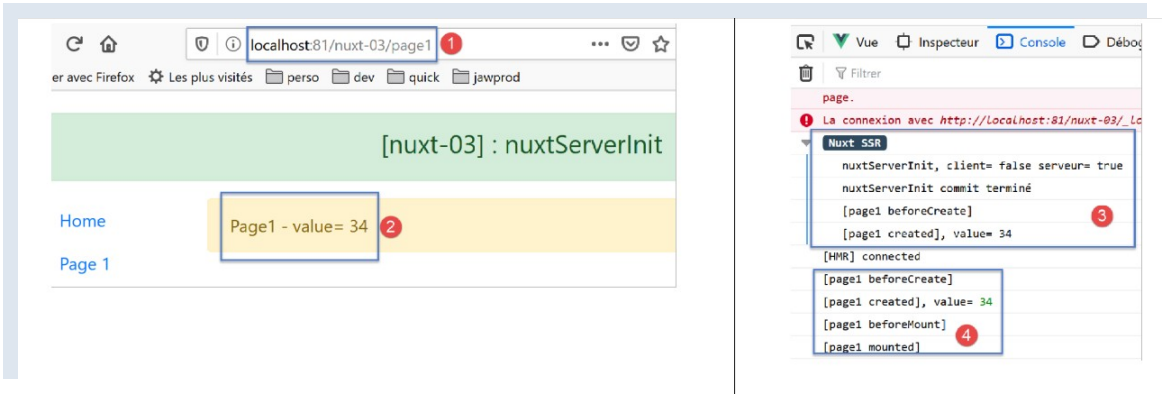
- en [5], on voit que la fonction [nuxtServerInit] a été exécutée par le serveur avant le cycle de vie de la page [index]. [nuxt] a attendu que la fonction asynchrone ait terminé son travail avant de passer au cycle de vie ;
- en [4], on voit que le client n'a pas exécuté la fonction [nuxtServerInit] ;

Maintenant naviguons deux fois : index --> page1 --> index. Les logs sont alors les suivants :



- en [1-2], on voit que la fonction [nuxtServerInit] n'est pas exécutée par le client ;

Maintenant tapons l'URL de la page [page1] à la main pour forcer un appel au serveur :



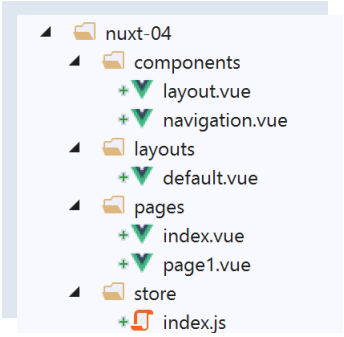
en [3-4], on retrouve le même mécanisme que celui qui avait précédé le chargement de la page [index] au démarrage. On rappelle ici ce qui a déjà été dit : lorsqu'on force l'appel d'une page au serveur, tout se passe comme si l'application redémarrait avec une page d'accueil qui serait la page demandée ;

7 Exemple [nuxt-04] : maintien d'une session client / serveur

Le projet [nuxt-04] aborde le problème du maintien d'une session client / serveur. On reprend le projet [nuxt-03] avec les modifications suivantes :

- la page [index] aura un bouton qui permettra d'incrémenter le compteur du store [Vuex] ;
- la page [page1] reste inchangée ;
- on voudrait que lorsqu'une page est demandée à la main au serveur, celui-ci renvoie la page demandée avec un store [Vuex] qui aurait pour valeur de compteur, la dernière valeur que celui-ci avait côté client ;

Le projet [nuxt-04] est initialement créé par recopie du projet [nuxt-03] :



Seule la page [index] change :

```
1.  <!-- page [index] -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <template slot="right">
8.        <b-alert show variant="warning"> Home - value= {{ value }} </b-alert>
9.        <!-- bouton -->
10.       <b-button @click="incrementCounter" class="ml-3" variant="primary">Incrémenter</b-button>
11.     </template>
12.   </Layout>
13. </template>
14.
15. <script>
16. /* eslint-disable no-undef */
17. /* eslint-disable no-console */
18. /* eslint-disable nuxt/no-env-in-hooks */
19.
20. import Layout from '@/components/layout'
21. import Navigation from '@/components/navigation'
22. export default {
23.   name: 'Home',
24.   // composants utilisés
25.   components: {
26.     Layout,
27.     Navigation
28.   },
29.   data() {
30.     return {
31.       value: 0
32.     }
33.   },
34.   // cycle de vie
35.   beforeCreate() {
36.     // client et serveur
37.     console.log('[home beforeCreate]')
38.   },
39.   created() {
40.     // client et serveur
```

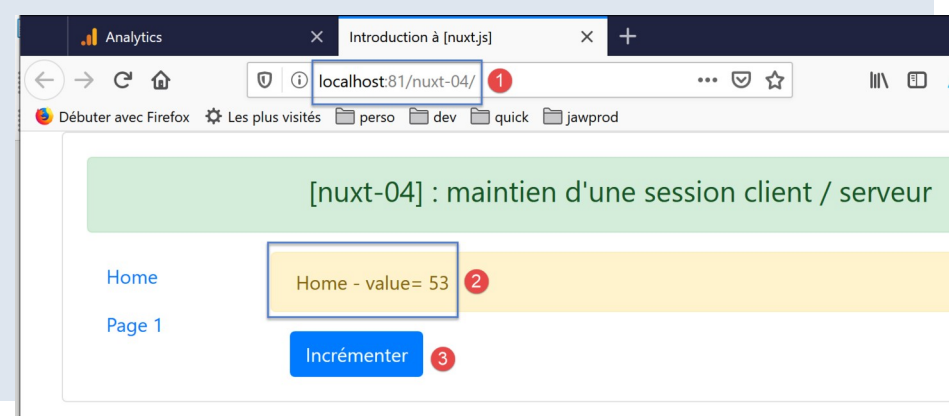
```
41.     this.value = this.$store.state.counter
42.     console.log('[home created], value=', this.value)
43.   },
44.   beforeMount() {
45.     // client seulement
46.     console.log('[home beforeMount]')
47.   },
48.   mounted() {
49.     // client seulement
50.     console.log('[home mounted]')
51.   },
52.   // gestion des évts
53.   methods: {
54.     incrementCounter() {
55.       console.log('incrementCounter')
56.       // incrément du compteur de 1
57.       this.$store.commit('increment', 1)
58.       // chgt de la valeur affichée
59.       this.value = this.$store.state.counter
60.     }
61.   }
62. }
63. </script>
```

- ligne 10 : on a ajouté un bouton pour incrémenter le compteur du store [Vuex] ;
- ligne 54 : la méthode qui gère le [clic] sur le bouton [Incrémenter] ;
- ligne 57 : le compteur du store est incrémenté de 1 ;
- ligne 59 : la valeur du compteur est affectée à la propriété [value] afin qu'elle soit affichée par la ligne 8 ;

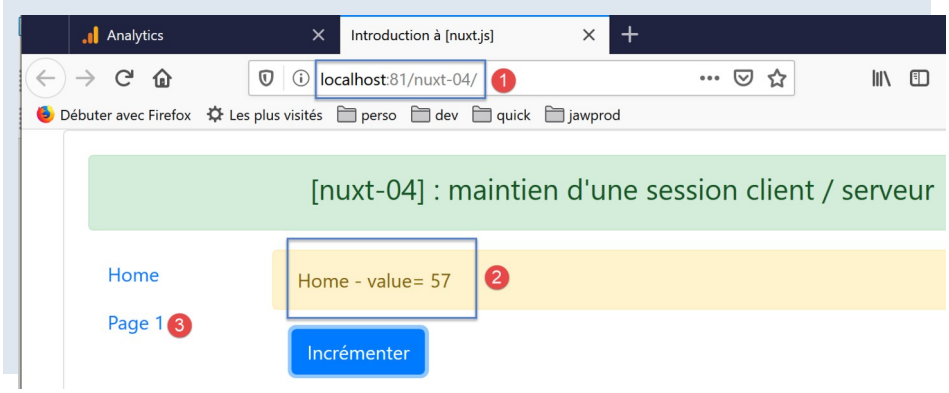
On exécute le projet [nuxt-04] avec le fichier [nuxt.config.js] suivant :

```
1.  // répertoire du code source
2.    srcDir: 'nuxt-04',
3.    // routeur
4.    router: {
5.      // racine des URL de l'application
6.      base: '/nuxt-04/'
7.    },
8.    // serveur
9.    server: {
10.     // port de service, 3000 par défaut
11.     port: 81,
12.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
13.     // 0.0.0.0 = toutes les adresses réseau de la machine
14.     host: 'localhost'
15. }
```

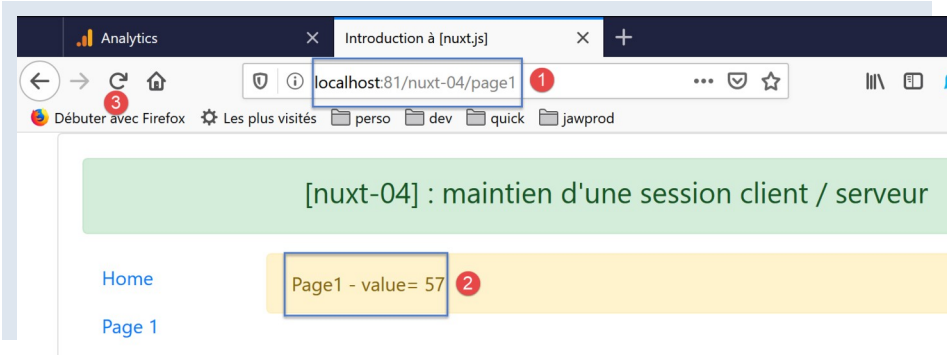
A l'exécution la première page affichée est la suivante :



En utilisant plusieurs fois le bouton [3], on a la nouvelle page suivante :



Si on utilise le lien [3], on a la page suivante :



- en [2], la page [page1] [1] affiche bien la valeur du compteur ;

Maintenant, rafraîchissons la page avec [3]. La nouvelle page est la suivante :

- en [2], on a perdu la valeur courante du compteur. On est revenu à sa valeur initiale ;

Ce résultat est tout a fait compréhensible si on regarde les logs :

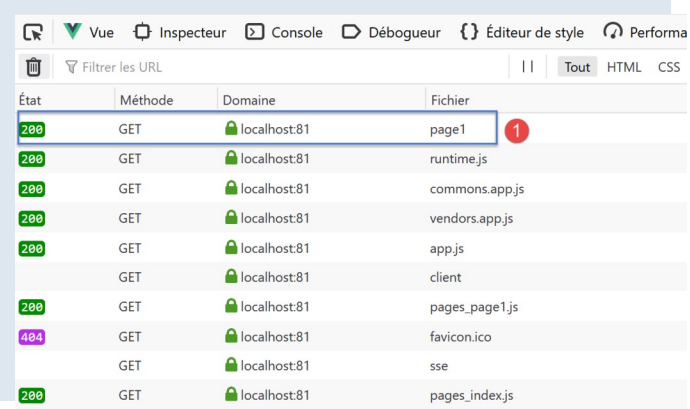
- en [1], le serveur a réexécuté la fonction [nuxtServerInit]. Or celle-ci est la suivante :

```
1.  /* eslint-disable no-console */
2.  export const state = () => ({
3.    // compteut
4.    counter: 0
5.  })
```

```
6.
7.  export const mutations = {
8.    // incrémentation du compteur d'une valeur [inc]
9.    increment(state, inc) {
10.     state.counter += inc
11.   }
12. }
13.
14. export const actions = {
15.   async nuxtServerInit(store, context) {
16.     // qui exécute ce code ?
17.     console.log('nuxtServerInit, client=', process.client, 'serveur=', process.server)
18.     // on attend la fin d'une promesse
19.     await new Promise(function(resolve, reject) {
20.       // on a normalement ici une fonction asynchrone
21.       // on la simule avec une attente d'une seconde
22.       setTimeout(() => {
23.         // succès
24.         resolve()
25.       }, 1000)
26.     })
27.     // on modifie le store
28.     store.commit('increment', 53)
29.     // log
30.     console.log('nuxtServerInit commit terminé')
31.   }
32. }
```

La ligne 28 affecte la valeur 53 au compteur.

Examinons la requête HTTP faite par le navigateur lorsqu'on a rafraîchi la page [page1] :



On voit qu'outre la page [page1] [1], le client demande un certain nombre de scripts au serveur. On notera les scripts [pages_index, pages_page1] qui sont les scripts associés aux pages [index, page]. Ces scripts sont fournis à chaque requête au serveur quelque soit la page demandée ;

En [1], la page [page1] est demandée au serveur avec la requête HTTP suivante :

```
1.  GET http://localhost:81/nuxt-04/page1
2.  Host: localhost:81
3.  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0) Gecko/20100101 Firefox/70.0
4.  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5.  Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
6.  Accept-Encoding: gzip, deflate
7.  DNT: 1
8.  Connection: keep-alive
9.  Upgrade-Insecure-Requests: 1
10. Pragma: no-cache
11. Cache-Control: no-cache
```

On voit que cette requête ne transmet aucune information au serveur. Celui-ci n'a donc aucun moyen de connaître l'état du store [Vuex] côté client. Il aurait fallu pour cela que le client lui envoie cette information.

Dans le projet suivant [nuxt-05] nous utilisons un cookie pour que le client puisse envoyer de l'information au serveur lorsque celui-ci est sollicité.

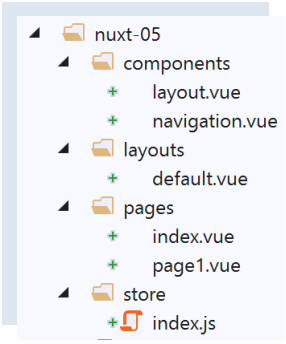
8 Exemple [nuxt-05] : persistance du store avec un cookie de session

Objectif : on voudrait que le store [Vuex] ne soit pas réinitialisé à chaque requête vers le serveur. Pour ce faire nous allons utiliser un cookie de session :

- le store sera initialisé par le serveur et mis par celui-ci dans un cookie de session ;
- le navigateur client recevra ce cookie de session et mécaniquement l'enverra à chaque nouvelle requête vers le serveur ;
- le serveur pourra alors récupérer ce cookie de session et travailler avec le store qu'il contient, un store mis à jour par le client ;

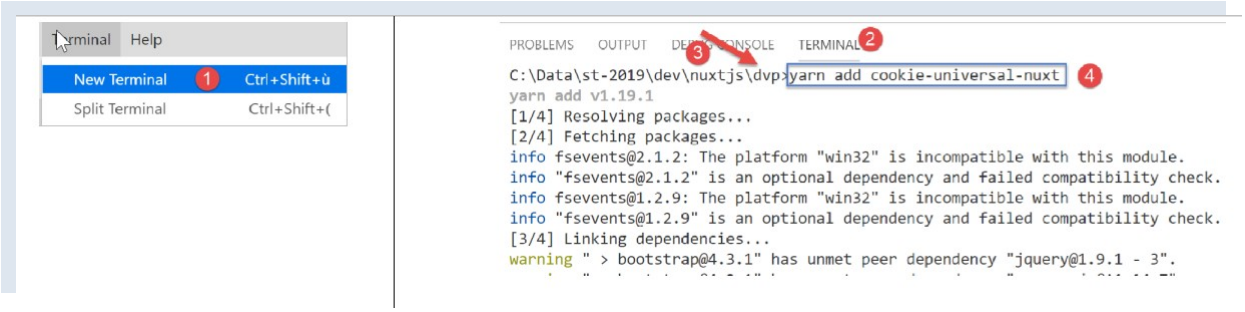
8.1 Présentation

Le projet [nuxt-05] est obtenu initialement par recopie du projet [nuxt-04] :



Nous allons voir que seul le fichier [store / index.js] va changer.

Pour utiliser des cookies avec [nuxt], nous allons utiliser le module [cookie-universal-nuxt] que nous installlons avec [yarn] dans un terminal VSCode :



- en [4], on tape la commande [yarn add cookie-universal-nuxt] ;

Un nouveau module est ainsi ajouté au fichier [package.json] du projet [dvp] :

```
1.  ...
2.  },
3.    "dependencies": {
4.      "@nuxtjs/axios": "^5.3.6",
5.      "bootstrap": "^4.1.3",
6.      "bootstrap-vue": "^2.0.0",
7.      "cookie-universal-nuxt": "^2.0.19",
8.      "nuxt": "^2.0.0"
9.  },
```

8.2 Le fichier de configuration [nuxt.config.js]

Pour que [nuxt] puisse utiliser les cookies de [cookie-universal-nuxt], il faut déclarer ce module dans le fichier de configuration [nuxt.config.js] :

```
1.  ...
2.  ],
3.    /*
4.     ** Nuxt.js modules
5.     */
6.    modules: [
7.      // Doc: https://bootstrap-vue.js.org
8.      'bootstrap-vue/nuxt',
9.      // Doc: https://axios.nuxtjs.org/usage
10.     '@nuxtjs/axios',
11.     // https://www.npmjs.com/package/cookie-universal-nuxt
12.     'cookie-universal-nuxt'
13.   ],
14. ...
```

- ligne 12, le module [cookie-universal-nuxt] est ajouté au tableau des modules [6] de [nuxt] ;

Le fichier [nuxt.config.js] est au final le suivant :

```
1.  export default {
2.    mode: 'universal',
3.    /*
4.     ** Headers of the page
5.     */
6.    head: {
7.      title: 'Introduction à [nuxt.js]',
8.      meta: [
9.        { charset: 'utf-8' },
10.       { name: 'viewport', content: 'width=device-width, initial-scale=1' },
11.       {
12.         hid: 'description',
13.         name: 'description',
14.         content: 'ssr routing loading asyncdata middleware plugins store'
15.       }
16.     ],
17.     link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
18.   },
19.   /*
20.    ** Customize the progress-bar color
21.    */
22.   loading: { color: '#fff' },
23.   /*
24.    ** Global CSS
25.    */
26.   css: [],
27.   /*
28.    ** Plugins to load before mounting the App
29.    */
30.   plugins: [],
31.   /*
32.    ** Nuxt.js dev-modules
33.    */
34.   buildModules: [
35.     // Doc: https://github.com/nuxt-community/eslint-module
36.     '@nuxtjs/eslint-module'
37.   ],
38.   /*
39.    ** Nuxt.js modules
40.    */
41.   modules: [
42.     // Doc: https://bootstrap-vue.js.org
43.     'bootstrap-vue/nuxt',
44.     // Doc: https://axios.nuxtjs.org/usage
45.     '@nuxtjs/axios',
46.     // https://www.npmjs.com/package/cookie-universal-nuxt
47.     'cookie-universal-nuxt'
48.   ],
```

```
49.   /*
50.    ** Axios module configuration
51.    ** See https://axios.nuxtjs.org/options
52.    */
53.   axios: {},
54.   /*
55.    ** Build configuration
56.    */
57.   build: {
58.     /*
59.      ** You can extend webpack config here
60.      */
61.     extend(config, ctx) {}
62.   },
63.   // répertoire du code source
64.   srcDir: 'nuxt-05',
65.   // routeur
66.   router: {
67.     // racine des URL de l'application
68.     base: '/nuxt-05/'
69.   },
70.   // serveur
71.   server: {
72.     // port de service, 3000 par défaut
73.     port: 81,
74.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
75.     // 0.0.0.0 = toutes les adresses réseau de la machine
76.     host: 'localhost'
77.   },
78.   // environnement
79.   env: {
80.     maxAge: 60 * 5
81.   }
82. }
```

- ligne 79 : on a ajouté la clé [env] au fichier. Cette clé est un mot réservé. Les éléments déclarés dans cet objet sont disponibles à partir de l'objet [context.env] dans les éléments de l'application ;
- ligne 80 : l'attribut [maxAge] sera la durée de vie maximale du cookie de session, durée qui se mesure à partir de la dernière fois que le cookie a été initialisé. Cette durée s'exprime en secondes. On a mis ici une durée de vie de 5 minutes ;

8.3 Le principe de la persistance du store

Les cookies échangés entre le client et le serveur sont disponibles de chaque côté (client et serveur) dans :

- [context.app.$cookies] là ou l'objet [context] est disponible, ç-à-d à peu près partout ;
- [this.$cookies] à l'intérieur d'une vue ;

On obtient un cookie particulier avec l'expression [...$cookies.**get**('nom_du_cookie')]. On fixe la valeur d'un cookie avec l'expression [...$cookies.**set**('nom_du_cookie', valeur_du_cookie)].

Le principe du cookie de persistance du store sera le suivant :

- lorsque le serveur va initialiser le store dans la fonction [nuxtServerInit], l'état du store sera stocké dans un cookie nommé 'session' ;
- le cookie 'session' fera alors partie de la réponse HTTP du serveur. On sait qu'un navigateur renvoie au serveur les cookies que celui-ci lui a envoyés. Il le fait à chaque nouvelle requête qu'il fait au serveur. On sait également que le serveur envoie le store à l'intérieur la page qu'il envoie au client ;
- au sein du navigateur, l'application cliente récupère le store envoyé par le serveur et va ensuite faire son travail. On fera en sorte qu'à chaque fois qu'elle modifie le store, le nouvel état de celui-ci soit stocké dans le cookie 'session' enregistré par le navigateur ;
- si l'utilisateur force un appel au serveur, le navigateur client renverra automatiquement tous les cookies que le serveur lui a précédemment envoyés, notamment le cookie nommé 'session' ;
- lorsque suite à cet appel, le serveur va réinitialiser de nouveau le store, il récupèrera le cookie nommé 'session' et initialisera l'état du store avec la valeur de celui-ci ;
- il y aura donc continuité du store entre le client et le serveur ;

8.4 Initialisation du store

Le store est implémenté dans le fichier [store / index.js] :

```
1.  /* eslint-disable no-console */
2.  export const state = () => ({
3.    // compteur
4.    counter: 0
5.  })
6.
7.  export const mutations = {
8.    // incrémentation du compteur d'une valeur [inc]
9.    increment(state, inc) {
10.     state.counter += inc
11.   },
12.   // remplacement du state
13.   replace(state, newState) {
14.     for (const attr in newState) {
15.       state[attr] = newState[attr]
16.     }
17.   }
18. }
19.
20. export const actions = {
21.   async nuxtServerInit(store, context) {
22.     // qui exécute ce code ?
23.     console.log('nuxtServerInit, client=', process.client, 'serveur=', process.server, 'env=',
    context.env)
24.     // on attend la fin d'une promesse
25.     await new Promise(function(resolve, reject) {
26.       // on a normalement ici une fonction asynchrone
27.       // on la simule avec une attente d'une seconde
28.       setTimeout(() => {
29.         // init session
30.         initStore(store, context)
31.         // succès
32.         resolve()
33.       }, 1000)
34.     })
35.   }
36. }
37.
38. function initStore(store, context) {
39.   // y-a-t-il un cookie de session dans la requête en cours
40.   const cookies = context.app.$cookies
41.   const session = cookies.get('session')
42.   if (!session) {
43.     // pas de session existante
44.     console.log("nuxtServerInit, initialisation d'une nouvelle session")
45.     // on initialise le store
46.     store.commit('increment', 77)
47.   } else {
48.     console.log("nuxtServerInit, reprise d'une session existante")
49.     // on met à jour le store avec le cookie de session
50.     store.commit('replace', session.store)
51.   }
52.   // on met le store dans le cookie de session
53.   cookies.set('session', { store: store.state }, { path: context.base, maxAge:
    context.env.maxAge })
54.   // log
55.   console.log('initStore terminé, store=', store.state)
56. }
```

Commentaires

- lignes 2-5 : le store sera constitué d'un compteur ;
- lignes 9-11 : ce compteur pourra être incrémenté ;
- lignes 13-17 : l'état du store pourra être initialisé à partir d'un nouvel état. Cette fonction est là pour montrer une initialisation possible du store lorsque celui-ci n'est pas limité au seul compteur comme ici ;
- lignes 21-35 : la fonction [nuxtServerInit] n'a pas changé ;

- ligne 30 : lorsque le temps d'attente d'une seconde est écoulé, on initialise le store à l'aide de la fonction des lignes 38-56 ;
- lignes 40-41 : on commence par récupérer le cookie nommé 'session' :

1. lors de la 1ère exécution de l'application et lors de la 1ère requête faite au serveur, ce cookie **n'existera pas encore**. Il sera alors créé (ligne 53) et sera envoyé au navigateur client ;
2. lors de la même exécution de l'application et lors des requêtes n°s 2, 3, ... faites au serveur, ce cookie **existera** car le navigateur client le renverra avec chaque nouvelle requête faite au serveur ;
3. lors d'une seconde exécution de l'application et lors de la 1ère requête faite au serveur, ce cookie **peut exister également**. En effet, à l'issue de l'étape 1, le cookie a été stocké sur le navigateur avec une certaine durée de vie. Si cette durée de vie n'est pas dépassée, le cookie nommé 'session' sera envoyé avec la 1ère requête faite au serveur

En résumé, pour chaque requête faite au serveur : si le cookie 'session' est déjà stocké sur le navigateur client alors le serveur va le recevoir, sinon il ne le recevra pas.

- lignes 42-47 : si le serveur ne reçoit pas le cookie de session, alors le store est initialisé par la ligne 46 ;
  - puis ligne 53, un cookie nommé 'session' sera créé et placé dans la réponse HTTP du serveur. La valeur du cookie est l'objet [`{ store: store.state }`]. C'est donc l'état du store et non le store lui-même qui est mis dans le cookie de session ;
  - le 3ième paramètre de la fonction [set] est un objet d'options :
    - [path] indique à quelle URL ce cookie devra être renvoyé. [context.base] est l'URL de base de l'application [nuxt-05]. Celle-ci est définie dans le fichier [nuxt.config.js] :

```
1.    // routeur
2.    router: {
3.      // racine des URL de l'application
4.      base: '/nuxt-05/'
5.  },
```

    - [maxAge] est la durée de vie en secondes du cookie sur le navigateur. Passée cette durée, le navigateur ne le renvoie plus au serveur. [`context.env.maxAge`] renvoie là encore une valeur inscrite dans le fichier [nuxt.config.js] :

```
1.  // environnement
2.    env: {
3.      maxAge: 60 * 5
4.  }
```

[env] est un mot clé réservé du fichier de configuration. On fixe ici la durée de vie à 5 minutes. Cette durée est mesurée par rapport à la dernière fois où le navigateur a reçu le cookie de session. Passée cette durée, le cookie ne sera pas renvoyé au serveur qui devra alors démarrer une nouvelle session ;

- lignes 48-50 : si le serveur reçoit le cookie de session, alors l'état du store est initialisé avec l'objet [store] du cookie de session. On se rappelle que cet objet contient l'état sauvegardé du store ;
  - puis ligne 53, le cookie de session sera placé dans la réponse faite au navigateur client :
    - la fonction [get] va chercher le cookie de session dans la **requête reçue** par le serveur ;
    - la fonction [set] met le cookie de session dans la **réponse** que le serveur fait au navigateur client ;

8.5 Incrémentation du compteur du store

L'incrémentation du compteur dans la page [index.vue] évolue de la façon suivante :

```
1.    // gestion des évts
2.    methods: {
3.      incrementCounter() {
4.        console.log('incrementCounter')
5.        // incrément du compteur de 1
6.        this.$store.commit('increment', 1)
7.        // chgt de la valeur affichée
8.        this.value = this.$store.state.counter
9.        // sauvegarde du store dans le cookie de session
10.       this.$cookies.set('session', { store: this.$store.state }, { path: this.
    $nuxt.context.base, maxAge: this.$nuxt.context.env.maxAge })
11.     }
12. }
```

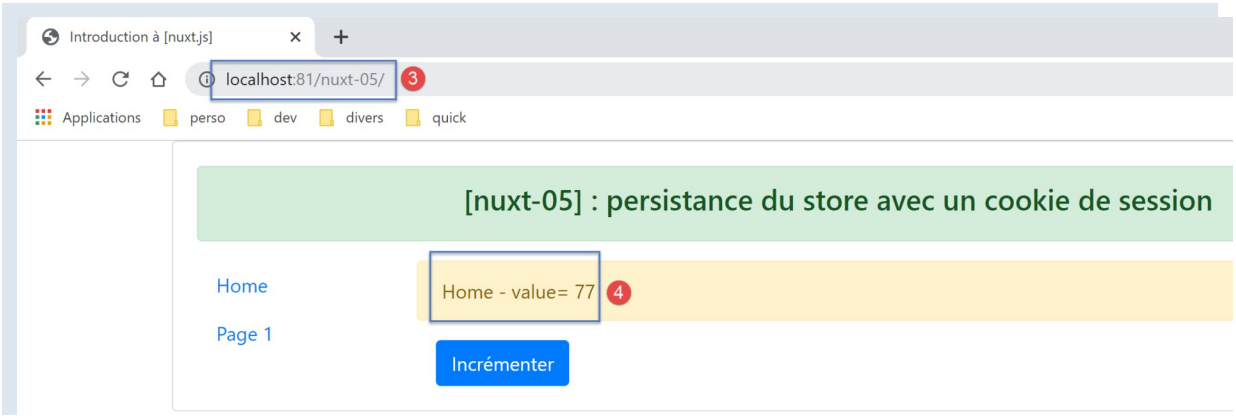
Du côté client, à chaque fois qu'on modifie le store, il faut le sauvegarder dans le cookie de session. En effet, l'utilisateur peut demander une URL à la main à tout moment et on doit être alors capable d'envoyer au serveur un store à jour. C'est pourquoi, ligne 10, après l'incrémentation du compteur du store, on sauvegarde l'état de celui-ci dans le cookie de session :

- les cookies sont disponibles dans la propriété [this.$cookies] ;
- l'état du store [`this.$store.state`] est sauvegardé dans le cookie associé à la clé [store] ;
- le chemin du cookie est [context.base]. Dans une vue, le contexte est disponible dans [this.$nuxt.context] ;
- la durée de vie du cookie est [context.env.maxAge] disponible ici dans la propriété [this.$nuxt.context.env.maxAge] ;

8.6 Exécution de l'exemple [nuxt-05]

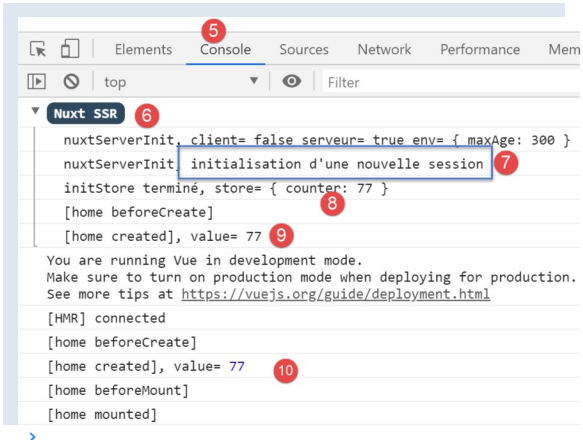
Nous lançons l'application [nuxt-05] :

Les copies d'écran qui suivent sont celles d'un navigateur Chrome. Nous demandons l'URL [http://localhost:81/nuxt-05/]. **N'oubliez pas** le dernier / derrière /nuxt-05, sinon vous n'aurez pas les résultats escomptés :



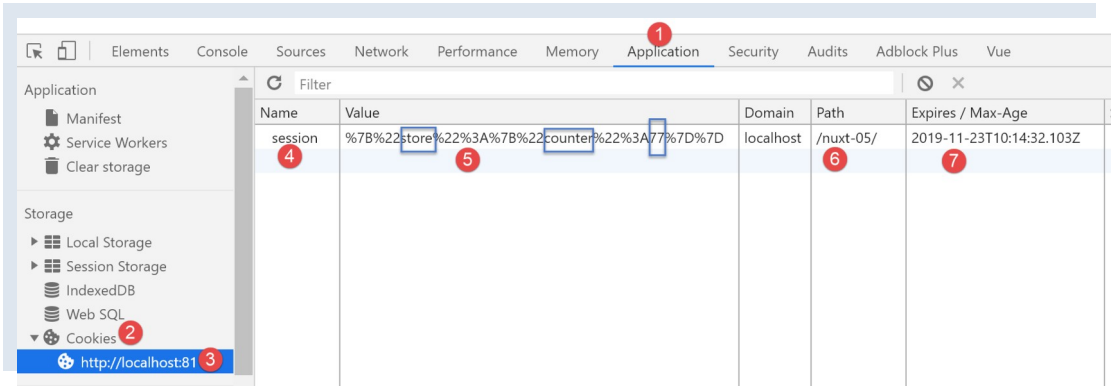
- en [4], nous avons obtenu la valeur initiale du store (77) ;

Examinons les logs du navigateur (F12) :



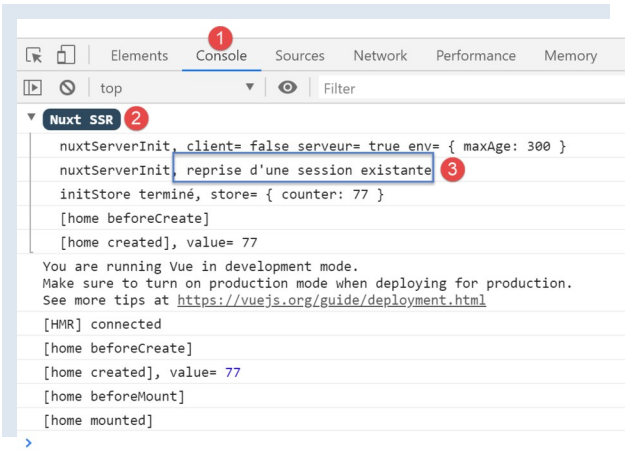
- en [5-6], les logs du serveur ;
- en [7], on voit que le serveur démarre une nouvelle session. Cela veut dire qu'il n'a pas reçu de cookie de session ;
- en [8], initialisation du compteur avec la valeur 77 ;
- en [9], la page [index] du serveur (9) et celle du client (10) affichent bien la même valeur du compteur ;

Maintenant regardons les cookies reçus par le navigateur :



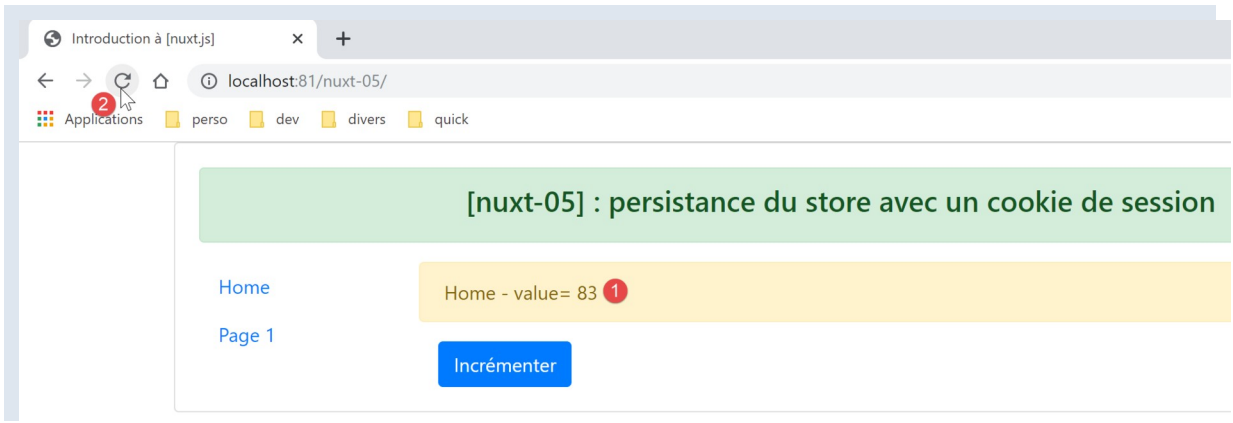
- en [1], choisissez l'onglet [Application] puis l'option [Cookies] [2]. Parmi tous les cookies de votre navigateur, choisissez celui du domaine [http://localhost:81];
- en [4], le cookie nommé 'session'. Si vous ne l'avez pas, rechargez la page [F5] : peut-être avez-vous dépassé sa durée de vie qui est de 5 mn ;
- en [5], la valeur du cookie. Bien que ce ne soit pas très lisible à cause de l'encodage des caractères { :, on distingue la valeur 77 du compteur ;
- en [6], l'URL du cookie : à chaque fois que cette URL sera demandée, le navigateur enverra le cookie au serveur ;
- en [7], l'heure de fin de validité du cookie. Lorsque cette heure sera dépassée, le cookie sera détruit sur le navigateur ;

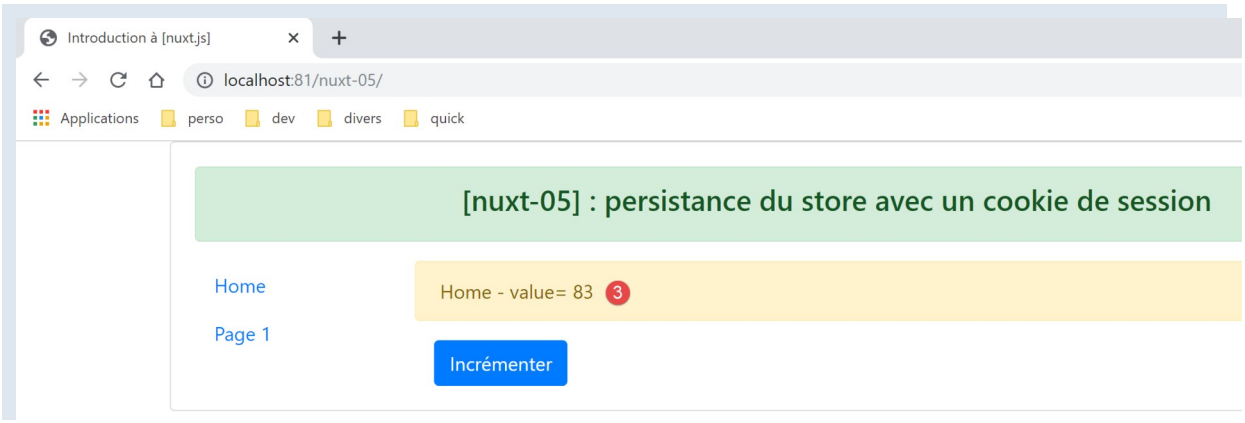
Assurez-vous d'avoir ce cookie. Si vous ne l'avez pas, rechargez la page (F5). Lorsque vous avez la page avec son cookie, rechargez de nouveau la page (F5). Les logs deviennent alors les suivants :



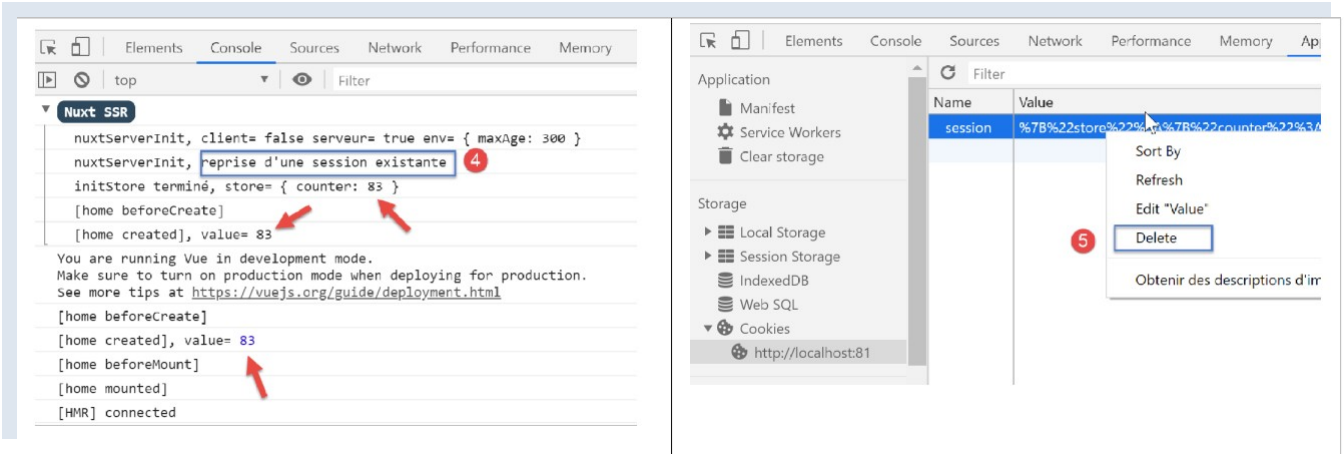
Cette fois-ci en [3], le serveur a bien récupéré le cookie de session. C'est le navigateur client qui le lui a envoyé.

Maintenant, procédez à des incrémentations du compteur, puis de temps en temps rechargez la page courante (F5), que ce soit [index] ou [page1], vous devez constater que le compteur ne revient pas à 77 comme dans l'exemple [nuxt-04] mais garde la valeur qu'il avait sur le navigateur client avant le rechargement de la page :





Les logs du navigateur sont alors les suivants :



Note : pour les tests vous pouvez avoir besoin de supprimer [5] le cookie de session stocké sur le navigateur pour repartir avec une nouvelle session, initialisée par le serveur, lors de la prochaine requête vers celui-ci.

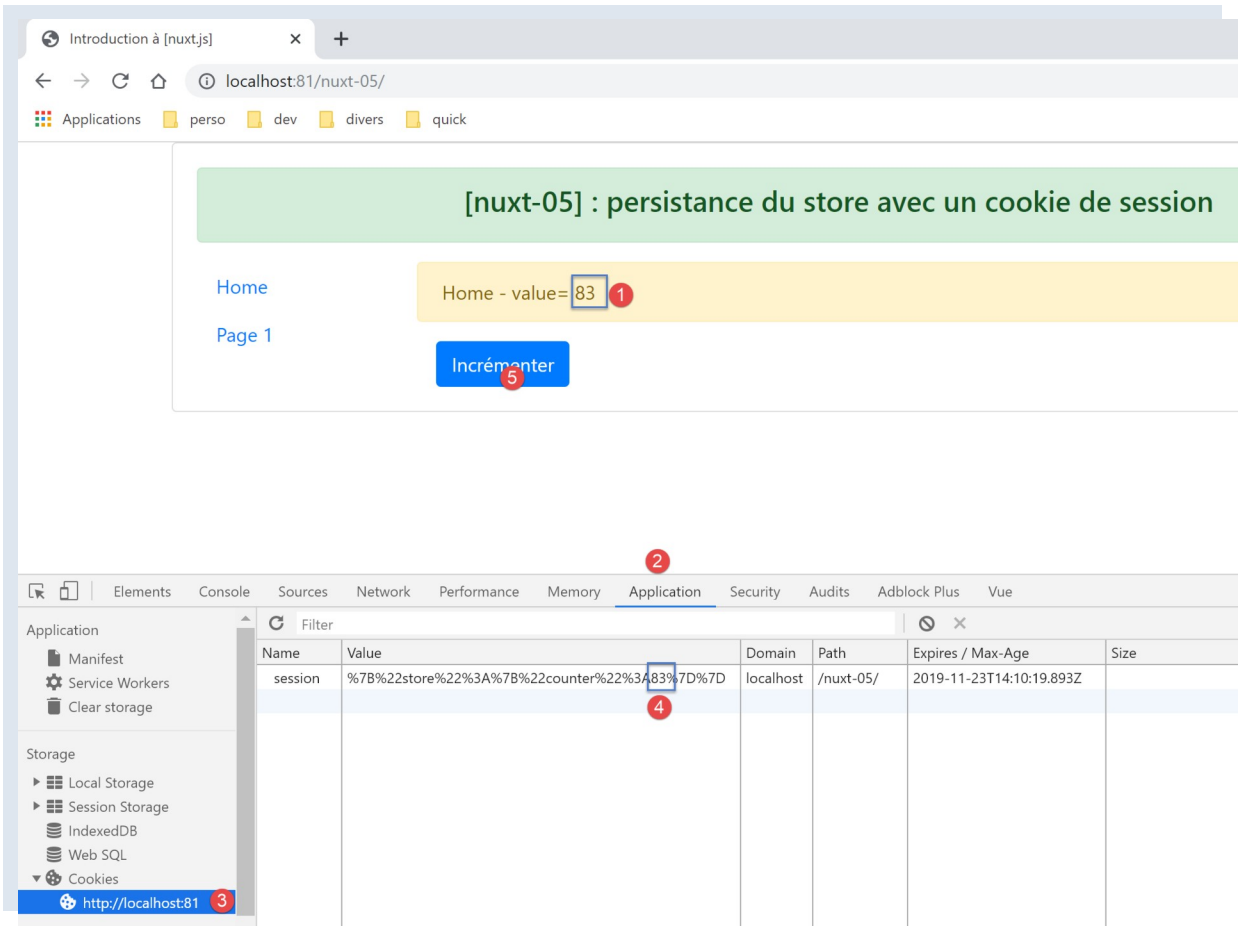
Enfin montrons l'influence de la fonction [incrementCounter] de la page [index] sur le cookie de session stocké sur le navigateur client :

```
1.  // gestion des évts
2.    methods: {
3.      incrementCounter() {
4.        console.log('incrementCounter')
```

```
5.        // incrément du compteur de 1
6.        this.$store.commit('increment', 1)
7.        // chgt de la valeur affichée
8.        this.value = this.$store.state.counter
9.        // sauvegarde du store dans le cookie de session
10.       this.$cookies.set('session', { store: this.$store.state }, { path: this.
    $nuxt.context.base, maxAge: this.$nuxt.context.env.maxAge })
11.     }
12.   }
```

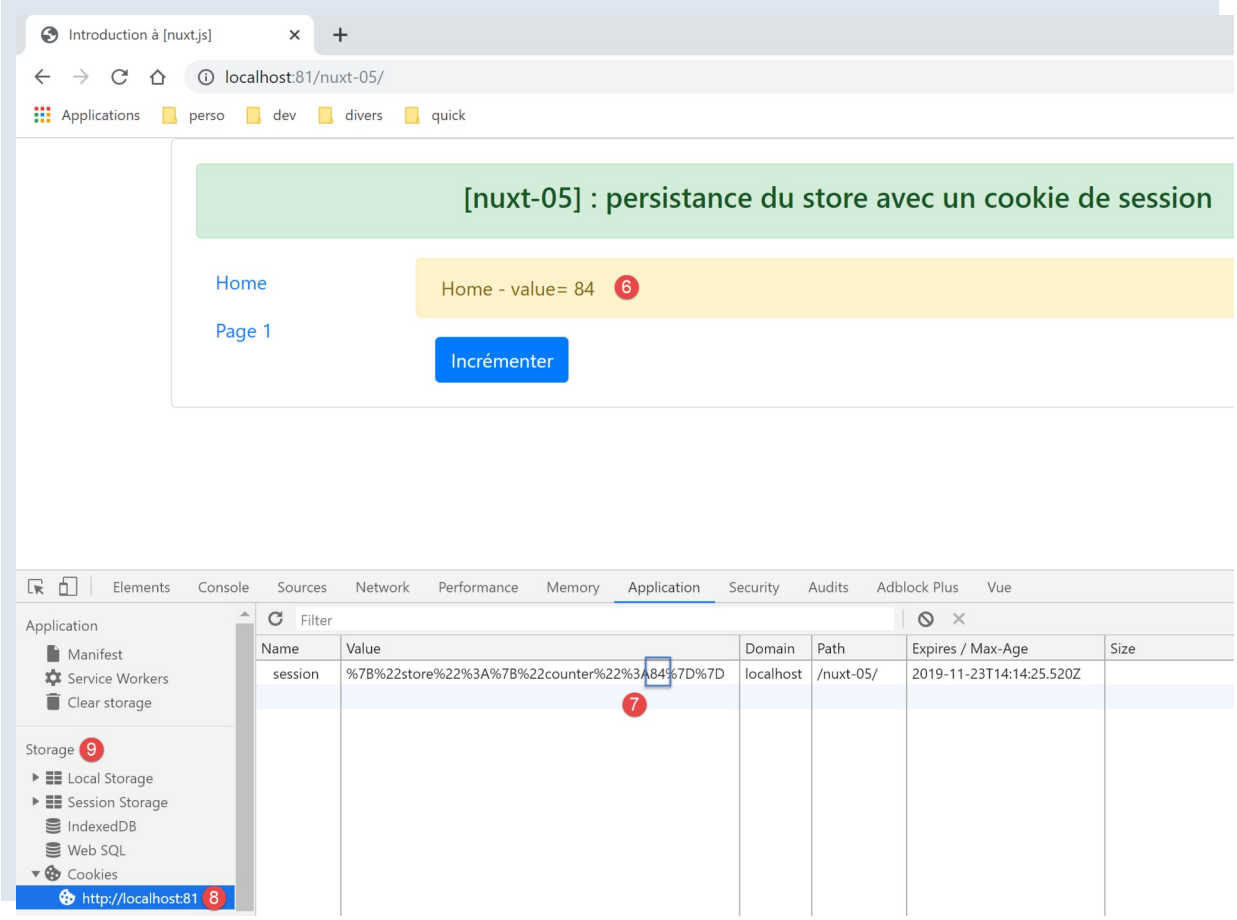
- ligne 10 : la modification du compteur est répercutée sur le cookie de session ;

Vérifions ce point. On part de la situation suivante :



- en [4], le compteur du cookie de session reflète bien la valeur affichée [1] ;

Maintenant, incrémentons le compteur une fois [5]. Le cookie de session en [4] évolue de la façon suivante :



- en [7], le compteur du cookie de session est bien passé à 84. Pour le voir, il faut rafraîchir la vue [8]. Pour cela sélectionnez une autre option du [Storage] [9], puis resélectionnez l'option [8]. La nouvelle valeur du cookie de session devrait alors apparaître ;

9 Exemple [nuxt-06] : injection dans le contexte d'un gestionnaire de session

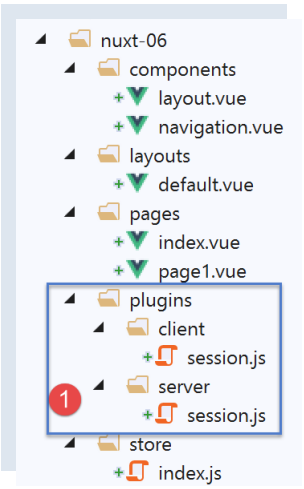
9.1 Présentation

L'exemple [nuxt-05] a montré qu'on pouvait persister le store même lorsque l'utilisateur force des appels au serveur. Les éléments du store sont réactifs pour que s'ils sont intégrés à des vues celles-ci soient réactives aux changements du store. On peut vouloir aussi persister des éléments au fil des échanges client / serveur sans pour autant vouloir qu'ils soient réactifs, tout simplement parce qu'ils ne sont pas affichés par des vues. On peut alors stocker ceux-ci dans la session sans qu'ils soient pour autant dans le store.

Le store est accessible facilement au travers de propriétés telles que [context.app.$store] en-dehors des vues ou [this.$store] dans les vues. On voudrait quelque chose d'analogue pour la session, quelque chose comme [context.app.$session] ou [this.$session]. On va voir que c'est possible grâce au concept d'injection. Seulement on ne peut pas injecter des objets dans le contexte, seulement des fonctions. Celle-ci sera alors disponible au travers des expressions [context.app.$session()] ou [this.$session()].

Enfin, nous allons présenter le concept [nuxt] de [plugin].

L'exemple [nuxt-06] est obtenu initialement par recopie du projet [nuxt-05] :



- en [1], nous ajouterons un dossier [plugins] ;

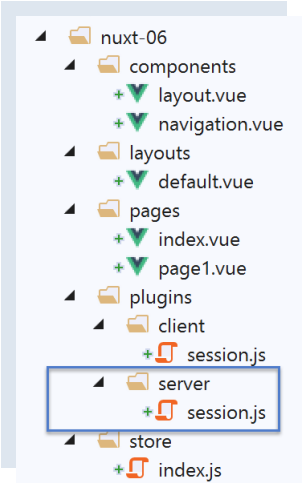
9.2 la notion de plugin [nuxt]

[nuxt] nomme [plugin] tout code exécuté au démarrage de l'application, avant même l'exécution de la fonction [nuxtServerInit] par le serveur qui était jusqu'à maintenant la première fonction utilisateur à être exécutée. Les plugins de l'application doivent être déclarés dans la clé [plugins] du fichier de configuration [nuxt.config.js] :

```
1.    /*
2.     ** Plugins to load before mounting the App
3.     */
4.    plugins: [
5.      { src: '~/plugins/client/session', mode: 'client' },
6.      { src: '~/plugins/server/session', mode: 'server' }
7.  ],
```

- lignes 5-6 : un plugin est désigné par son chemin [src] et son mode d'exécution [mode]. [mode] peut avoir trois valeurs :
  - [client] : le plugin doit être exécuté côté client uniquement ;
  - [server] : le plugin doit être exécuté côté serveur uniquement ;
  - absence de la clé [mode] : dans ce cas, le plugin doit être exécuté à la fois côté client et serveur ;

- lignes 5-6 : nous avons placé nos deux plugins dans un dossier [plugins]. Il n'y a aucune obligation à cela. Les plugins peuvent être placés n'importe où dans l'arborescence du projet. De même les noms des sous-dossiers [client, server] sont ici arbitraires ;



9.3 Le plugin [session] du serveur

Le plugin [server / session.js] est le suivant :

```
1.  /* eslint-disable no-console */
2.  export default (context, inject) => {
3.    // gestion de la session serveur
4.
5.    // y-a-t-il une session existante ?
6.    let value = context.app.$cookies.get('session')
7.    if (!value) {
8.      // nouvelle session
9.      console.log("[plugin session server], démarrage d'une nouvelle session")
10.     value = initValue
11.   } else {
12.     // session existante
13.     console.log("[plugin session server], reprise d'une session existante")
14.   }
15.   // définition de la session
16.   const session = {
17.     // contenu de la session
18.     value,
19.     // sauvegarde de la session dans un cookie
20.     save(context) {
21.       context.app.$cookies.set('session', this.value, { path: context.base, maxAge:
    context.env.maxAge })
22.     }
23.   }
24.   // on injecte une fonction dans [context, Vue] qui rendra la session courante
25.   inject('session', () => session)
26. }
27.
28. // valeur initiale de la session
29. const initValue = {
30.   initSessionDone: false
31. }
```

- ligne 2 : les plugins sont exécutés à chaque fois qu'il y a un appel au serveur : au démarrage et à chaque fois que l'utilisateur force un appel au serveur en tapant une URL à la main :
  - c'est d'abord le (ou les) plugin(s) du serveur qui est (sont) exécuté(s) en premier ;
  - lorsque le navigateur client a reçu la réponse du serveur, c'est au tour du (ou des) plugin(s) du client de s'exécuter ;
- ligne 2 : tout plugin, client ou serveur, reçoit deux paramètres :
  - [context] : le contexte du serveur ou du client selon qui exécute le plugin ;

    - [inject] : une fonction qui permet d'injecter une fonction dans le contexte du serveur ou du client ;
- le but du plugin [server / session] est double :
    - définir une session (lignes 16-23) ;
    - définir au sein du contexte, une fonction [$session] qui rendra comme résultat, la session de la ligne 16. C'est la ligne 25 qui fait cela ;
- lignes 16-23 : la session encapsulera ses données dans l'objet [value] de la ligne 18 ;
- lignes 20-22 : elle dispose d'une fonction [save] qui reçoit en paramètre un objet [context]. C'est le code appelant qui lui fournit ce contexte. Avec celui-ci, la fonction [save] sauvegarde la valeur de la session, l'objet [value], dans le cookie de session ;
- ligne 6 : lorsque le plugin [server / session] s'exécute, il commence par regarder si le serveur a reçu un cookie de session ;
    - si oui, l'objet [value] de la ligne 6 représente la valeur de la session, l'ensemble des données encapsulées dans celle-ci ;
    - si non, lignes 7-11, on fixe la valeur initiale de la session. Celle-ci sera l'objet [initValue] des lignes 29-31. Les éléments de la session seront définis dans la fonction [nuxtServerInit] qui est exécutée après le plugin serveur ;
- ligne 18 : la notation [value] est un raccourci pour la notation [value:value]. Le [value] de gauche est le nom d'une clé d'objet, le [value] de droite est l'objet [value] déclaré ligne 6 ;
- ligne 25 : lorsqu'on arrive à cette ligne, la session a été soit créée parce qu'elle n'existait pas, soit récupérée dans la requête HTTP du navigateur client ;
- ligne 25 : on injecte dans le contexte serveur une nouvelle fonction :
    - le 1er paramètre de [inject] est le nom de la fonction qu'on crée, ici 'session'. [nuxt] lui donnera en fait le nom '**$**session' ;
    - le 2ième paramètre est la définition de la fonction. Ici la fonction [$session]
        - n'admettra aucun paramètre ;
        - rendra l'objet [session] de la ligne 16 ;
- une fois le plugin exécuté :
    - la fonction [$session] est disponible dans [**context.app.$session**] là où l'objet [context] est disponible, ou [**this.$session**] dans une vue ou dans le store [vuex] ;
    - la fonction [$session] rend un objet [session] avec une unique clé [value] ;
    - à la création initiale de la session, l'objet [value] n'a qu'une clé [initStoreDone] (lignes 29-31). La clé [initStoreDone:false] sert à indiquer que le store n'a pas encore été placé dans la session. Cela va être fait par la fonction [nuxtServerInit] ;

9.4 Initialisation de la session

Une fois le plugin [session / server] exécuté par le serveur, celui-ci va exécuter le script [store / index.js] suivant :

```
1.  /* eslint-disable no-console */
2.  export const state = () => ({
3.    // compteur
4.    counter: 0
5.  })
6.
7.  export const mutations = {
8.    // incrémentation du compteur d'une valeur [inc]
9.    increment(state, inc) {
10.     state.counter += inc
11.   },
12.   // remplacement du state
13.   replace(state, newState) {
14.     for (const attr in newState) {
15.       state[attr] = newState[attr]
16.     }
17.   }
18. }
19.
20. export const actions = {
21.   async nuxtServerInit(store, context) {
22.     // qui exécute ce code ?
23.     console.log('nuxtServerInit, client=', process.client, 'serveur=', process.server, 'env=',
    context.env)
24.     // on attend la fin d'une promesse
25.     await new Promise(function(resolve, reject) {
26.       // on a normalement ici une fonction asynchrone
27.       // on la simule avec une attente d'une seconde
28.       setTimeout(() => {
```

```
29.         // init session
30.         initSession(store, context)
31.         // succès
32.         resolve()
33.       }, 1000)
34.     })
35.   }
36. }
37.
38. function initSession(store, context) {
39.   // store est le store à initialiser
40.
41.   // on récupère la session
42.   const session = context.app.$session()
43.   // la session a-t-elle été déjà initialisée ?
44.   if (!session.value.initSessionDone) {
45.     // on démarre un nouveau store
46.     console.log("nuxtServerInit, initialisation d'une nouvelle session")
47.     // on initialise le store
48.     store.commit('increment', 77)
49.     // on met le store dans la session
50.     session.value.store = store.state
51.     // on initialise une nouvelle session
52.     session.value.somethingImportant = { x: 2, y: 4 }
53.     // la session est désormais initialisée
54.     session.value.initSessionDone = true
55.   } else {
56.     console.log("nuxtServerInit, reprise d'un store existant")
57.     // on met à jour le store avec le store de la session
58.     store.commit('replace', session.value.store)
59.   }
60.   // on sauvegarde la session
61.   session.save(context)
62.   // log
63.   console.log('initSession terminé, store=', store.state, 'session=', session.value)
64. }
```

Par rapport au store du projet [nuxt-05], seule la fonction [initSession] (anciennement *initStore*) des lignes 38-60 change :

- ligne 42 : on récupère la session grâce à la fonction [$session] qui a été injectée dans le contexte du serveur ;
- ligne 44 : on regarde si la session a déjà été initialisée ;
- lignes 45-54 : si ce n'est pas le cas :
  - ligne 48 : le store est initialisé ;
  - ligne 50 : l'état du store est mis dans la session ;
  - ligne 52 : on ajoute un autre objet [somethingImportant] dans la session. Celui-ci ne fera pas partie du store ;
  - ligne 54 : on note le fait que la session est désormais initialisée ;
- lignes 55-59 : si la session était déjà initialisée :
  - ligne 58 : le nouveau store est initialisé avec le contenu de la session ;
- ligne 61 : la session est sauvegardée dans le cookie de session. On rappelle que cela consiste à placer le cookie dans la réponse HTTP que le serveur va faire au navigateur client ;

9.5 Le plugin [client / session] du client

Une fois que le serveur a exécuté les scripts [plugins / server / session] et [store / index], il va envoyer l'une des pages [index, page1] au navigateur client. Dans la réponse HTTP du serveur, il y aura le cookie de session. Une fois la page reçue par le navigateur client, les scripts client embarqués dans la page vont s'exécuter. Le plugin [client / session] va alors s'exécuter :

```
1.  /* eslint-disable no-console */
2.  export default (context, inject) => {
3.    // gestion de la session client
4.
5.    // la session existe forcément, initialisée par le serveur
6.    console.log('[plugin session client], reprise de la session du serveur')
7.
8.    // définition de la session
9.    const session = {
```

```
10.     // contenu de la session
11.     value: context.app.$cookies.get('session'),
12.     // sauvegarde de la session dans un cookie
13.     save(context) {
14.       context.app.$cookies.set('session', this.value, { path: context.base, maxAge:
    context.env.maxAge })
15.     }
16.   }
17.
18.   // on injecte une fonction dans [context, Vue] qui rendra la session courante
19.   inject('session', () => session)
20. }
```

- lorsque le plugin client s'exécute, le cookie de session a déjà été reçu par le navigateur client ;
- l'objectif du plugin [client] est d'injecter lui-aussi une fonction [$session] dans le contexte du client. Cette fonction rendrait la session envoyée par le serveur ;
- ligne 19 : la fonction injectée [$session] rendra la session des lignes 9-16 ;
- lignes 9-16 : l'objet [session] géré par le client. Ce sera une copie de la session envoyée par le serveur ;
- ligne 11 : la valeur de la session client est prise dans le cookie de session envoyé par le serveur [nuxt] ;
- lignes 13-15 : comme pour la session du serveur, la session du client a une fonction [save] qui permet de sauvegarder la valeur de la session, [this.value] ligne 14, dans le cookie de session stocké sur le navigateur ;

9.6 La page [index]

La page [index] évolue de la façon suivante :

```
1.  <!-- page [index] -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <template slot="right">
8.        <b-alert show variant="warning"> Home - session= {{ jsonSession }}, counter= {{
    $store.state.counter }} </b-alert>
9.        <!-- bouton -->
10.       <b-button @click="incrementCounter" class="ml-3" variant="primary">Incrémenter</b-button>
11.     </template>
12.   </Layout>
13. </template>
14.
15. <script>
16. /* eslint-disable no-undef */
17. /* eslint-disable no-console */
18. /* eslint-disable nuxt/no-env-in-hooks */
19.
20. import Layout from '@/components/layout'
21. import Navigation from '@/components/navigation'
22. export default {
23.   name: 'Home',
24.   // composants utilisés
25.   components: {
26.     Layout,
27.     Navigation
28.   },
29.   computed: {
30.     jsonSession() {
31.       return JSON.stringify(this.$session().value)
32.     }
33.   },
34.   // cycle de vie
35.   beforeCreate() {
36.     // client et serveur
37.     console.log('[home beforeCreate]')
38.   },
39.   created() {
40.     // client et serveur
41.     console.log('[home created], session=', this.$session().value)
42.   },
43.   beforeMount() {
44.     // client seulement
```

```
45.     console.log('[home beforeMount]')
46.   },
47.   mounted() {
48.     // client seulement
49.     console.log('[home mounted]')
50.   },
51.   // gestion des évts
52.   methods: {
53.     incrementCounter() {
54.       console.log('incrementCounter')
55.       // incrément du compteur de 1
56.       this.$store.commit('increment', 1)
57.       // modification session
58.       const session = this.$session()
59.       session.value.store = this.$store.state
60.       session.value.somethingImportant.x++
61.       session.value.somethingImportant.y++
62.       // sauvegarde de la session dans le cookie de session
63.       session.save(this.$nuxt.context)
64.     }
65.   }
66. }
67. </script>
```

Il faut se rappeler que cette page est exécutée aussi bien côté serveur que côté client.

- ligne 8 : on affiche maintenant et la session et le store ;
- ligne 30 : [jsonSession] est une propriété calculée qui rend la chaîne jSON de la valeur de la session ;
- ligne 41 : on affiche la valeur de la session à l'aide de la fonction injectée [this.$session]. Celle-ci existe aussi bien dans le contexte du serveur que dans celui du client ;
- ligne 53 : la méthode [incrementCounter] n'est elle exécutée que côté client ;
- ligne 56 : le compteur du store est incrémenté et affiché comme auparavant ;
- ligne 58 : on récupère la session grâce à la fonction injectée [this.$session] ;
- ligne 59 : le store de la session est mis à jour ;
- lignes 60-61 : on incrémente les attributs [somethingImportant.x, somethingImportant.y] de la session. Cela juste pour montrer qu'une session peut servir à transporter autre chose que le store ;
- ligne 63 : la session est sauvegardée dans le cookie de session stocké sur le navigateur. Dans une vue client, le contexte de celui-ci est disponible dans [this.$nuxt.context] ;

Le but de la page [index] est de montrer que la session n'est pas réactive alors que le store l'est. Lorsqu'on va incrémenter les éléments de la session, on découvrira que la vue n'est pas mise à jour. La vue [page1] présente une solution à ce problème.

9.7 La page [page1]

La page [page1] est obtenue par recopie de la page [index] puis quelque peu modifiée :

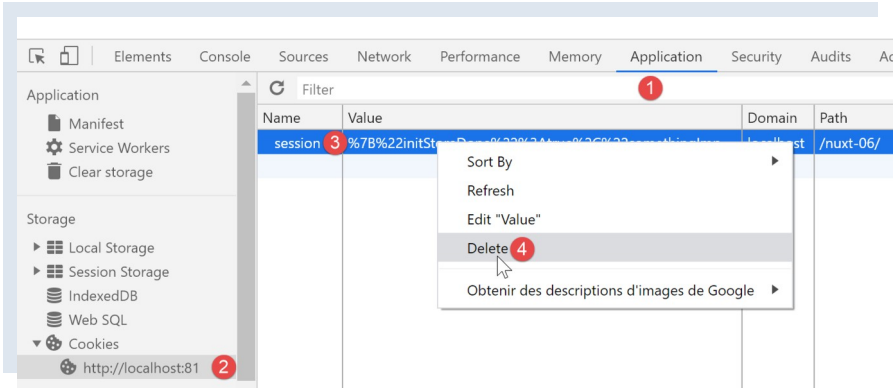
```
1.  <!-- page [index] -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <template slot="right">
8.        <b-alert show variant="warning"> Page1 - session= {{ jsonSession }}, counter= {{
    $store.state.counter }} </b-alert>
9.        <!-- bouton -->
10.       <b-button @click="incrementCounter" class="ml-3" variant="primary">Incrémenter</b-
    button>
11.     </template>
12.   </Layout>
13. </template>
14.
15. <script>
16. /* eslint-disable no-undef */
17. /* eslint-disable no-console */
18. /* eslint-disable nuxt/no-env-in-hooks */
19.
20. import Layout from '@/components/layout'
```

```
21. import Navigation from '@/components/navigation'
22. export default {
23.   name: 'Page1',
24.   // composants utilisés
25.   components: {
26.     Layout,
27.     Navigation
28.   },
29.   data() {
30.     return {
31.       session: {}
32.     }
33.   },
34.   computed: {
35.     jsonSession() {
36.       return JSON.stringify(this.session.value)
37.     }
38.   },
39.   // cycle de vie
40.   beforeCreate() {
41.     // client et serveur
42.     console.log('[page1 beforeCreate]')
43.   },
44.   created() {
45.     // client et serveur
46.     // on met la session dans les propriétés réactives de la page
47.     this.session = this.$session()
48.     // log
49.     console.log('[page1 created], session=', this.session.value)
50.   },
51.   beforeMount() {
52.     // client seulement
53.     console.log('[page1 beforeMount]')
54.   },
55.   mounted() {
56.     // client seulement
57.     console.log('[page1 mounted]')
58.   },
59.   // gestion des évts
60.   methods: {
61.     incrementCounter() {
62.       console.log('incrementCounter')
63.       // incrément du compteur de 1
64.       this.$store.commit('increment', 1)
65.       // modification session
66.       this.session.value.store = this.$store.state
67.       this.session.value.somethingImportant.x++
68.       this.session.value.somethingImportant.y++
69.       // sauvegarde de la session dans le cookie de session
70.       this.session.save(this.$nuxt.context)
71.     }
72.   }
73. }
74. </script>
```

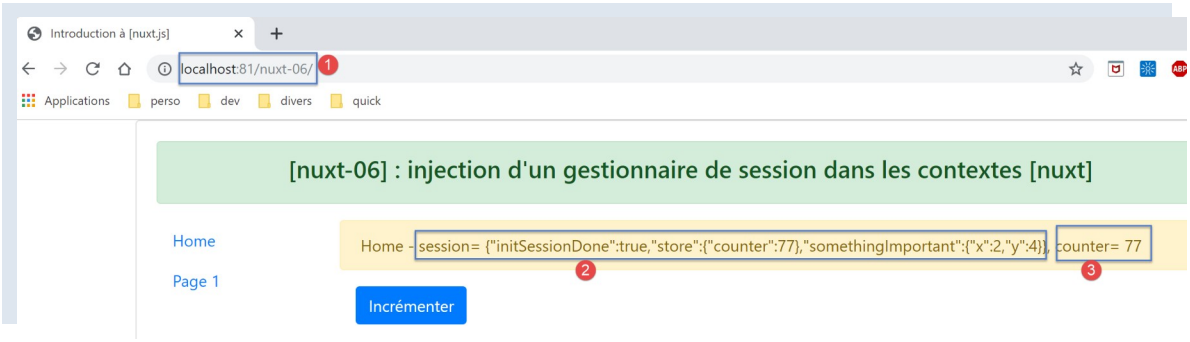
- ligne 47 : la principale différence vient du fait qu'on met la session courante dans les propriétés de la page (lignes 29-33). Cela va avoir pour effet que désormais la session va devenir réactive. Lorsque la fonction [incrementCounter] va incrémenter les éléments de la session, la vue [page1] sera mise à jour ;

9.8 Exécution du projet

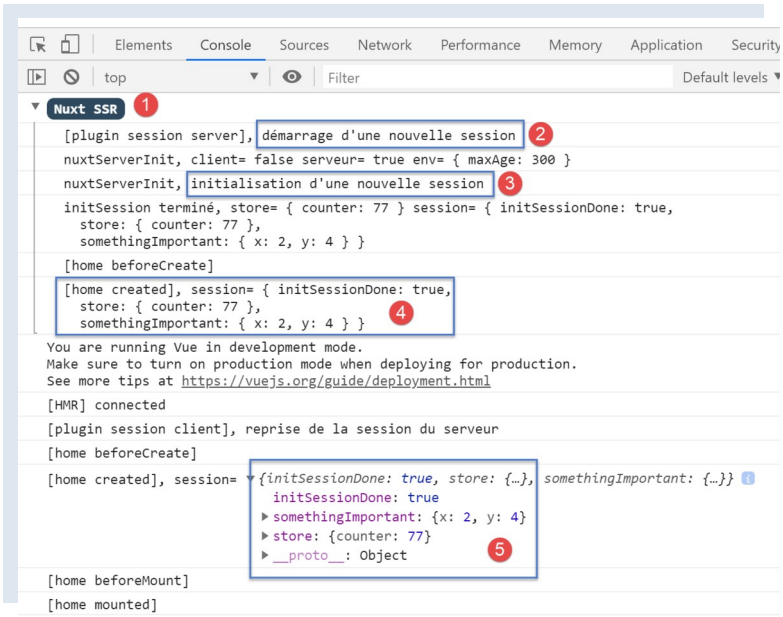
Avant d'exécuter le projet, vérifiez le cookie de session de votre navigateur et s'il existe supprimez-le pour que le serveur crée une session neuve :



Maintenant demandons l'URL [http://localhost:81/nuxt-06/] :

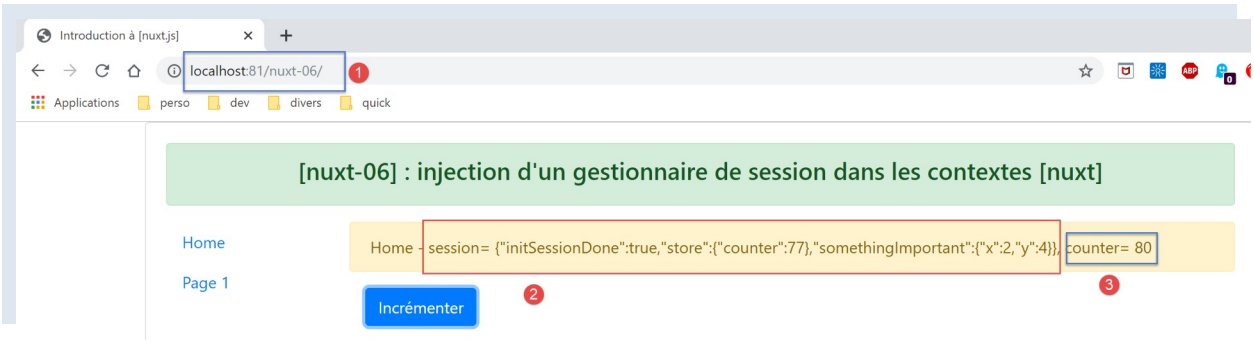


Les logs dans le navigateur sont alors les suivants :



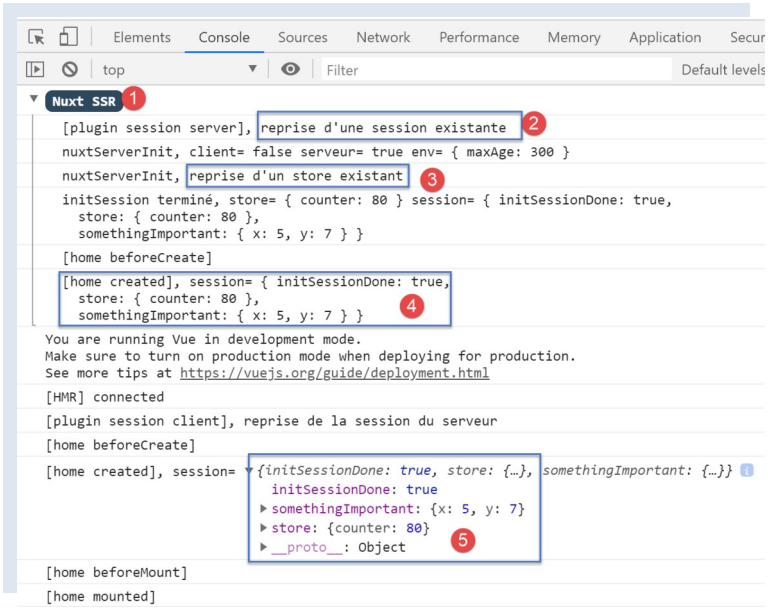
- en [2], le serveur démarre une nouvelle session dans le plugin [session] du serveur ;
- en [3], cette nouvelle session est initialisée dans [nuxtServerInit] ;
- en [4], la nouvelle session telle qu'elle est connue sur le serveur ;
- en [5], le client a correctement récupéré cette session ;

Maintenant incrémentons le compteur trois fois :



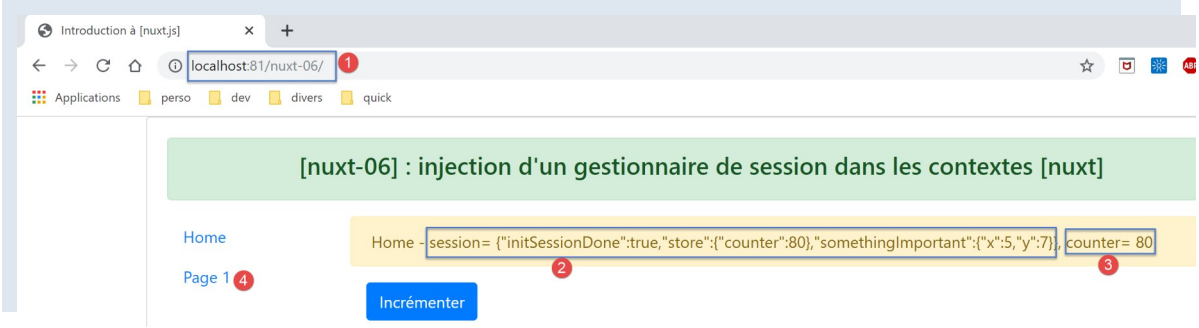
- en [3], le compteur a bien été incrémenté mais pas la session en [2]. Alors que [3] affiche le store qui est réactif, [2] affiche la session qui elle n'est pas réactive :

Maintenant rechargeons la page (F5). Les logs sont les suivants suite à ce rechargement :



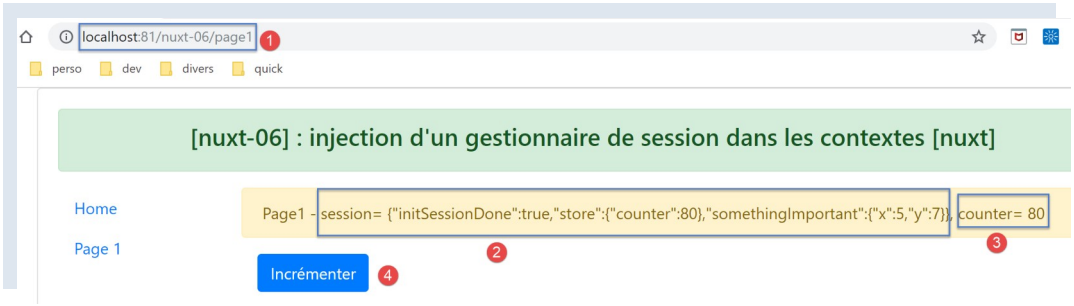
- en [2], on voit que le serveur a reçu un cookie de session envoyé par le navigateur client ;
- en [4], on voit que le store n'est pas réinitialisé mais repris dans la session reçue ;
- en [4-5] : on voit que les attributs de la session ont bien tous été incrémentés trois fois ;

La page envoyée par le serveur est alors la suivante ;

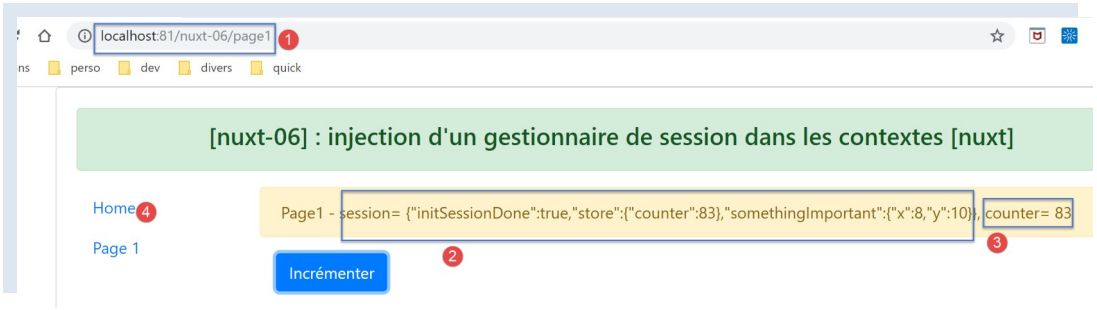


La conclusion issue de cette page est que la session peut transporter d'autres éléments que le store mais ceux-ci ne sont pas réactifs.

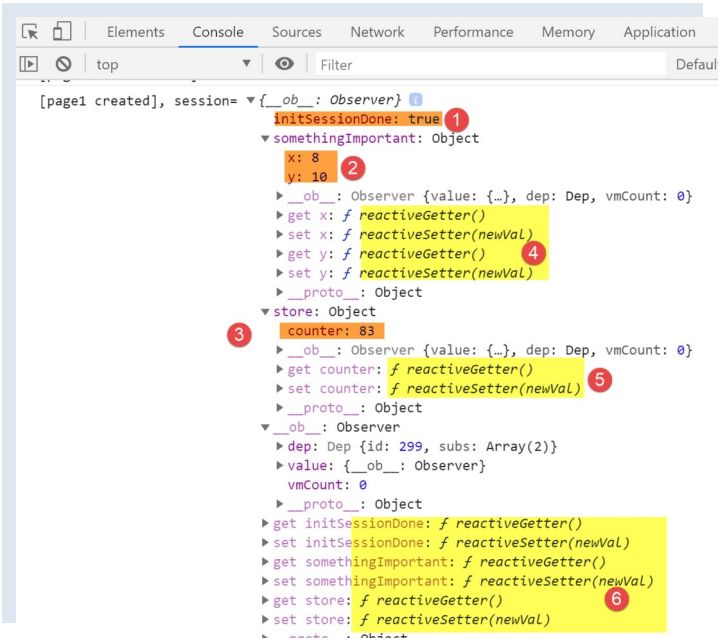
Maintenant cliquons sur le lien [Page 1] [4]. La nouvelle page affichée est alors la suivante :



Puis utilisons le bouton [Incrémenter] trois fois. La page devient la suivante :

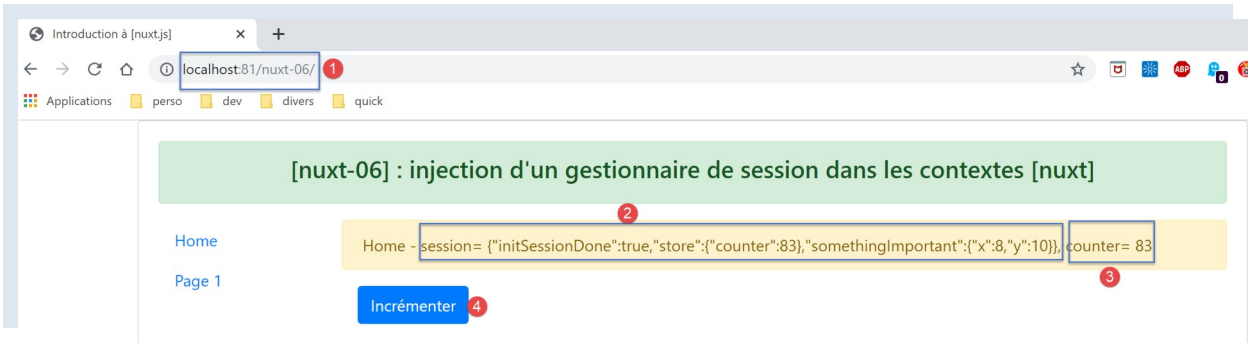


Cette fois-ci la session s'affiche correctement en [2]. Elle est ici réactive. Cela se voit dans les logs :

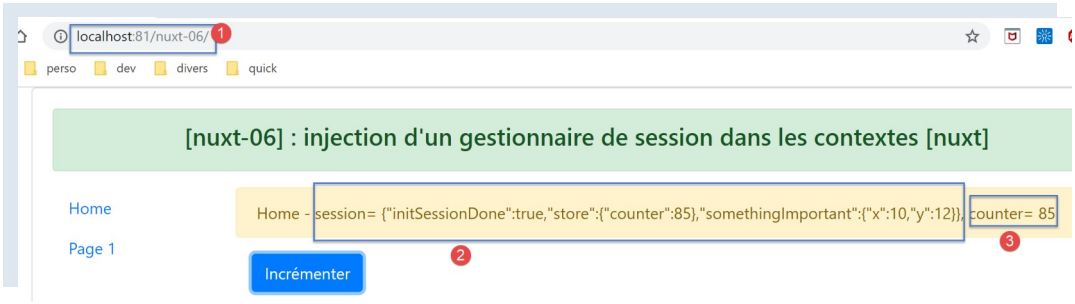


- en [1-3], les valeurs de la session ;
- en [4-6], les getters et setters réactifs des éléments de la session ;

Maintenant cliquons sur le lien [Home] [4]. Nous obtenons la page suivante :

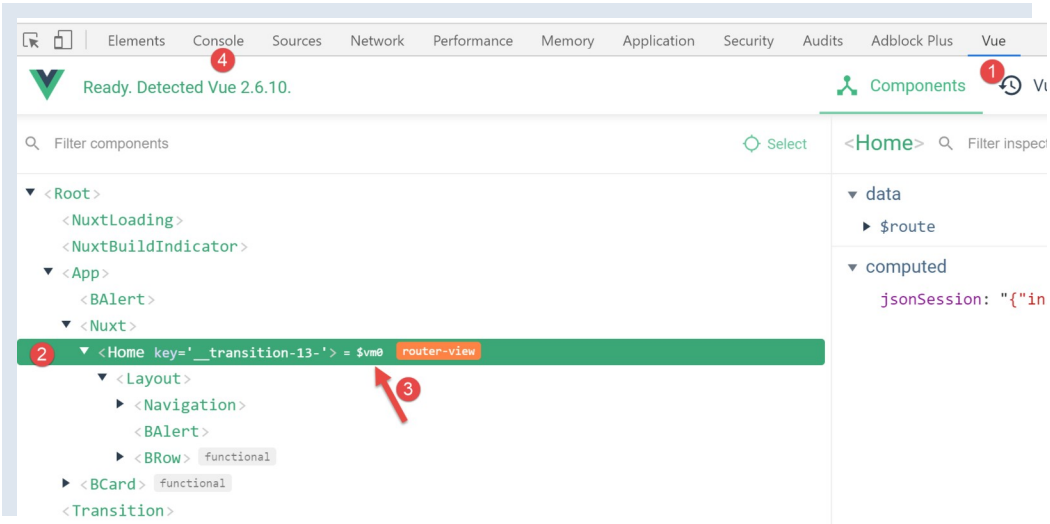


Puis cliquons deux fois sur le bouton [Incrémenter] [4]. La page devient la suivante :



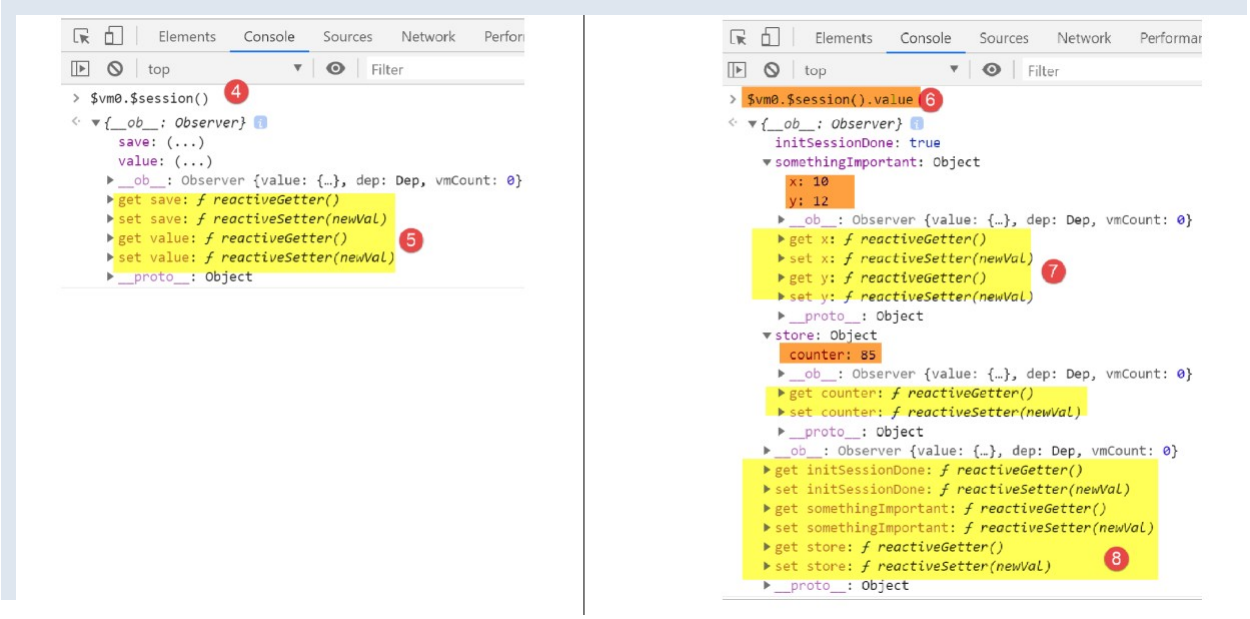
On constate que là également la session est devenue réactive [2].

Demandons la valeur rendue par la fonction [this.$session()] :



- dans l'onglet [Vue], on sélectionne la page courante [Home] pour en obtenir la référence [$vm0] [3] ;

Puis dans l'onglet [Console] [4], demandons la valeur de la fonction [$vm0.$session()] :



- en [5], on voit que la session est devenue réactive alors qu'elle ne l'était pas initialement ;
- en [6], on demande à voir la valeur de la session ;
- en [7-8], on découvre que cette valeur est elle également devenue réactive ;

On a donc là un résultat inattendu : si un élément devient réactif dans une page parce qu'il a été placé dans les propriétés de la page, alors il devient également réactif dans les pages où il ne fait pas partie des propriétés.

9.9 Conclusion

L'exemple [nuxt-05] a montré qu'on pouvait persister le store au fil des requêtes faites au serveur. L'exemple [nuxt-06] fait la même chose avec un objet qu'on a appelé [session] par analogie avec la session web. On a vu que cette session pouvait avoir les mêmes propriétés que le store [Vuex] et devenir réactive elle aussi alors que nativement elle ne l'était pas.

Alors quel est l'intérêt du store [Vuex] ? Je dois avouer que pour l'instant il ne m'est pas apparu. Il est probable que quelque chose m'a échappé. Donc dans le doute, je conseillerai d'utiliser :

- un **store [Vuex]** pour y mettre tout ce qui doit être partagé entre les pages du client, et ce qui éventuellement doit être partagé entre le client et le serveur ;
- un **cookie de session** si le store doit être persisté lors d'un appel du client vers le serveur, la session ne contenant alors que le store ;

Les exemples [nuxt-05] et [nuxt-06] avaient pour but de montrer comment on pouvait assurer la continuité de l'application lorsque l'utilisateur force l'appel au serveur en tapant manuellement des URL. On rappelle que le comportement par défaut dans ce cas est un redémarrage de l'application, son état du moment étant alors perdu.

10 Exemple [nuxt-07] : les contextes client et serveur

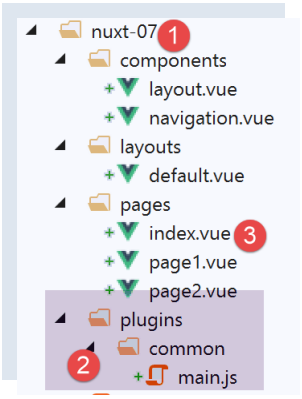
10.1 Présentation

L'exemple [nuxt-07] vise à explorer l'objet [context] du côté serveur et du côté client. Il ne faut pas oublier que ces deux acteurs des applications [nuxt] sont séparés : **ils ne partagent** rien sauf :

- ce que le serveur veut bien envoyer au client (dans la réponse HTTP et la page envoyée) ;
- ce que le client veut bien envoyer au serveur (dans sa requête HTTP) ;

Si donc, comme on va le voir, les objets manipulés par le serveur et ceux manipulés par le client **portent le même nom**, ils ne sont pas identiques : ils peuvent parfois être des copies l'un de l'autre mais n'ont jamais la même référence. Modifier un objet client n'a aucun effet sur l'objet de même nom, côté serveur, et vice-versa.

L'exemple [nuxt-07] est obtenu initialement par recopie de l'exemple [nuxt-01] :



- en [2], nous allons ajouter un plugin commun au client et au serveur ;
- en [3], nous allons modifier quelque peu la page [index] ;

10.2 Le plugin [common / main.js]

Le plugin [common / main.js] est exécuté à la fois par le client et le serveur : cela est dû à la configuration [nuxt.config.js] suivante :

```
1.  /*
2.     ** Plugins to load before mounting the App
3.     */
4.  plugins: [{ src: '~/plugins/common/main.js' }],
```

- ligne 4 : l'absence de la propriété [mode] fait que le plugin [`~/plugins/common/main.js`] sera exécuté à la fois par le client et le serveur : le serveur d'abord, le client ensuite ;

Ce plugin sera le suivant :

```
1.  /* eslint-disable no-undef */
2.  /* eslint-disable no-console */
3.  export default function(...args) {
4.    // qui exécute ce code ?
5.    console.log('[main server], process.server=', process.server, 'process.client=',
    process.client)
6.    const who = process.server ? 'server' : 'client'
7.    const main = '[main ' + who + ']'
8.    // nbre d'arguments
9.    console.log(main + ', il y a', args.length, 'arguments')
10.
11.   // 1er argument
12.   const context = args[0]
13.   // clés du contexte
14.   dumpkeys(main + ', context', context)
```

```
15.   // l'application
16.   dumpkeys(main + ', context.app', context.app)
17.   // la route
18.   dumpkeys(main + ', context.route', context.route)
19.   console.log(main + ', context.route=', context.route)
20.   // le router
21.   dumpkeys(main + ', context.app.router', context.app.router)
22.   // le router.options.routes
23.   dumpkeys(main + ', context.app.router.options.routes', context.app.router.options.routes)
24.   console.log(main + ', context.app.router.options.routes=',
    context.app.router.options.routes)
25.
26.   // 2ième argument
27.   const inject = args[1]
28.   console.log('inject=', typeof inject)
29. }
30.
31. function dumpkeys(message, object) {
32.   // liste des clés de [object]
33.   const ligne = 'Liste des clés [' + message + ']'
34.   console.log(ligne)
35.   // liste des clés
36.   if (object) {
37.     console.log(Object.keys(object))
38.   }
39. }
```

- lignes 31-39 : la fonction [dumpkeys] liste les propriétés de l'objet passé en 2ième paramètre. Cette liste est précédée du message passé en 1er paramètre ;
- ligne 3 : on veut savoir combien d'arguments reçoit la fonction. Pour cela, on utilise la notation [...args] qui va avoir pour effet de mettre les paramètres effectifs de la fonction dans le tableau [args]. On va découvrir qu'il y a deux arguments ;
- ligne 5 : on affiche qui exécute le code, du serveur ou du client ;
- ligne 6 : l'exécuteur du code, client ou serveur ;
- ligne 7 : une constante chaîne de caractères utilisée dans les logs ;
- ligne 12 : on va découvrir que le 1er argument reçu par la plugin est le contexte de l'exécuteur ;
- ligne 14 : liste des clés de l'objet [context] ;
- ligne 16 : on va découvrir que l'objet [context] a une propriété [app] qui représente l'application [nuxt] ;
- ligne 18 : on va découvrir que l'objet [context] a une propriété [route] qui représente la route courante du routeur ;
- ligne 21 : on va découvrir que l'objet [app] a une propriété [router] qui représente le routeur ;
- ligne 23 : l'objet [router.options.routes] représente les différentes routes de l'application ;
- lignes 27-28 : le second argument du plugin est la fonction [inject] que nous avons utilisée dans l'exemple [nuxt-06] ;

10.3 Le plugin exécuté par le serveur

A l'exécution, le serveur affiche la chose suivante :

```
1.  [main server], process.server= true process.client= false
2.  [main server], il y a 2 arguments
3.  Liste des clés [[main server], context]
4.  [
5.    'isStatic',
6.    'isDev',
7.    'isHMR',
8.    'app',
9.    'payload',
10.   'error',
11.   'base',
12.   'env',
13.   'req',
14.   'res',
15.   'ssrContext',
16.   'redirect',
17.   'beforeNuxtRender',
18.   'route',
19.   'next',
20.   '_redirected',
21.   '_errored',
```

```
22.   'params',
23.   'query',
24.   '$axios'
25. ]
```

- lignes 11-12 : nous avons déjà eu l'occasion d'utiliser les propriétés [base] et [env] dont les valeurs proviennent du fichier [nuxt.config.js] ;
- ligne 8 : la propriété [app] désigne l'application [nuxt] ;
- ligne 18 : la propriété [route] désigne la route courante du routeur, ç-à-d la page que va envoyer le serveur ;
- ligne 13 : la requête HTTP du navigateur client ;
- ligne 14 : la réponse HTTP du serveur ;

La liste des propriétés de [context.app] est la suivante :

```
1.  Liste des clés [[main server], context.app]
2.  [
3.    'router',
4.    'nuxt',
5.    'head',
6.    'render',
7.    'data',
8.    'beforeCreate',
9.    'created',
10.   'mounted',
11.   'watch',
12.   'computed',
13.   'methods',
14.   'components',
15.   'context',
16.   '$axios'
17. ]
```

- ligne 3 : la propriété [router] nous donne accès au router de l'application. C'est important sous [nuxt] puisque le routeur est défini par [nuxt] lui-même et non par le développeur. Cette propriété est un accès donné au développeur pour modifier le routeur ;

La liste des propriétés de [context.route] sont les suivantes :

```
1.  Liste des clés [[main server], context.route]
2.  [
3.    'name',
4.    'meta',
5.    'path',
6.    'hash',
7.    'query',
8.    'params',
9.    'fullPath',
10.   'matched'
11. ]
```

La route [context.route] au démarrage du serveur est la suivante :

```
1.  [main server], context.route= {
2.    name: 'index',
3.    meta: [
4.      {}
5.    ],
6.    path: '/',
7.    hash: '',
8.    query: {},
9.    params: {},
10.   fullPath: '/',
11.   matched: [
12.     {
13.       path: '',
14.       regex: /^(?:\/(?=$))?$/i,
15.       components: [Object],
16.       instances: {},
17.       name: 'index',
18.       parent: undefined,
19.       matchAs: undefined,
```

```
20.       redirect: undefined,
21.       beforeEnter: undefined,
22.       meta: {},
23.       props: {}
24.     }
25.   ]
26. }
```

- ligne 2 : on voit que la prochaine page du serveur est [index] et que son chemin est [/] (ligne 10) ;
- ligne 22 : la propriété [meta] permet d'ajouter des propriétés aux routes ;

Les propriétés du routeur [context.app.router] du serveur sont les suivantes :

```
1.  Liste des clés [[main server], context.app.router]
2.  [
3.    'app',
4.    'apps',
5.    'options',
6.    'beforeHooks',
7.    'resolveHooks',
8.    'afterHooks',
9.    'matcher',
10.   'fallback',
11.   'mode',
12.   'history'
13. ]
```

C'est dans la propriété [context.app.router.options.routes] qu'on trouve les différentes routes de l'application :

```
1.  Liste des clés [[main server], context.app.router.options.routes]
2.  [
3.    '0',
4.    '1',
5.    '2'
6.  ]
7.  [main server], context.app.router.options.routes= [
8.    {
9.      path: '/page1',
10.     component: [Function: _d7b6c762],
11.     name: 'page1'
12.   },
13.   {
14.     path: '/page2',
15.     component: [Function: _d79a9860],
16.     name: 'page2'
17.   },
18.   {
19.     path: '/',
20.     component: [Function: _31eaad9f],
21.     name: 'index'
22.   }
23. ]
```

Enfin le 2ième argument :

```
inject= function
```

10.4 La page [index] du serveur

La page [index] est la suivante :

```
1.  <!-- page principale -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="warning">
8.        Home
9.      </b-alert>
10.   </Layout>
```

```
11. </template>
12.
13. <script>
14. /* eslint-disable no-undef */
15. /* eslint-disable no-console */
16. /* eslint-disable nuxt/no-env-in-hooks */
17.
18. import Navigation from '@/components/navigation'
19. import Layout from '@/components/layout'
20.
21. export default {
22.   name: 'Home',
23.   // composants utilisés
24.   components: {
25.     Layout,
26.     Navigation
27.   },
28.   data() {
29.     return {
30.       who: process.server ? 'server' : 'client'
31.     }
32.   },
33.
34.   // cycle de vie
35.   beforeCreate() {
36.     // client et serveur
37.     console.log('[home beforeCreate]')
38.   },
39.   created() {
40.     // client et serveur
41.     console.log('[home ' + this.who + ' created]')
42.     this.dumpkeys('[home ' + this.who + ' created], this.$nuxt', this.$nuxt)
43.     this.dumpkeys('[home ' + this.who + ' created], this.$nuxt.context', this.$nuxt.context)
44.   },
45.   beforeMount() {
46.     // client seulement
47.     console.log('[home ' + this.who + ' beforeMount]')
48.   },
49.   mounted() {
50.     // client seulement
51.     console.log('[home ' + this.who + ' mounted]')
52.   },
53.   methods: {
54.     dumpkeys(message, object) {
55.       // liste des clés de [object]
56.       const ligne = 'Liste des clés [' + message + ']'
57.       console.log(ligne)
58.       if (object) {
59.         console.log(Object.keys(object))
60.       }
61.     }
62.   }
63. }
64. </script>
```

- ligne 43 : on a déjà vu que le contexte d'une page pouvait être trouvé dans [this.$nuxt.context] ;
- ligne 42 : on affiche également les propriétés de l'objet [this.$nuxt] ;

Exécutée par le serveur, cette page donne naissance aux logs suivants :

```
1.  [home beforeCreate]
2.  [home server created]
3.  Liste des clés [[home server created], this.$nuxt]
4.  [
5.    '_uid',
6.    '_isVue',
7.    '$options',
8.    '_renderProxy',
9.    '_self',
10.   '$parent',
11.   '$root',
12.   '$children',
13.   '$refs',
```

```
14.   '_watcher',
15.   '_inactive',
16.   '_directInactive',
17.   '_isMounted',
18.   '_isDestroyed',
19.   '_isBeingDestroyed',
20.   '_events',
21.   '_hasHookEvent',
22.   '_vnode',
23.   '_staticTrees',
24.   '$vnode',
25.   '$slots',
26.   '$scopedSlots',
27.   '_c',
28.   '$createElement',
29.   '$attrs',
30.   '$listeners',
31.   '_routerRoot',
32.   '_router',
33.   '_route',
34.   '_bv__modal',
35.   '_bv__toast',
36.   '_vueMeta',
37.   'nuxt',
38.   '_watchers',
39.   'refreshOnlineStatus',
40.   'refresh',
41.   'errorChanged',
42.   'setLayout',
43.   'loadLayout',
44.   '_data',
45.   'layoutName',
46.   'layout',
47.   'isOnline',
48.   '_computedWatchers',
49.   'isOffline',
50.   'error',
51.   'context'
52. ]
```

Les propriétés du contexte serveur dans la page [index] sont les suivantes :

```
1.  Liste des clés [[home server created], this.$nuxt.context]
2.  [
3.    'isStatic',
4.    'isDev',
5.    'isHMR',
6.    'app',
7.    'payload',
8.    'error',
9.    'base',
10.   'env',
11.   'req',
12.   'res',
13.   'ssrContext',
14.   'redirect',
15.   'beforeNuxtRender',
16.   'route',
17.   'next',
18.   '_redirected',
19.   '_errored',
20.   'params',
21.   'query',
22.   '$axios'
23. ]
```

Ce sont les mêmes propriétés que dans l'objet [context] du plugin.

10.5 Le plugin exécuté par le client

Une fois que le serveur a envoyé la page [index] au navigateur client, les scripts client prennent la main. Le plugin [main.js] va alors être exécuté. Ces logs sont les suivants :

```
1.  [main server], process.server= false process.client= true
2.  [main client], il y a 2 arguments
3.  Liste des clés [[main client], context]
4.  Array(17)0: "isStatic"1: "isDev"2: "isHMR"3: "app"4: "payload"5: "error"6: "base"7: "env"8:
    "redirect"9: "nuxtState"10: "route"11: "next"12: "_redirected"13: "_errored"14: "params"15:
    "query"16: "$axios"length: 17__proto__: Array(0)
5.  Liste des clés [[main client], context.app]
6.  Array(14)0: "router"1: "nuxt"2: "head"3: "render"4: "data"5: "beforeCreate"6: "created"7:
    "mounted"8: "watch"9: "computed"10: "methods"11: "components"12: "context"13: "$axios"length:
    14__proto__: Array(0)
7.  Liste des clés [[main client], context.route]
8.  Array(8)0: "name"1: "meta"2: "path"3: "hash"4: "query"5: "params"6: "fullPath"7:
    "matched"length: 8__proto__: Array(0)
9.  [main client], context.route= ObjectfullPath: "/"hash: ""matched: [{…}]meta: [{…}]name:
    "index"params: {}path: "/"query: {}__proto__: Object
10. Liste des clés [[main client], context.app.router]
11. Array(10)0: "app"1: "apps"2: "options"3: "beforeHooks"4: "resolveHooks"5: "afterHooks"6:
    "matcher"7: "fallback"8: "mode"9: "history"length: 10__proto__: Array(0)
12. Liste des clés [[main client], context.app.router.options.routes]
13. Array(3)0: "0"1: "1"2: "2"length: 3__proto__: Array(0)
14. [main client], context.app.router.options.routes= Array(3)0: {path: "/page1", name: "page1",
    component: ƒ}1: {path: "/page2", name: "page2", component: ƒ}2: {path: "/", name: "index",
    component: ƒ}length: 3__proto__: Array(0)
15. inject= function
```

On retrouve des propriétés analogues à celles trouvées côté serveur avec certaines propriétés qui ont disparu et d'autres qui sont apparues. Ainsi ligne 4, on ne retrouve pas les propriétés [req, res] qui étaient les requêtes HTTP du navigateur client et la réponse HTTP du serveur.

10.6 La page [index] du client

La page [index] du client produit les logs suivants :

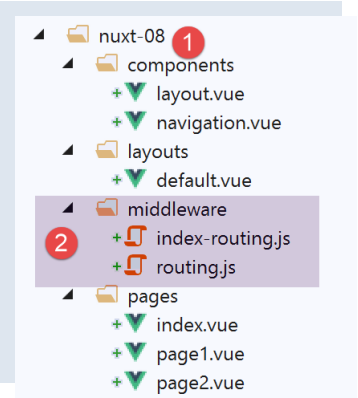
```
1.  [home beforeCreate]
2.  [home client created]
3.  Liste des clés [[home client created], this.$nuxt]
4.  (51) ["_uid", "_isVue", "$options", "_renderProxy", "_self", "$parent", "$root", "$children",
    "$refs", "_watcher", "_inactive", "_directInactive", "_isMounted", "_isDestroyed",
    "_isBeingDestroyed", "_events", "_hasHookEvent", "_vnode", "_staticTrees", "$vnode",
    "$slots", "$scopedSlots", "_c", "$createElement", "$attrs", "$listeners", "_routerRoot",
    "_router", "_route", "_bv__modal", "_bv__toast", "_vueMeta", "nuxt", "_watchers",
    "refreshOnlineStatus", "refresh", "errorChanged", "setLayout", "loadLayout", "_data",
    "layoutName", "layout", "isOnline", "_computedWatchers", "isOffline", "error", "context",
    "_name", "setTransitions", "$loading", "$el"]
5.  Liste des clés [[home client created], this.$nuxt.context]
6.  (17) ["isStatic", "isDev", "isHMR", "app", "payload", "error", "base", "env", "redirect",
    "nuxtState", "route", "next", "_redirected", "_errored", "params", "query", "$axios"]
7.  [home client beforeMount]
8.  [home client mounted]
```

- ligne 4 : les propriétés de l'objet [this.$nuxt]. C'est un objet riche avec 51 propriétés ;
- ligne 6 : les propriétés de l'objet [this.$nuxt.context]. On retrouve les mêmes propriétés que dans l'objet [context] du plugin client ;

11 Exemple [nuxt-08] : middlewares de routage

Dans cet exemple, nous introduisons la notion de middlewares de routage, des scripts exécutés à chaque changement de route.

L'exemple [nuxt-08] est obtenu initialement par recopie du projet [nuxt-01] :



Les middlewares de routage doivent être dans un dossier appelé [middleware] [2]. Il peut y avoir un routage à deux niveaux :

- un routage appliqué à chaque navigation. Celui-ci fait alors l'objet d'une déclaration dans le fichier [nuxt.config.js] ;
- un routage appliqué à une page particulière, lorsque celle-ci est la cible du routage. Ce routage est alors déclaré dans cette page cible ;

11.1 Routage général

Le fichier [middleware / routing.js] assurera le routage général. Il fait l'objet de la déclaration suivante dans le fichier [nuxt.config.js] :

```
1.    router: {
2.      base: '/nuxt-08/',
3.      middleware: ['routing']
4.  },
```

Les middlewares de routage général sont une propriété du routeur (ligne 1). Il peut y avoir plusieurs middlewares de routage. C'est pourquoi ligne 3, la valeur de la propriété [middleware] est un tableau. On voit qu'on n'utilise pas de chemin pour désigner le middleware. Il sera automatiquement cherché dans le dossier [middleware] du projet ;

Le middleware [routing] ne fait ici que des logs :

```
1.  /* eslint-disable no-undef */
2.  /* eslint-disable no-console */
3.  export default function(...args) {
4.    // qui exécute ce code ?
5.    console.log('[routing], process.server=', process.server, 'process.client=', process.client)
6.    const who = process.server ? 'server' : 'client'
7.    const routing = '[routing ' + who + ']'
8.    // nbre d'arguments
9.    console.log(routing + ', il y a', args.length, 'argument(s)')
10.
11.   // 1er argument
12.   const context = args[0]
13.   // clés du contexte
14.   dumpkeys(routing + ', context', context)
15.   // l'application
16.   dumpkeys(routing + ', context.app', context.app)
17.   // la route
18.   dumpkeys(routing + ', context.route', context.route)
```

```
19.   console.log(routing + ', context.route=', context.route)
20.   // le router
21.   dumpkeys(routing + ', context.app.router', context.app.router)
22.   // le router.options.routes
23.   dumpkeys(routing + ', context.app.router.options.routes', context.app.router.options.routes)
24.   console.log(routing + ', context.app.router.options.routes=',
    context.app.router.options.routes)
25. }
26.
27. function dumpkeys(message, object) {
28.   // liste des clés de [object]
29.   const ligne = 'Liste des clés [' + message + ']'
30.   console.log(ligne)
31.   // liste des clés
32.   if (object) {
33.     console.log(Object.keys(object))
34.   }
35. }
```

- ligne 3 : on va découvrir que le middleware reçoit un argument : le contexte de l'exécuteur (serveur ou client) ;
- lignes 4-25 : on affiche les propriétés de différents objets pour savoir ce qui est utilisable. On va découvrir que le contexte du middleware est quasi identique au contexte du plugin ;

11.2 Routage pour une page particulière

On veut contrôler la façon dont on arrive à la page [index]. Pour cela, il nous faut introduire la propriété [middleware] dans cette page [index] :

```
1.  <script>
2.  /* eslint-disable no-undef */
3.  /* eslint-disable no-console */
4.  /* eslint-disable nuxt/no-env-in-hooks */
5.
6.  import Navigation from '@/components/navigation'
7.  import Layout from '@/components/layout'
8.
9.  export default {
10.   name: 'Home',
11.   // composants utilisés
12.   components: {
13.     Layout,
14.     Navigation
15.   },
16.   // cycle de vie
17.   beforeCreate() {
18.     // client et serveur
19.     console.log('[home beforeCreate]')
20.   },
21.   created() {
22.     // client et serveur
23.     console.log('[home created]')
24.   },
25.   beforeMount() {
26.     // client seulement
27.     console.log('[home beforeMount]')
28.   },
29.   mounted() {
30.     // client seulement
31.     console.log('[home mounted]')
32.   },
33.   // routage
34.   middleware: ['index-routing']
35. }
36. </script>
```

- ligne 34 : la propriété [middleware] liste les scripts à exécuter à chaque fois que la prochaine page affichée est la page [index]. Là encore, ces scripts seront cherchés dans le dossier [middleware] du projet ;

Le middleware [index-routing] est le suivant :

```
1.  /* eslint-disable no-undef */
2.  /* eslint-disable no-console */
3.  export default function(...args) {
4.    // qui exécute ce code ?
5.    console.log('[index-routing], process.server=', process.server, 'process.client=',
    process.client)
6.    const who = process.server ? 'server' : 'client'
7.    const indexRouting = '[index-routing ' + who + ']'
8.    // nbre d'arguments
9.    console.log(indexRouting + ', il y a', args.length, 'argument(s)')
10.
11.   // 1er argument
12.   const context = args[0]
13.   // clés du contexte
14.   dumpkeys(indexRouting + ', context', context)
15.   // l'application
16.   dumpkeys(indexRouting + ', context.app', context.app)
17.   // la route
18.   dumpkeys(indexRouting + ', context.route', context.route)
19.   console.log(indexRouting + ', context.route=', context.route)
20.   // le router
21.   dumpkeys(indexRouting + ', context.app.router', context.app.router)
22.   // le router.options.routes
23.   dumpkeys(indexRouting + ', context.app.router.options.routes',
    context.app.router.options.routes)
24.   console.log(indexRouting + ', context.app.router.options.routes=',
    context.app.router.options.routes)
25.   // d'où vient-on ?
26.   if (context.from) {
27.     console.log('from=', context.from)
28.   }
29. }
30.
31. function dumpkeys(message, object) {
32.   // liste des clés de [object]
33.   const ligne = 'Liste des clés [' + message + ']'
34.   console.log(ligne)
35.   // liste des clés
36.   if (object) {
37.     console.log(Object.keys(object))
38.   }
39. }
```

Le code de [index-routing] est identique à celui de [routing] et produit les mêmes résultats. Ce qui nous intéresse c'est de voir quand ces deux middlewares sont exécutés.

11.3 Exécution du projet

Nous exécutons le projet. Les logs sont alors les suivants :

C'est le script [routing] qui est exécuté en premier par le serveur :

```
1.  [routing], process.server= true process.client= false
2.  [routing server], il y a 1 argument(s)
3.  Liste des clés [[routing server], context]
4.  [ 'isStatic',
5.    'isDev',
6.    'isHMR',
7.    'app',
8.    'payload',
9.    'error',
10.   'base',
11.   'env',
12.   'req',
13.   'res',
14.   'ssrContext',
15.   'redirect',
16.   'beforeNuxtRender',
17.   'route',
18.   'next',
19.   '_redirected',
20.   '_errored',
```

```
21.   'params',
22.   'query',
23.   '$axios' ]
24. Liste des clés [[routing server], context.app]
25. [ 'router',
26.   'nuxt',
27.   'head',
28.   'render',
29.   'data',
30.   'beforeCreate',
31.   'created',
32.   'mounted',
33.   'watch',
34.   'computed',
35.   'methods',
36.   'components',
37.   'context',
38.   '$axios' ]
39. Liste des clés [[routing server], context.route]
40. [ 'name',
41.   'meta',
42.   'path',
43.   'hash',
44.   'query',
45.   'params',
46.   'fullPath',
47.   'matched' ]
48. [routing server], context.route= { name: 'index',
49.   meta: [ {} ],
50.   path: '/',
51.   hash: '',
52.   query: {},
53.   params: {},
54.   fullPath: '/',
55.   matched:
56.    [ { path: '',
57.        regex: /^(?:\/(?=$))?$/i,
58.        components: [Object],
59.        instances: {},
60.        name: 'index',
61.        parent: undefined,
62.        matchAs: undefined,
63.        redirect: undefined,
64.        beforeEnter: undefined,
65.        meta: {},
66.        props: {} } ] }
67. Liste des clés [[routing server], context.app.router]
68. [ 'app',
69.   'apps',
70.   'options',
71.   'beforeHooks',
72.   'resolveHooks',
73.   'afterHooks',
74.   'matcher',
75.   'fallback',
76.   'mode',
77.   'history' ]
78. Liste des clés [[routing server], context.app.router.options.routes]
79. [ '0', '1', '2' ]
80. [routing server], context.app.router.options.routes= [ { path: '/page1',
81.     component: [Function: _61cefe10],
82.     name: 'page1' },
83.   { path: '/page2',
84.     component: [Function: _61dd1591],
85.     name: 'page2' },
86.   { path: '/', component: [Function: _00d5e140], name: 'index' } ]
```

On retrouve là ce qu'on avait obtenu avec les plugins.

- ligne 15 : la propriété [redirect] est souvent utilisée dans les middlewares : elle permet de changer la cible du routage en cours ;

Puis, parce que la page qui va être affichée est la page [index], le serveur exécute le script [index-routing] et affiche les logs suivants :

```
1.  [index-routing], process.server= true process.client= false
2.  [index-routing server], il y a 1 argument(s)
3.  Liste des clés [[index-routing server], context]
4.  [ 'isStatic',
5.    'isDev',
6.    'isHMR',
7.    'app',
8.    'payload',
9.    'error',
10.   'base',
11.   'env',
12.   'req',
13.   'res',
14.   'ssrContext',
15.   'redirect',
16.   'beforeNuxtRender',
17.   'route',
18.   'next',
19.   '_redirected',
20.   '_errored',
21.   'params',
22.   'query',
23.   '$axios' ]
24. Liste des clés [[index-routing server], context.app]
25. [ 'router',
26.   'nuxt',
27.   'head',
28.   'render',
29.   'data',
30.   'beforeCreate',
31.   'created',
32.   'mounted',
33.   'watch',
34.   'computed',
35.   'methods',
36.   'components',
37.   'context',
38.   '$axios' ]
39. Liste des clés [[index-routing server], context.route]
40. [ 'name',
41.   'meta',
42.   'path',
43.   'hash',
44.   'query',
45.   'params',
46.   'fullPath',
47.   'matched' ]
48. [index-routing server], context.route= { name: 'index',
49.   meta: [ {} ],
50.   path: '/',
51.   hash: '',
52.   query: {},
53.   params: {},
54.   fullPath: '/',
55.   matched:
56.    [ { path: '',
57.        regex: /^(?:\/(?=$))?$/i,
58.        components: [Object],
59.        instances: {},
60.        name: 'index',
61.        parent: undefined,
62.        matchAs: undefined,
63.        redirect: undefined,
64.        beforeEnter: undefined,
65.        meta: {},
66.        props: {} } ] }
67. Liste des clés [[index-routing server], context.app.router]
68. [ 'app',
69.   'apps',
70.   'options',
71.   'beforeHooks',
```

```
72.   'resolveHooks',
73.   'afterHooks',
74.   'matcher',
75.   'fallback',
76.   'mode',
77.   'history' ]
78. Liste des clés [[index-routing server], context.app.router.options.routes]
79. [ '0', '1', '2' ]
80. [index-routing server], context.app.router.options.routes= [ { path: '/page1',
81.     component: [Function: _61cefe10],
82.     name: 'page1' },
83.   { path: '/page2',
84.     component: [Function: _61dd1591],
85.     name: 'page2' },
86.   { path: '/', component: [Function: _00d5e140], name: 'index' } ]
```

Les résultats obtenus avec le script [index-routing] sont analogues à ceux obtenus avec le script [routing].

Une fois la page [index] reçue par le navigateur client, les scripts client prennent la main. Les logs deviennent les suivants :

```
1.  [home beforeCreate]
2.  [home created]
3.  [home beforeMount]
4.  [home mounted]
```

On voit donc qu'au démarrage de l'application **le client n'exécute aucun middleware**. Cela veut dire que cela se produira à chaque fois que l'utilisateur forcera un appel au serveur. Les middlewares ne sont exécutés par le client que lors d'une navigation au sein du client. Naviguons par exemple vers la page [page1] (nous sommes sur la page [index]) avec le lien [Page 1]. Les logs sont alors les suivants :

```
1.  [routing], process.server= false process.client= true
2.  [routing client], il y a 1 argument(s)
3.  Liste des clés [[routing client], context]
4.  (18) ["isStatic", "isDev", "isHMR", "app", "payload", "error", "base", "env", "redirect",
    "nuxtState", "route", "next", "_redirected", "_errored", "params", "query", "$axios", "from"]
5.  Liste des clés [[routing client], context.app]
6.  (14) ["router", "nuxt", "head", "render", "data", "beforeCreate", "created", "mounted",
    "watch", "computed", "methods", "components", "context", "$axios"]
7.  Liste des clés [[routing client], context.route]
8.  (8) ["name", "meta", "path", "hash", "query", "params", "fullPath", "matched"]
9.  [routing client], context.route= {name: "page1", meta: Array(1), path: "/page1", hash: "",
    query: {…}, …}
10. Liste des clés [[routing client], context.app.router]
11. (10) ["app", "apps", "options", "beforeHooks", "resolveHooks", "afterHooks", "matcher",
    "fallback", "mode", "history"]
12. Liste des clés [[routing client], context.app.router.options.routes]
13. (3) ["0", "1", "2"]
14. [routing client], context.app.router.options.routes= (3) [{…}, {…}, {…}]0: {path: "/page1",
    name: "page1", component: ƒ}1: {path: "/page2", name: "page2", component: ƒ}2: {path: "/",
    name: "index", component: ƒ}length: 3__proto__: Array(0)
15. [page1 beforeCreate]
16. [page1 created]
17. [page1 beforeMount]
18. [page1 mounted]
```

- ligne 2 : le middleware [routing] est exécuté par le client ;
- ligne 4 : notez la propriété [from] : c'est la route d'où l'on vient ;
- ligne 9 : [context.route] est la route où l'on va ;
- lignes 15-18 : affichage de la page [page1] ;

Maintenant revenons à la page [index] avec le lien [Home]. Les logs sont alors les suivants :

```
1.  [routing], process.server= false process.client= true
2.  [routing client], il y a 1 argument(s)
3.  Liste des clés [[routing client], context]
4.  (18) ["isStatic", "isDev", "isHMR", "app", "payload", "error", "base", "env", "redirect",
    "nuxtState", "route", "next", "_redirected", "_errored", "params", "query", "$axios", "from"]
5.  Liste des clés [[routing client], context.app]
```

```
6.  (14) ["router", "nuxt", "head", "render", "data", "beforeCreate", "created", "mounted",
    "watch", "computed", "methods", "components", "context", "$axios"]
7.  Liste des clés [[routing client], context.route]
8.  (8) ["name", "meta", "path", "hash", "query", "params", "fullPath", "matched"]
9.  [routing client], context.route= {name: "index", meta: Array(1), path: "/", hash: "", query:
    {…}, …}
10. Liste des clés [[routing client], context.app.router]
11. (10) ["app", "apps", "options", "beforeHooks", "resolveHooks", "afterHooks", "matcher",
    "fallback", "mode", "history"]
12. Liste des clés [[routing client], context.app.router.options.routes]
13. (3) ["0", "1", "2"]
14. [routing client], context.app.router.options.routes= (3) [{…}, {…}, {…}]0: {path: "/page1",
    name: "page1", component: ƒ}1: {path: "/page2", name: "page2", component: ƒ}2: {path: "/",
    name: "index", component: ƒ}length: 3__proto__: Array(0)
15. [index-routing], process.server= false process.client= true
16. [index-routing client], il y a 1 argument(s)
17. Liste des clés [[index-routing client], context]
18. (18) ["isStatic", "isDev", "isHMR", "app", "payload", "error", "base", "env", "redirect",
    "nuxtState", "route", "next", "_redirected", "_errored", "params", "query", "$axios", "from"]
19. Liste des clés [[index-routing client], context.app]
20. (14) ["router", "nuxt", "head", "render", "data", "beforeCreate", "created", "mounted",
    "watch", "computed", "methods", "components", "context", "$axios"]
21. Liste des clés [[index-routing client], context.route]
22. (8) ["name", "meta", "path", "hash", "query", "params", "fullPath", "matched"]
23. [index-routing client], context.route= {name: "index", meta: Array(1), path: "/", hash: "",
    query: {…}, …}
24. Liste des clés [[index-routing client], context.app.router]
25. (10) ["app", "apps", "options", "beforeHooks", "resolveHooks", "afterHooks", "matcher",
    "fallback", "mode", "history"]
26. Liste des clés [[index-routing client], context.app.router.options.routes]
27. (3) ["0", "1", "2"]
28. [index-routing client], context.app.router.options.routes= (3) [{…}, {…}, {…}]0: {path:
    "/page1", name: "page1", component: ƒ}1: {path: "/page2", name: "page2", component: ƒ}2:
    {path: "/", name: "index", component: ƒ}length: 3__proto__: Array(0)
29. from= {name: "page1", meta: Array(1), path: "/page1", hash: "", query: {…}, …}
30. [home beforeCreate]
31. [home created]
32. [home beforeMount]
33. [home mounted]
```

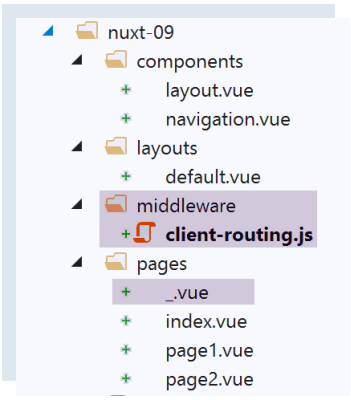
- lignes 1-15 : le client exécute le middleware [routing]. C'est normal. Il est exécuté à chaque changement de route;
- lignes 16-29 : le client exécute le middleware [index-routing], ceci parce que :
  - [index] est la cible de la route courante (cf ligne 23) ;
  - la page [index] a défini un middleware, nommément [index-routing] ;

On voit donc que les middlewares de routage général sont exécutés par le client avant les middlewares attachés aux pages.

12 Exemple [nuxt-09] : contrôle de la navigation

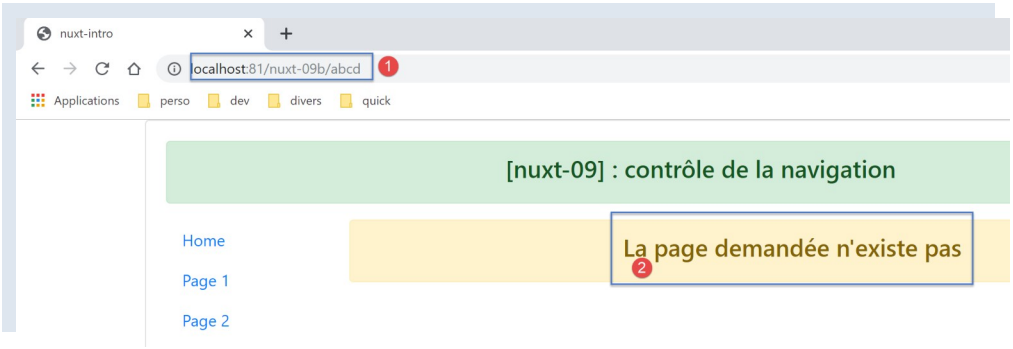
L'exemple [nuxt-09] utilise un middleware pour contrôler la navigation du client. Par ailleurs, on ajoute une nouvelle vue pour les cas où l'utilisateur demande au serveur une URL qui n'existe pas dans l'application.

L'exemple [nuxt-09] est initialement obtenu par recopie de l'exemple [nuxt-01] :



12.1 La page [_.vue]

Nous avons dit que les routes de l'application étaient construites à partir du contenu du dossier [pages]. Ici, nous avons ajouté dans ce dossier la page [_.vue]. Cette page particulière est affichée à chaque fois que l'application est routée vers une page qui n'existe pas. Dans notre exemple ici, cela ne peut pas arriver pour le client. Mais cela peut arriver pour le serveur si par exemple on lui demande l'URL [/nuxt-09/**abcd**]. La page [abcd] n'existant pas, c'est la page [_.vue] qui va être affichée. Ici, ce sera la suivante :



Le code de la page [_.vue] est le suivant :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <!-- mise en page -->
4.    <Layout :left="true" :right="true">
5.      <!-- alerte dans la colonne de droite -->
6.      <template slot="right">
7.        <!-- message sur fond jaune -->
8.        <b-alert show variant="warning" align="center">
9.          <h4>La page demandée n'existe pas</h4>
10.       </b-alert>
11.     </template>
12.     <!-- menu de navigation dans la colonne de gauche -->
13.     <Navigation slot="left" />
14.   </Layout>
15. </template>
16.
```

```
17. <script>
18. /* eslint-disable no-undef */
19. /* eslint-disable no-console */
20. /* eslint-disable nuxt/no-env-in-hooks */
21.
22. import Layout from '@/components/layout'
23. import Navigation from '@/components/navigation'
24.
25. export default {
26.   name: '404',
27.   // composants utilisés
28.   components: {
29.     Layout,
30.     Navigation
31.   },
32.   // cycle de vie
33.   beforeCreate() {
34.     // client et serveur
35.     console.log('[404 beforeCreate]')
36.   },
37.   created() {
38.     // client et serveur
39.     console.log('[404 created]')
40.   },
41.   beforeMount() {
42.     // client seulement
43.     console.log('[404 beforeMount]')
44.   },
45.   mounted() {
46.     // client seulement
47.     console.log('[404 mounted]')
48.   }
49. }
50. </script>
```

12.2 Le middleware du client

Le code du middleware du client [client-routing] est le suivant :

```
1.  /* eslint-disable no-undef */
2.  /* eslint-disable no-console */
3.  export default function({ route, from, redirect }) {
4.    // seulement le client
5.    if (process.client) {
6.      console.log('[client-routing]')
7.      // ordre de navigation souhaité
8.      const routes = ['index', 'page1', 'page2', 'index']
9.      // route courante
10.     const current = route.name
11.     // route précédente
12.     const previous = from.name
13.     // on veut une navigation circulaire
14.     // routes[i] vers routes[i+1]
15.     for (let i = 0; i < routes.length - 1; i++) {
16.       if (previous === routes[i] && current !== routes[i + 1]) {
17.         // on reste sur la même page
18.         redirect({ name: routes[i] })
19.         return
20.       }
21.     }
22.   }
23. }
```

- ligne 3 : nous savons que la fonction de routing ne reçoit qu'un paramètre, l'objet [context] de celui qui l'exécute, serveur ou client. La notation [`function({ route, from, redirect })`]
  - est équivalente à [`function({ route:route, from:from, redirect:redirect })`] ;
  - ce qui fait que `{ route:route, from:from, redirect:redirect } <-- context` ;
  - ce qui crée trois paramètres [route, from, redirect] tels que :
    - route=context.route ;
    - redirect=context.redirect ;
    - from=context.from ;

La documentation de [nuxt] utilise abondamment cette notation. Il faut la connaître ;

- ligne 8 : un tableau des noms de pages dans l'ordre de navigation souhaité
- ligne 10 : le nom de la page de destination du routage courant ;
- lignes 12 : le nom de la page précédente du routage courant ;
- ligne 14 : comme exercice, on ne va autoriser qu'une navigation circulaire [index --> page1 --> page2 --> index] ;
- lignes 15-21 : on parcourt le tableau donnant l'ordre de navigation souhaité ;
- ligne 16 : si on découvre que routes[i] était la dernière page routée alors la suivante doit être routes[i+1] ;
- lignes 18-19 : si ce n'est pas le cas, on redirige l'application vers routes[i], ç-à-d qu'on ne change pas de page : on refuse la navigation ;

12.3 Exécution

On exécute l'exemple avec le fichier [nuxt.config.js] suivant :

```
1.  export default {
2.    mode: 'universal',
3.    /*
4.     ** Headers of the page
5.     */
6.    head: {
7.      title: process.env.npm_package_name || "Introduction à nuxt.js par l'exemple",
8.      meta: [
9.        { charset: 'utf-8' },
10.       { name: 'viewport', content: 'width=device-width, initial-scale=1' },
11.       { hid: 'description', name: 'description', content: process.env.npm_package_description
    || '' }
12.     ],
13.     link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
14.   },
15.   /*
16.    ** Customize the progress-bar color
17.    */
18.   loading: { color: '#fff' },
19.   /*
20.    ** Global CSS
21.    */
22.   css: [],
23.   /*
24.    ** Plugins to load before mounting the App
25.    */
26.   plugins: [],
27.   /*
28.    ** Nuxt.js dev-modules
29.    */
30.   buildModules: [
31.     // Doc: https://github.com/nuxt-community/eslint-module
32.     '@nuxtjs/eslint-module'
33.   ],
34.   /*
35.    ** Nuxt.js modules
36.    */
37.   modules: [
38.     // Doc: https://bootstrap-vue.js.org
39.     'bootstrap-vue/nuxt',
40.     // Doc: https://axios.nuxtjs.org/usage
41.     '@nuxtjs/axios'
42.   ],
43.   /*
44.    ** Axios module configuration
45.    ** See https://axios.nuxtjs.org/options
46.    */
47.   axios: {},
48.   /*
49.    ** Build configuration
50.    */
51.   build: {
52.     /*
53.      ** You can extend webpack config here
54.      */
```

```
55.     extend(config, ctx) {}
56.   },
57.   // répertoire du code source
58.   srcDir: 'nuxt-09',
59.   router: {
60.     base: '/nuxt-09/',
61.     middleware: ['client-routing']
62.   },
63.   // serveur
64.   server: {
65.     port: 81, // default: 3000
66.     host: 'localhost' // default: localhost
67.   }
68. }
```

Vérifiez les points suivants :

- lorsque vous êtes sur la page [Home], vous ne pouvez naviguer que vers la page [Page 1] ;
- lorsque vous êtes sur la page [Page 1], vous ne pouvez naviguer que vers la page [Page 2] ;
- lorsque vous êtes sur la page [Page 2], vous ne pouvez naviguer que vers la page [Home] ;
- lorsque vous demandez une URL incorrecte telle que [http://localhost:81/nuxt-09b/**abcd**] alors vous obtenez la vue qui indique que la page demandée n'existe pas ;

13 Exemple [nuxt-10] : asyncData et loading

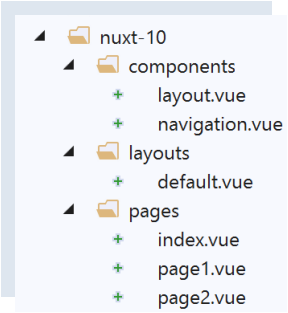
La fonction [asyncData] dans une page permet de charger de façon asynchrone des données souvent externes. [nuxt] attend la fin de la fonction [asyncData] avant de commencer le cycle de vie de la page. Celle-ci n'est donc rendue qu'une fois les données externes obtenues. Elle est exécutée aussi bien par le serveur que par le client avec les règles suivantes :

- lorsque la page est demandée directement au serveur, seul le serveur exécute la fonction [asyncData] ;
- ensuite lors de la navigation client, seul le client exécute la fonction [asyncData] ;

Au final seul l'un des deux, client ou serveur, exécute la fonction. Par ailleurs, quand c'est le client qui exécute la fonction [asyncData], [nuxt] affiche une barre de progression qui peut être paramétrée.

La fonction [asyncData] permet de délivrer aux moteurs de recherche des pages avec leurs données ce qui les rend plus signifiantes.

L'exemple [nuxt-10] est obtenu initialement par recopie du projet [nuxt-01] :



Seule la page [page1] évolue.

13.1 La page [page1]

Le code de la page [page1] est le suivant :

```
1.  <!-- vue n° 1 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="primary"> Page 1 -- result={{ result }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13. /* eslint-disable nuxt/no-timing-in-fetch-data */
14.
15. import Navigation from '@/components/navigation'
16. import Layout from '@/components/layout'
17.
18. export default {
19.   name: 'Page1',
20.   // composants utilisés
21.   components: {
22.     Layout,
23.     Navigation
24.   },
25.   // données asynchrones
26.   asyncData(context) {
27.     // log
28.     console.log('[page1 asyncData started]')
29.     // on rend une promesse
```

```
30.     return new Promise(function(resolve, reject) {
31.       // on simule une fonction asynchrone
32.       setTimeout(function () {
33.         // on rend le résultat asynchrone - un nombre aléatoire ici
34.         resolve({ result: Math.floor(Math.random() * Math.floor(100)) })
35.         // log
36.         console.log('[page1 asyncData finished]')
37.       }, 5000)
38.     })
39.   },
40.
41.   // cycle de vie
42.   beforeCreate() {
43.     console.log('[page1 beforeCreate]')
44.   },
45.   created() {
46.     console.log('[page1 created]')
47.   },
48.   beforeMount() {
49.     console.log('[page1 beforeMount]')
50.   },
51.   mounted() {
52.     console.log('[page1 mounted]')
53.   }
54. }
55. </script>
```

- lignes 26-39 : la fonction [asyncData]. Nous avons déjà étudié cette fonction (cf paragraphe lien). Elle est exécutée **avant** le cycle de vie de la page. Pour cette raison, on ne peut utiliser le mot clé [this] dans la fonction ;
- ligne 30 : elle doit rendre une promesse [Promise] ou utiliser la syntaxe async / await ;
- lignes 32-37 : la fonction asynchrone de la promesse est simulée avec une attente de 5 secondes (ligne 37) ;
- ligne 34 : le résultat de la fonction asynchrone est rendu sous la forme d'un objet {result :...}. L'objet asynchrone rendu par la fonction [asyncData] **est intégré dans l'objet [data] de la page**. C'est pourquoi l'objet [result] est-il disponible ligne 7 du template alors même que la page n'avait pas défini d'objet [data] ;

13.2 Configuration de la barre de progression de [asyncData]

Lorsque la page [page1] est la cible d'une navigation au sein du client (mode SPA), le client exécute la fonction [asyncData] et [nuxt] affiche alors une barre de progression qu'elle cache lorsque la fonction [asyncData] a rendu son résultat. La propriété [loading] du fichier [nuxt.config.js] permet de configurer cette barre :

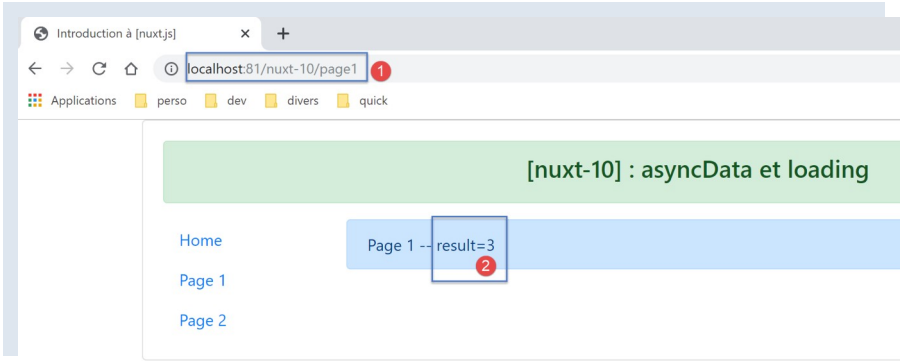
```
1.  loading: {
2.      color: 'blue',
3.      height: '5px',
4.      throttle: 200,
5.      continuous: true
6.  },
```

Par défaut, l'image d'attente de [nuxt] est une barre de progression, faisant la largeur de la page. Cette barre a une couleur et une épaisseur. Le trait coloré grandit progressivement de 0 % à 100 % de sa taille, plus ou moins vite donc selon la durée de l'attente.

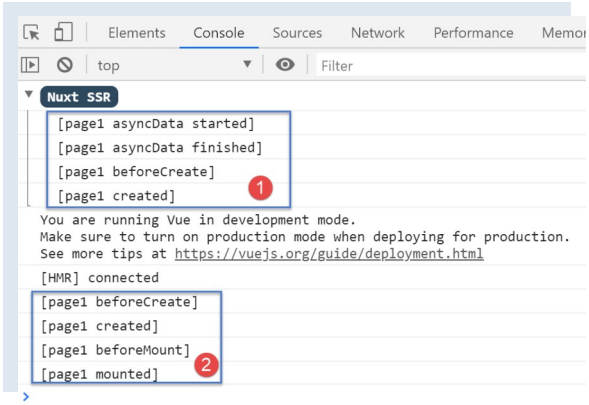
- ligne 2 : fixe la couleur de la barre de progression ;
- ligne 3 : fixe l'épaisseur de la barre en pixels ;
- ligne 4 : [throttle] est le délai en millisecondes avant que l'animation ne démarre. Cela permet de ne pas avoir d'image d'animation lorsque la fonction [asyncData] rend son résultat rapidement ;
- ligne 5 : [continuous] fixe le comportement de l'animation de la barre de progression. Par défaut, la barre grandit progressivement de 0 % à 100 % de sa taille, plus ou moins vite selon la durée de l'attente. Avec [continuous:true], le barre colorée grandit à vitesse constante de 0 à 100 % de sa taille, puis recommence tant que la fonction [asyncData] n'a pas rendu son résultat ;

13.3 Exécution

Lançons l'application, puis demandons, **à la main**, au serveur la page [page1] :



Les logs sont les suivants :



- on voit que seul le serveur [1] a exécuté la fonction [asyncData] et il l'a fait avant le cycle de la page ;

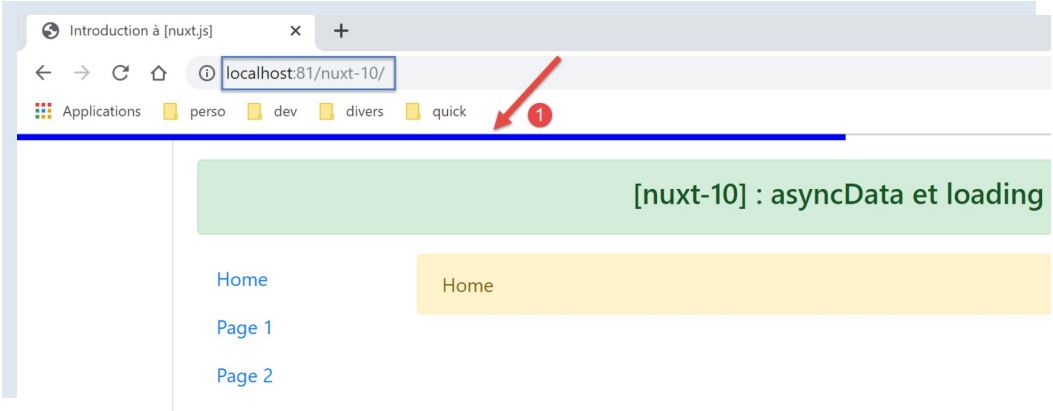
Maintenant examinons la page envoyée par le serveur (code source) :

```
1.  <!doctype html>
2.  <html data-n-head-ssr>
3.  <head>
4.    <title>Introduction à [nuxt.js]</title>
5.    <meta data-n-head="ssr" charset="utf-8">
6.    <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
7.    <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing
    loading asyncdata middleware plugins store">
8.    <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
9.    <base href="/nuxt-10/">
10.   <link rel="preload" href="/nuxt-10/_nuxt/runtime.js" as="script">
11.   <link rel="preload" href="/nuxt-10/_nuxt/commons.app.js" as="script">
12.   <link rel="preload" href="/nuxt-10/_nuxt/vendors.app.js" as="script">
13.   <link rel="preload" href="/nuxt-10/_nuxt/app.js" as="script">
14. ...
15. </head>
16. <body>
17.   <div data-server-rendered="true" id="__nuxt">
18.     <div id="__layout">
19.       <div class="container">
20.         <div class="card">
21.           <div class="card-body">
22.             <div role="alert" aria-live="polite" aria-atomic="true" align="center" class="alert
    alert-success">
23.             <h4>[nuxt-10] : asyncData et loading</h4>
24.             </div>
25.             <div>
26.               <div class="row">
27.                 <div class="col-2">
28.                   <ul class="nav flex-column">
29.                     <li class="nav-item">
30.                       <a href="/nuxt-10/" target="_self" class="nav-link">
31.                         Home
```

```
32.                       </a>
33.                     </li>
34.                     <li class="nav-item">
35.                       <a href="/nuxt-10/page1" target="_self" class="nav-link active nuxt-link-
    active">
36.                         Page 1
37.                       </a>
38.                     </li>
39.                     <li class="nav-item">
40.                       <a href="/nuxt-10/page2" target="_self" class="nav-link">
41.                         Page 2
42.                       </a>
43.                     </li>
44.                   </ul>
45.                 </div> <div class="col-10"><div role="alert" aria-live="polite" aria-
    atomic="true" class="alert alert-primary"> Page 1 -- result=3 </div></div>
46.               </div>
47.             </div>
48.           </div>
49.         </div>
50.       </div>
51.     </div>
52.   </div>
53.   <script>window.__NUXT__ = (function (a, b, c) {
54.   return {
55.     layout: "default", data: [{ result: 3 }], error: null, serverRendered: true,
56.     logs: [
57.       { date: new Date(1574939615256), args: ["[page1 asyncData started]"], type: a, level: b,
    tag: c },
58.       { date: new Date(1574939620263), args: ["[page1 asyncData finished]"], type: a, level:
    b, tag: c },
59.       { date: new Date(1574939620285), args: ["[page1 beforeCreate]"], type: a, level: b, tag:
    c },
60.       { date: new Date(1574939620287), args: ["[page1 created]"], type: a, level: b, tag: c }
61.     ]
62.   }
63.     }("log", 2, ""));</script>
64.   <script src="/nuxt-10/_nuxt/runtime.js" defer></script>
65.   <script src="/nuxt-10/_nuxt/commons.app.js" defer></script>
66.   <script src="/nuxt-10/_nuxt/vendors.app.js" defer></script>
67.   <script src="/nuxt-10/_nuxt/app.js" defer></script>
68. </body>
69. </html>
```

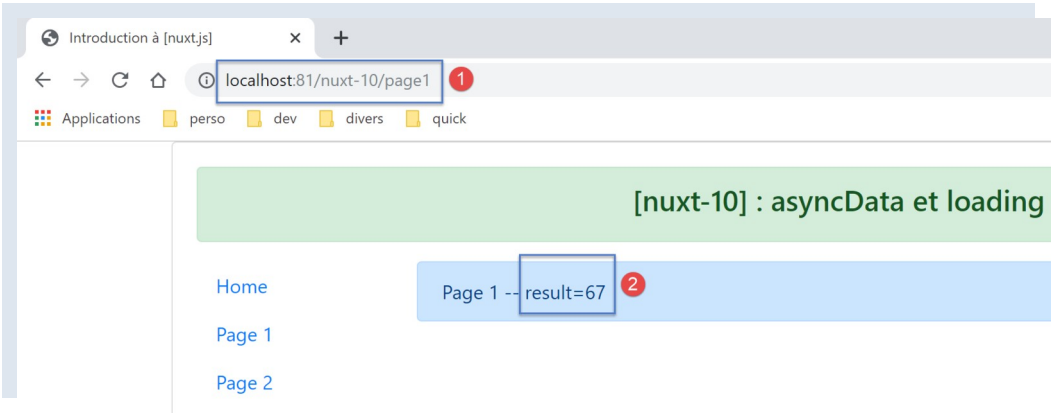
- ligne 55 : on voit que le serveur a envoyé au client un tableau [data] qui contient l'objet [result:3] qui a été intégré à l'objet [data] de la page [page1] du serveur. Afin que le client puisse faire de même et donc afficher la même page que le serveur, celui-ci lui transmet l'objet [result]. On rappelle que le client ne va pas exécuter la fonction [asyncData]. Il va simplement utiliser les données calculées par le serveur ;

Maintenant naviguons de la page [Home] à la page [Page 1] en utilisant le menu de navigation :

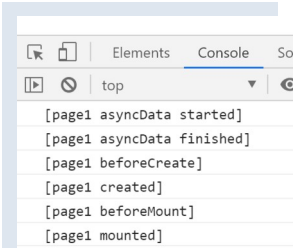


- en [1], on voit apparaître la barre de progression ;

Au bout de 5 secondes, on a la page [Page 1] :



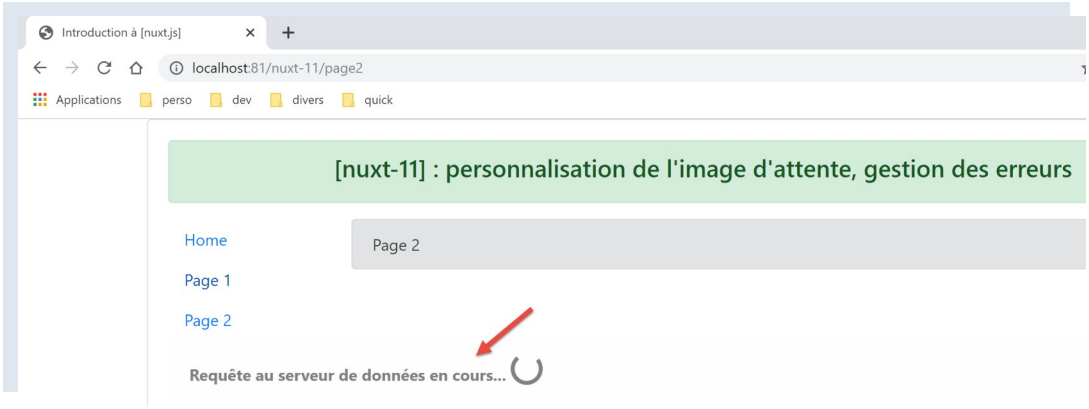
Les logs sont les suivants :



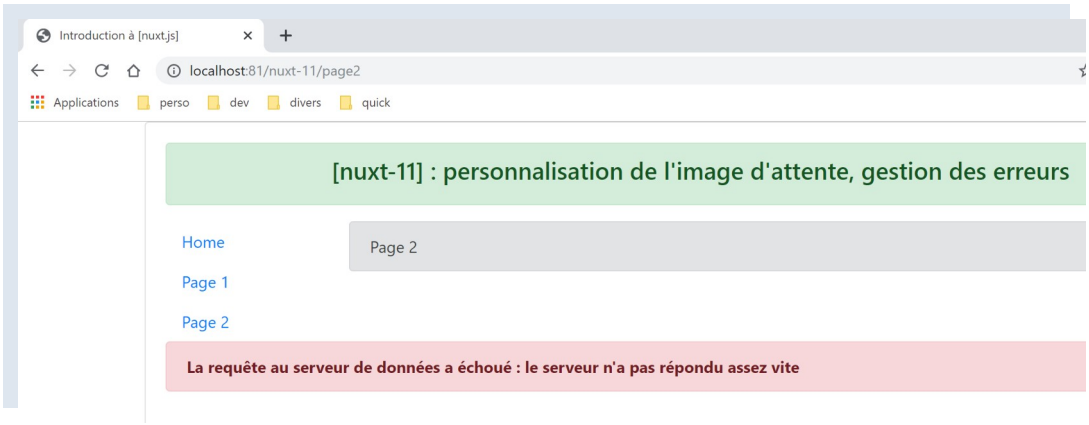
On voit que le client a exécuté la fonction [asyncData] avant le cycle de vie de la page.

14 Exemple [nuxt-11] : personnalisation de l'image d'attente

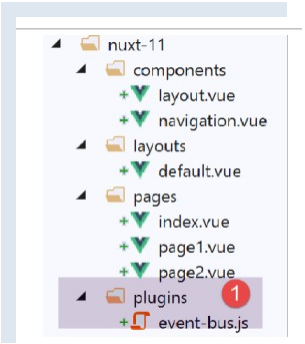
Par défaut, l'image d'attente de [nuxt] est une barre de progression. L'exemple [nuxt-11] montre qu'on peut la remplacer par sa propre image d'attente :



L'exemple [nuxt-11] montre également comment gérer des erreurs de chargement.



L'exemple [nuxt-11] est obtenu initialement par recopie de l'exemple [nuxt-10] :



Nous allons ajouter en [1], un plugin pour le client dont le rôle sera de gérer des événements entre composants.

14.1 Le plugin [event-bus]

Le plugin [event-bus] sera exécuté par le client et le serveur, mais on verra qu'il ne fonctionne pas côté serveur. Son code est le suivant :

```
1.  // on crée un bus d'événements entre les vues
2.  import Vue from 'vue'
3.  export default (context, inject) => {
4.    // le bus d'événements
5.    const eventBus = new Vue()
6.    // injection d'une fonction [eventBus] dans le contexte
7.    inject('eventBus', () => eventBus)
8.  }
```

- ligne 5 : le bus d'événements est une instance de la classe [Vue]. En effet, celle-ci a les méthodes pour gérer des événements :
  - [$emit] : pour émettre un événement ;
  - [$on] : pour se mettre à l'écoute d'un événement particulier ;

  Ce bus d'événements ne gèrera qu'un événement, [loading], qui sera utilisé par les pages pour démarrer / arrêter l'animation d'attente de la fin d'une fonction asynchrone ;
- ligne 7 : on crée une fonction [$eventBus] (1er argument) dont le rôle sera de rendre l'objet [eventBus] que l'on vient de créer (2ième argument). Cette fonction est injectée dans le contexte pour qu'elle soit disponible dans les objets [context.app] et l'objet [this] des pages ;

14.2 Le layout [default.vue]

Le layout [default.vue] évolue de la façon suivante :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <!-- un message -->
5.        <b-alert show variant="success" align="center">
6.          <h4>[nuxt-11] : personnalisation de l'attente, gestion des erreurs</h4>
7.        </b-alert>
8.        <!-- la vue courante du routage -->
9.        <nuxt />
10.       <!-- loading -->
11.       <b-alert v-if="showLoading" show variant="light">
12.         <strong>Requête au serveur de données en cours...</strong>
13.         <div class="spinner-border ml-auto" role="status" aria-hidden="true"></div>
14.       </b-alert>
15.       <!-- erreur de chargement -->
16.       <b-alert v-if="showErrorLoading" show variant="danger">
17.         <strong>La requête au serveur de données a échoué : {{ errorLoadingMessage }}</strong>
18.       </b-alert>
19.     </b-card>
20.   </div>
21. </template>
22.
23. <script>
24. /* eslint-disable no-console */
25. export default {
26.   name: 'App',
27.   data() {
28.     return {
29.       showLoading: false,
30.       showErrorLoading: false
31.     }
32.   },
33.   // cycle de vie
34.   beforeCreate() {
35.     console.log('[default beforeCreate]')
36.   },
37.   created() {
38.     console.log('[default created]')
39.     // on écoute l'évt [loading]
40.     this.$eventBus().$on('loading', this.mShowLoading)
41.     // ainsi que l'évt [errorLoadingMessage]
42.     this.$eventBus().$on('errorLoading', this.mShowErrorLoading)
43.   },
44.   beforeMount() {
45.     console.log('[default beforeMount]')
46.   },
```

```
47.   mounted() {
48.     console.log('[default mounted]')
49.   },
50.   methods: {
51.     // gestion du chargement
52.     mShowLoading(value) {
53.       console.log('[default mShowLoading], showLoading=', value)
54.       this.showLoading = value
55.     },
56.     // erreur de chargement
57.     mShowErrorLoading(value, errorLoadingMessage) {
58.       console.log('[default mShowErrorLoading], showErrorLoading=', value,
    'errorLoadingMessage=', errorLoadingMessage)
59.       this.showErrorLoading = value
60.       this.errorLoadingMessage = errorLoadingMessage
61.     }
62.   }
63. }
64. </script>
```

- lignes 11-14 : l'animation d'attente. Elle n'est affichée que si la propriété [showLoading] est vraie (ligne 29) ;
- lignes 16-18 : le message d'erreur du chargement. Il n'est affiché que si la propriété [showErrorLoading] (ligne 30) est vraie ;
- lignes 29-30 : au chargement initial du composant, l'animation d'attente est cachée ainsi que le message d'erreur ;
- lignes 37-43 : lorsqu'elle est créée, la page écoute l'événement [loading] (1er argument) sur le bus d'événements créé par le plugin. A sa réception, elle fait exécuter la méthode [mShowLoading] des lignes 52-55 (2ième argument) ;
- lignes 52-55 : la valeur reçue par la méthode [mShowLoading] sera un booléen true / false. Elle sert à montrer / cacher le message d'attente ;
- lignes 41-42 : lorsqu'elle est créée, la page écoute l'événement [errorLoading] (1er argument) sur le bus d'événements créé par le plugin. A sa réception, elle fait exécuter la méthode [mShowErrorLoading] des lignes 57-61 (2ième argument) ;
- ligne 57 : la méthode [mShowErrorLoading] reçoit deux arguments :
  - le 1er argument est un booléen true / false pour montrer / cacher le message d'erreur ;
  - le 2ième argument n'est présent que s'il y a eu erreur. Il représente le message d'erreur à afficher ;
- les logs des lignes 53 et 58 vont nous montrer que les méthodes [showLoading] et [showErrorLoading] **ne sont pas exécutées** côté serveur ;

14.3 La page [page1]

Le code de la page [page1] évolue de la façon suivante :

```
1.  <!-- vue n° 1 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="primary"> Page 1 -- result={{ result }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13. /* eslint-disable nuxt/no-timing-in-fetch-data */
14.
15. import Navigation from '@/components/navigation'
16. import Layout from '@/components/layout'
17.
18. export default {
19.   name: 'Page1',
20.   // composants utilisés
21.   components: {
22.     Layout,
23.     Navigation
24.   },
25.   // données asynchrones
26.   asyncData(context) {
```

```
27.     // log
28.     console.log('[page1 asyncData started]')
29.     // début attente
30.     context.app.$eventBus().$emit('loading', true)
31.     // pas d'erreur
32.     context.app.$eventBus().$emit('errorLoading', false)
33.     // on rend une promesse
34.     return new Promise(function(resolve, reject) {
35.       // on simule une fonction asynchrone
36.       setTimeout(function() {
37.         // fin attente
38.         context.app.$eventBus().$emit('loading', false)
39.         // log
40.         console.log('[page1 asyncData finished]')
41.         // on rend le résultat asynchrone - un nombre aléatoire ici
42.         resolve({ result: Math.floor(Math.random() * Math.floor(100)) })
43.       }, 5000)
44.     })
45.   },
46.
47.   // cycle de vie
48.   beforeCreate() {
49.     console.log('[page1 beforeCreate]')
50.   },
51.   created() {
52.     console.log('[page1 created]')
53.   },
54.   beforeMount() {
55.     console.log('[page1 beforeMount]')
56.   },
57.   mounted() {
58.     console.log('[page1 mounted]')
59.   }
60. }
61. </script>
```

- les modifications ont lieu dans la fonction [asyncData] des lignes 26-47 ;
- lignes 29-30 : avant que ne commence la fonction asynchrone on émet l'événement [loading] à destination des autres pages de l'application. On rappelle que dans [asyncData], on n'a pas accès à l'objet [this] pas encore créé. On utilise alors le contexte que reçoit en argument la fonction [asyncData] (ligne 26) ;
- ligne 30 : on utilise le bus d'événements pour indiquer que le chargement va commencer ;
- ligne 38 : on utilise le bus d'événements pour indiquer que le chargement est terminé ;

Note : A l'exécution, lorsque la page [page1] est demandée au serveur, on ne voit pas l'image d'attente. Dans les logs on voit que côté serveur la méthode [default.mShowLoading] n'est pas appelée. De toute façon, voir l'image d'attente n'a pas de sens lorsque la page est demandée au serveur. Celui-ci n'envoie la page au navigateur client qu'une fois la fonction [asyncData] terminée. L'image d'attente est alors inutile. Ce sera le cas pour toutes les pages de l'application demandées directement au serveur.

14.4 La page [index]

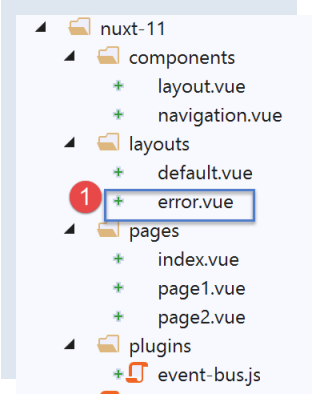
Le code de la page [index] est le suivant :

```
1.  <!-- page principale -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="warning">
8.        Home
9.      </b-alert>
10.   </Layout>
11. </template>
12.
13. <script>
14. /* eslint-disable no-undef */
15. /* eslint-disable no-console */
16. /* eslint-disable nuxt/no-env-in-hooks */
17. /* eslint-disable nuxt/no-timing-in-fetch-data */
18.
```

```
19. import Navigation from '@/components/navigation'
20. import Layout from '@/components/layout'
21.
22. export default {
23.   name: 'Home',
24.   // composants utilisés
25.   components: {
26.     Layout,
27.     Navigation
28.   },
29.   // données asynchrones
30.   asyncData(context) {
31.     // log
32.     console.log('[page1 asyncData started]')
33.     // début attente
34.     context.app.$eventBus().$emit('loading', true)
35.     // pas d'erreur
36.     context.app.$eventBus().$emit('errorLoading', false)
37.     // on rend une promesse
38.     return new Promise(function(resolve, reject) {
39.       // on simule une fonction asynchrone
40.       setTimeout(function() {
41.         // fin attente
42.         context.app.$eventBus().$emit('loading', false)
43.         // log
44.         console.log('[page1 asyncData finished]')
45.         // on rend une erreur
46.         reject(new Error("le serveur n'a pas répondu assez vite"))
47.       }, 5000)
48.     }).catch((e) => context.error({ statusCode: 500, message: e.message }))
49.   },
50.   // cycle de vie
51.   beforeCreate() {
52.     console.log('[home beforeCreate]')
53.   },
54.   created() {
55.     console.log('[home created]')
56.   },
57.   beforeMount() {
58.     console.log('[home beforeMount]')
59.   },
60.   mounted() {
61.     console.log('[home mounted]')
62.     // pas d'erreur
63.     this.$eventBus().$emit('errorLoading', false)
64.   }
65. }
66. </script>
```

- lignes 30-49 : la fonction [asyncData] est identique à celle de la page [page1] à un détail près : ligne 46, on termine la fonction asynchrone sur un échec (utilisation de la méthode [reject]) ;
- ligne 46 : le paramètre de la fonction [reject] est une instance de la classe [Error]. Le paramètre du constructeur [Error] est le message de l'erreur ;
- ligne 48 : cette erreur est interceptée par la méthode [catch] de la [Promise] qui reçoit l'erreur en paramètre. On utilise alors la fonction [context.error] pour déclarer l'erreur. Le paramètre de la fonction [context.error] est un objet avec ici deux propriétés :
  - [statusCode] : un code HTTP d'erreur ;
  - [message] : un message d'erreur ;

Que [asyncData] soit exécutée par le client ou le serveur, en cas d'erreur [context.error], [nuxt] affiche la page [layouts / **error.vue**] :



Bien que ce soit une page, la page [error.vue] est cherchée dans le dossier [layouts] (peut-être pour éviter qu'elle soit incluse dans les routes de l'application ?). Ici, la page [error.vue] est la suivante :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <!-- mise en page -->
4.    <Layout :left="true" :right="true">
5.      <!-- alerte dans la colonne de droite -->
6.      <template slot="right">
7.        <!-- message sur fond jaune -->
8.        <b-alert show variant="danger" align="center">
9.          <h4>L'erreur suivante s'est produite : {{ JSON.stringify(error) }}</h4>
10.       </b-alert>
11.     </template>
12.     <!-- menu de navigation dans la colonne de gauche -->
13.     <Navigation slot="left" />
14.   </Layout>
15. </template>
16.
17. <script>
18. /* eslint-disable no-undef */
19. /* eslint-disable no-console */
20. /* eslint-disable nuxt/no-env-in-hooks */
21.
22. import Layout from '@/components/layout'
23. import Navigation from '@/components/navigation'
24.
25. export default {
26.   name: 'Error',
27.   // composants utilisés
28.   components: {
29.     Layout,
30.     Navigation
31.   },
32.   // propriété [props]
33.   props: { error: { type: Object, default: () => 'waiting ...' } },
34.   // cycle de vie
35.   beforeCreate() {
36.     // client et serveur
37.     console.log('[error beforeCreate]')
38.   },
39.   created() {
40.     // client et serveur
41.     console.log('[error created, error=]', this.error)
42.   },
43.   beforeMount() {
44.     // client seulement
45.     console.log('[error beforeMount]')
46.   },
47.   mounted() {
48.     // client seulement
49.     console.log('[error mounted]')
50.   }
51. }
52. </script>
```

Lorsque [nuxt] affiche la page [error.vue], elle lui passe en propriété [props], l'erreur qui s'est produite (ligne 33). Si l'erreur a été provoquée par [context.error(**objet1**)], la propriété [props] de la page [error.vue] aura la valeur [**objet1**]. La documentation [nuxt] indique que [objet1] doit avoir au moins les attributs [statusCode, message]. La ligne 9 affiche la chaîne jSON de l'objet [objet1] reçu.

14.5 La page [page2]

La page [page2] montre une autre façon de gérer l'erreur :

- dans [page1], l'erreur est affichée dans une page à part [error.vue] ;
- dans [page2], l'erreur sera affichée dans la page [page2] qui a provoqué l'erreur ;

Le code de [page2] est le suivant :

```
1.  <!-- vue n° 2 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-alert slot="right" show variant="secondary">
8.        Page 2
9.      </b-alert>
10.   </Layout>
11. </template>
12.
13. <script>
14. /* eslint-disable no-console */
15. /* eslint-disable nuxt/no-timing-in-fetch-data */
16.
17. import Navigation from '@/components/navigation'
18. import Layout from '@/components/layout'
19.
20. export default {
21.   name: 'Page2',
22.   // composants utilisés
23.   components: {
24.     Layout,
25.     Navigation
26.   },
27.   // données asynchrones
28.   asyncData(context) {
29.     // log
30.     console.log('[page2 asyncData started]')
31.     // début attente
32.     context.app.$eventBus().$emit('loading', true)
33.     // pas d'erreur
34.     context.app.$eventBus().$emit('errorLoading', false)
35.     // on rend une promesse
36.     return new Promise(function(resolve, reject) {
37.       // on simule une fonction asynchrone
38.       setTimeout(function() {
39.         // fin attente
40.         context.app.$eventBus().$emit('loading', false)
41.         // on génére arbitrairement une erreur
42.         const errorLoadingMessage = "le serveur n'a pas répondu assez vite"
43.         // fin avec succès
44.         resolve({ showErrorLoading: true, errorLoadingMessage })
45.         // log
46.         console.log('[page2 asyncData finished]')
47.       }, 5000)
48.     })
49.   },
50.   // cycle de vie
51.   beforeCreate() {
52.     console.log('[page2 beforeCreate]')
53.   },
54.   created() {
55.     console.log('[page2 created]')
56.   },
```

```
57.   beforeMount() {
58.     console.log('[page2 beforeMount]')
59.   },
60.   mounted() {
61.     console.log('[page2 mounted]')
62.     // client
63.     if (this.showErrorLoading) {
64.       console.log('[page2 mounted, showErrorLoading=true]')
65.       this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
66.     }
67.   }
68. }
69. </script>
```

De nouveau, on insère une fonction [asyncData] dans le code de la page et comme [index], [page2] va générer une erreur qu'on va cette fois gérer différemment.

- ligne 44 : le serveur comme le client terminent la promesse sur un succès en rendant le résultat [`{ showErrorLoading: true, errorLoadingMessage }`]. On sait que cela va avoir pour effet d'inclure les propriétés [showerrorLoading, errorLoadingMessage] dans les propriétés [data] de la page et que le client va recevoir ces propriétés ;
- lignes 60-67 : on sait que la fonction [mounted] n'est exécutée que par le client ;
- ligne 63 : le client teste si la propriété [`showErrorLoading`] a été positionnée (par le serveur ou le client selon les cas). Si oui, il émet l'événement ['errorLoading'] (ligne 65) pour que la page [default] affiche le message d'erreur [`this.errorLoadingMessage`]. Au final, le serveur envoie une page **sans message d'erreur affiché**. Celui-ci est affiché au dernier moment par le client lorsque la page est 'montée' ;

14.6 Exécution

14.6.1 [nuxt.config]

Le fichier [nuxt.config.js] d'exécution est le suivant :

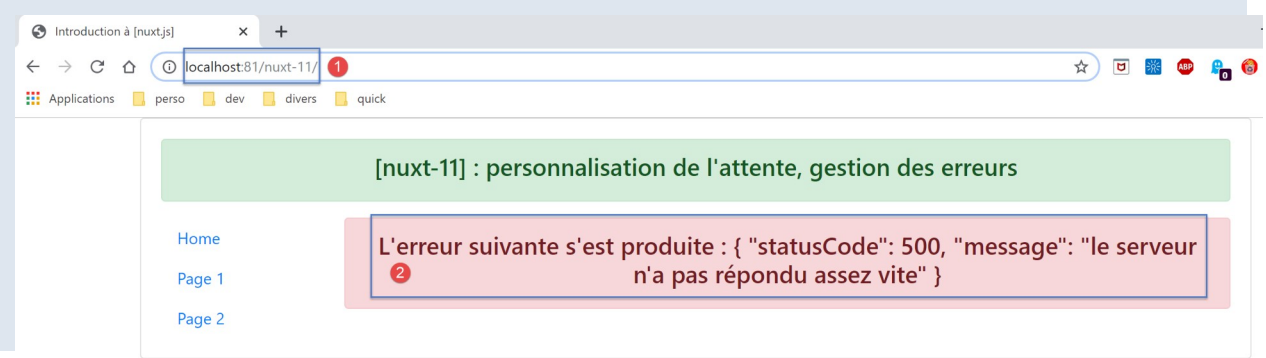
```
1.  export default {
2.    mode: 'universal',
3.    /*
4.     ** Headers of the page
5.     */
6.    head: {
7.      title: 'Introduction à [nuxt.js]',
8.      meta: [
9.        { charset: 'utf-8' },
10.       { name: 'viewport', content: 'width=device-width, initial-scale=1' },
11.       {
12.         hid: 'description',
13.         name: 'description',
14.         content: 'ssr routing loading asyncdata middleware plugins store'
15.       }
16.     ],
17.     link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
18.   },
19.   /*
20.    ** Customize the progress-bar color
21.    */
22.   loading: false,
23.
24.   /*
25.    ** Global CSS
26.    */
27.   css: [],
28.   /*
29.    ** Plugins to load before mounting the App
30.    */
31.   plugins: [{ src: '@/plugins/event-bus' }],
32.   /*
33.    ** Nuxt.js dev-modules
34.    */
35.   buildModules: [
36.     // Doc: https://github.com/nuxt-community/eslint-module
37.     '@nuxtjs/eslint-module'
```

```
38.   ],
39.   /*
40.    ** Nuxt.js modules
41.    */
42.   modules: [
43.     // Doc: https://bootstrap-vue.js.org
44.     'bootstrap-vue/nuxt',
45.     // Doc: https://axios.nuxtjs.org/usage
46.     '@nuxtjs/axios'
47.   ],
48.   /*
49.    ** Axios module configuration
50.    ** See https://axios.nuxtjs.org/options
51.    */
52.   axios: {},
53.   /*
54.    ** Build configuration
55.    */
56.   build: {
57.     /*
58.      ** You can extend webpack config here
59.      */
60.     extend(config, ctx) {}
61.   },
62.   // répertoire du code source
63.   srcDir: 'nuxt-11',
64.   // routeur
65.   router: {
66.     // racine des URL de l'application
67.     base: '/nuxt-11/'
68.   },
69.   // serveur
70.   server: {
71.     // port de service, 3000 par défaut
72.     port: 81,
73.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
74.     // 0.0.0.0 = toutes les adresses réseau de la machine
75.     host: 'localhost'
76.   }
77. }
```

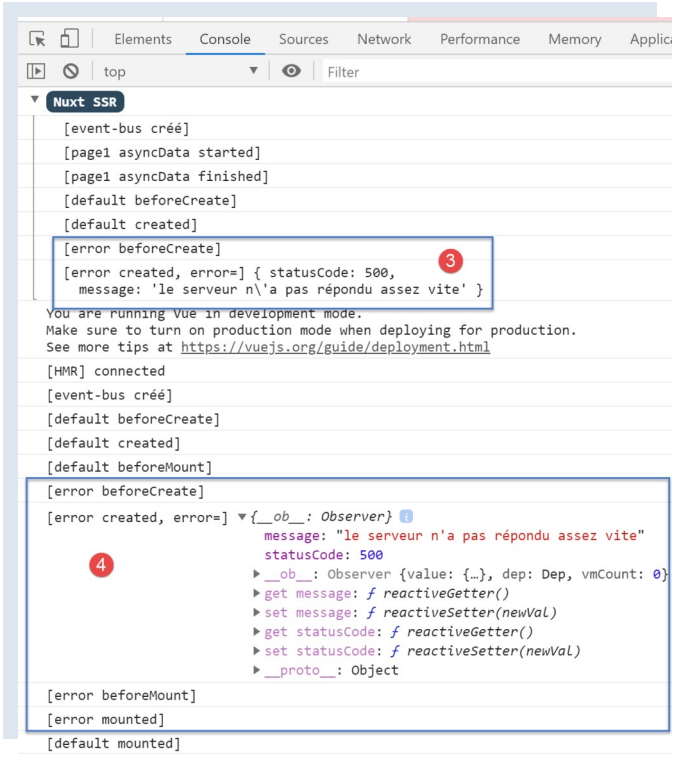
- ligne 22 : on met la propiété [loading] à [false] pour que [nuxt] n'utilise pas son image d'attente par défaut ;
- ligne 31 : le plugin qui définit le bus d'événements ;

14.6.2 *La* page [index] *exécutée par le* serveur

Demandons la page [index] au serveur (on tape l'URL [http://localhost:81/nuxt-11/] à la main). La page affichée par le navigateur client est la suivante :



Les logs sont les suivants :



- en [3], on voit que le serveur envoie la page [error.vue] ;
- en [4], on voit que le client affiche lui aussi la page [error] avec la même erreur que le serveur ;
- on peut remarquer que la méthode [mShowLoading] de la page [default] n'a pas été appelée côté serveur alors même que la page [index] avait activé une attente. Cette méthode est appelée à réception d'un événement et visiblement la gestion événementielle n'est pas implémentée côté serveur ;

Examinons le code source de la page reçue par le navigateur client :

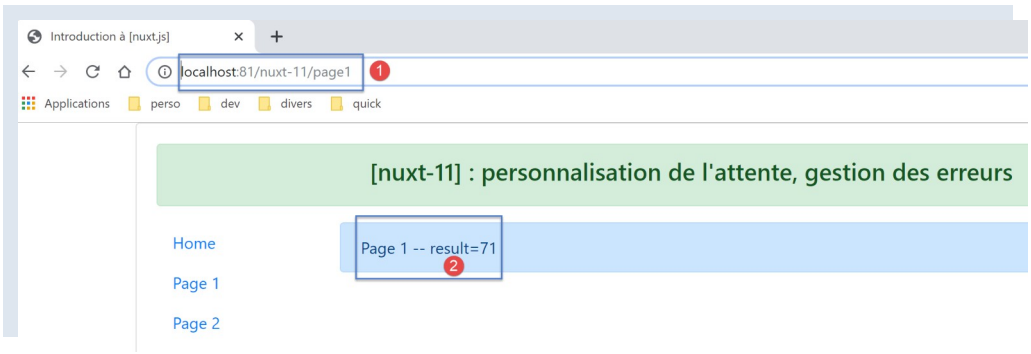
```
1.  <!doctype html>
2.  <html data-n-head-ssr>
3.  <head>
4.    <title>Introduction à [nuxt.js]</title>
5.    <meta data-n-head="ssr" charset="utf-8">
6.    <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
7.    <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing
    loading asyncdata middleware plugins store">
8.    <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
9.    <base href="/nuxt-11/">
10.   <link rel="preload" href="/nuxt-11/_nuxt/runtime.js" as="script">
11.   <link rel="preload" href="/nuxt-11/_nuxt/commons.app.js" as="script">
12.   <link rel="preload" href="/nuxt-11/_nuxt/vendors.app.js" as="script">
13.   <link rel="preload" href="/nuxt-11/_nuxt/app.js" as="script">
14.   ...
15. </head>
16. <body>
17.   <div data-server-rendered="true" id="__nuxt">
18.     <div id="__layout">
19.       <div class="container">
20.         <div class="card">
21.           <div class="card-body">
22.             <div role="alert" aria-live="polite" aria-atomic="true" align="center" class="alert
    alert-success">
23.             <h4>[nuxt-11] : personnalisation de l'attente, gestion des erreurs</h4>
24.               </div>
25.             <div>
26.               <div class="row">
27.                 <div class="col-2">
28.                   <ul class="nav flex-column">
```

```
29.                     <li class="nav-item">
30.                       <a href="/nuxt-11/" target="_self" class="nav-link active nuxt-link-
    active">
31.                         Home
32.                       </a>
33.                     </li>
34.                     <li class="nav-item">
35.                       <a href="/nuxt-11/page1" target="_self" class="nav-link">
36.                         Page 1
37.                       </a>
38.                     </li>
39.                     <li class="nav-item">
40.                       <a href="/nuxt-11/page2" target="_self" class="nav-link">
41.                         Page 2
42.                       </a>
43.                     </li>
44.                   </ul>
45.                 </div> <div class="col-10"><div role="alert" aria-live="polite" aria-
    atomic="true" align="center" class="alert alert-danger">
46.                   <h4>L'erreur suivante s'est produite :
    {&amp;quot;statusCode&amp;quot;:500,&amp;quot;message&amp;quot;:&amp;quot;le serveur n'a pas répondu assez
    vite&amp;quot;}</h4>
47.                     </div>
48.                   </div>
49.                 </div>
50.               </div>
51.             </div>
52.           </div>
53.         </div>
54.       </div>
55.   </div>
56.   <script>window.__NUXT__ = (function (a, b, c, d) {
57.   d.statusCode = 500; d.message = "le serveur n'a pas répondu assez vite";
58.   return {
59.     layout: "default", data: [d], error: d, serverRendered: true,
60.     logs: [
61.       { date: new Date(1575047424168), args: ["[event-bus créé]"], type: a, level: b, tag:
    c },
62.       { date: new Date(1575047424175), args: ["[page1 asyncData started]"], type: a, level: b,
    tag: c },
63.       { date: new Date(1575047429455), args: ["[page1 asyncData finished]"], type: a, level:
    b, tag: c },
64.       { date: new Date(1575047429515), args: ["[default beforeCreate]"], type: a, level: b,
    tag: c },
65.       { date: new Date(1575047429675), args: ["[default created]"], type: a, level: b, tag:
    c },
66.       { date: new Date(1575047430157), args: ["[error beforeCreate]"], type: a, level: b, tag:
    c },
67.       { date: new Date(1575047430246), args: ["[error created, error=]", "{ statusCode: 500,\n
    message: 'le serveur n\\'a pas répondu assez vite' }"], type: a, level: b, tag: c }]
68.   }
69.     }("log", 2, "", {}));</script>
70.   <script src="/nuxt-11/_nuxt/runtime.js" defer></script>
71.   <script src="/nuxt-11/_nuxt/commons.app.js" defer></script>
72.   <script src="/nuxt-11/_nuxt/vendors.app.js" defer></script>
73.   <script src="/nuxt-11/_nuxt/app.js" defer></script>
74. </body>
75. </html>
```

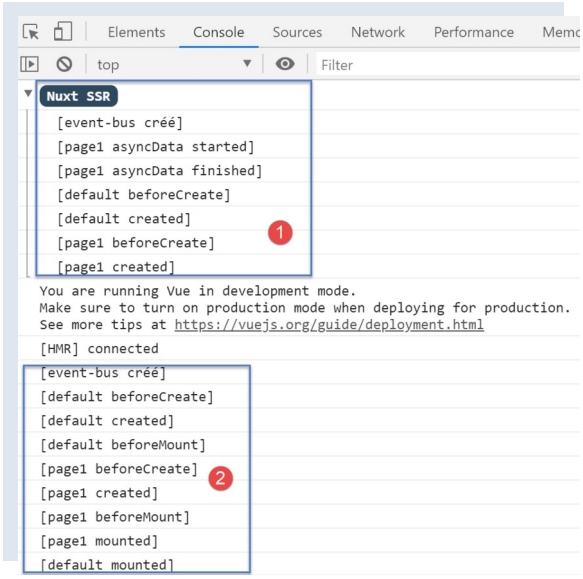
- ligne 57 : on voit que le serveur a envoyé un objet [d] qui représente l'erreur qui s'est produite côté serveur ;
- ligne 59 : on voit une propriété [error] ayant pour valeur l'objet [d]. On peut imaginer que c'est la présence de la propriété [error] dans la page envoyée par le serveur qui fait que les scripts client vont afficher la page [error.vue] avec l'erreur [error] ;

14.6.3 La page [page1] exécutée par le serveur

On tape à la main l'URL [http://localhost:81/nuxt-11/page1]. Au bout de 5 secondes, le navigateur affiche la page suivante :



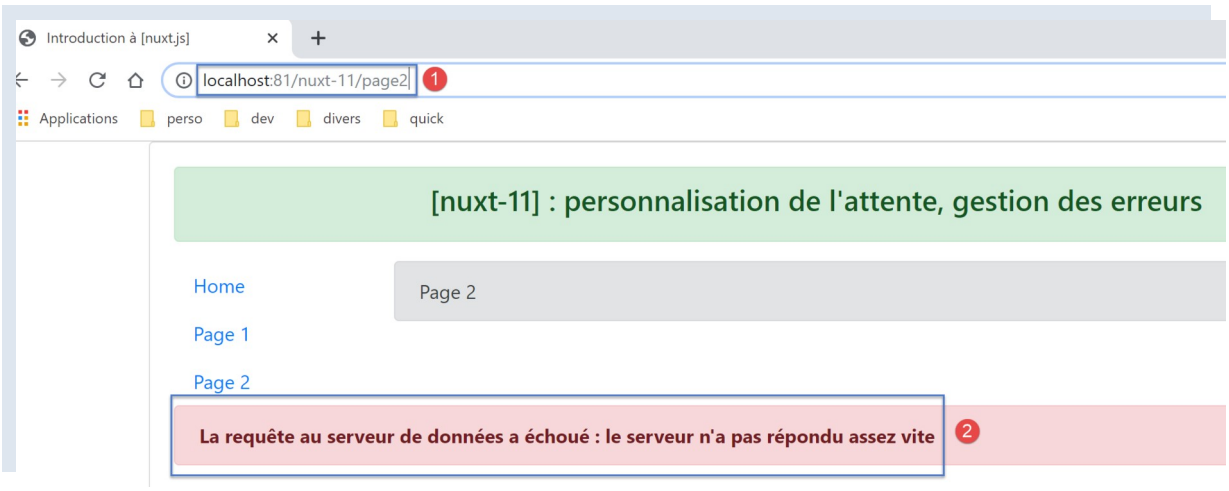
Les logs affichés sont les suivants :



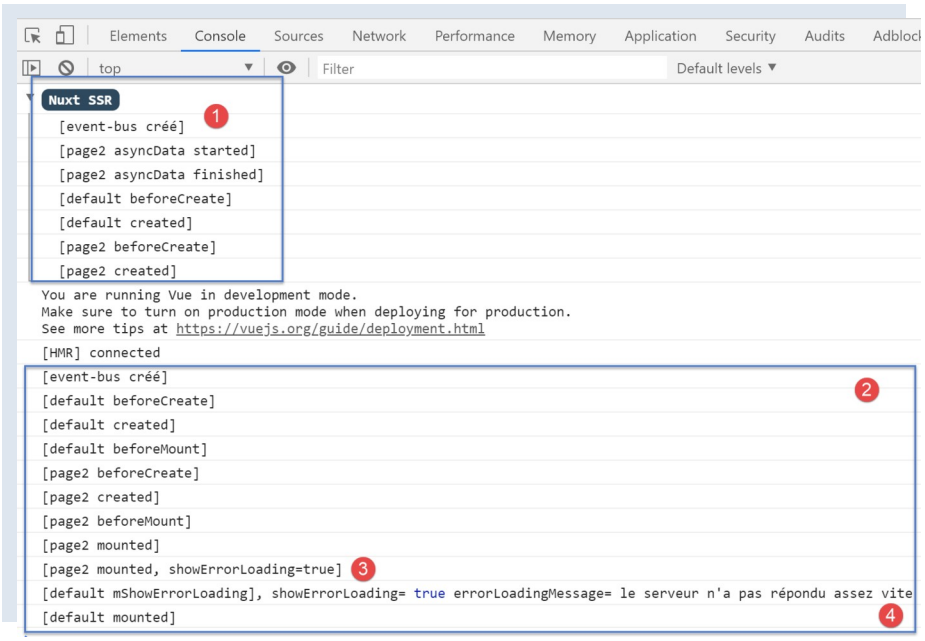
- en [1], les logsdu serveur. On peut remarquer que la méthode [mShowLoading] de la page [default] n'a pas été appelée ;
- en [2], les logs du client ;

14.6.4 La page [page2] exécutée par le serveur

Nous tapons à la main l'URL [http://localhost:81/nuxt-11/page2]. Au bout de 5 secondes, le navigateur affiche la page suivante :



Examinons les logs affichés dans le navigateur :

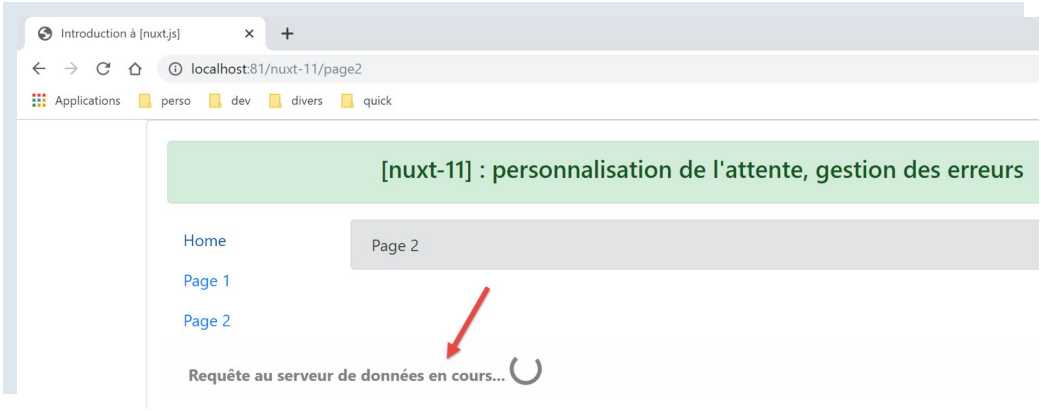


- en [1], les logs du serveur. On rappelle que le serveur a mis les propriétés [showErrorLoading, errorLoadingMessage] dans la page envoyée au navigateur client. On sait qu'alors ces propriétés vont être intégrées dans les [data] de la page affichée par le client
- en [3], lorsque la page [page2] est montée, elle trouve la propriété [showErrorLoading] à vrai. Elle envoie alors un événement à la page [default], pour qu'elle affiche le message d'erreur envoyé par le serveur [4] ;

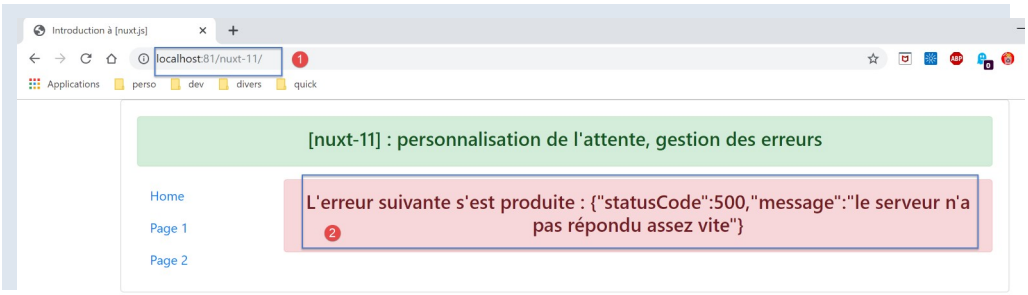
14.6.5 La page [index] exécutée par le client

On utilise maintenant les liens de navigation pour afficher les trois pages. Toutes les pages affichées par le client sont identiques à celles affichées par le serveur. La seule différence est que l'image d'attente de la fin des 5 secondes est affichée à chaque fois.

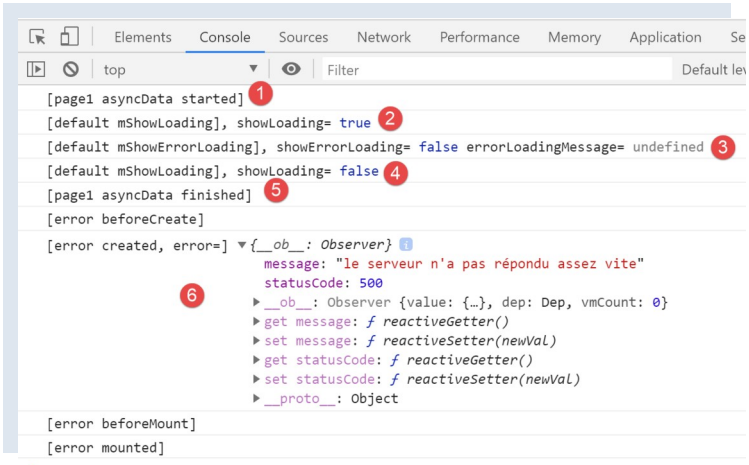
On commence par la page [index]. L'image d'attente est alors affichée :



puis au bout de 5 secondes, on obtient la page suivante :



La page finale est donc identique à celle obtenue côté serveur.

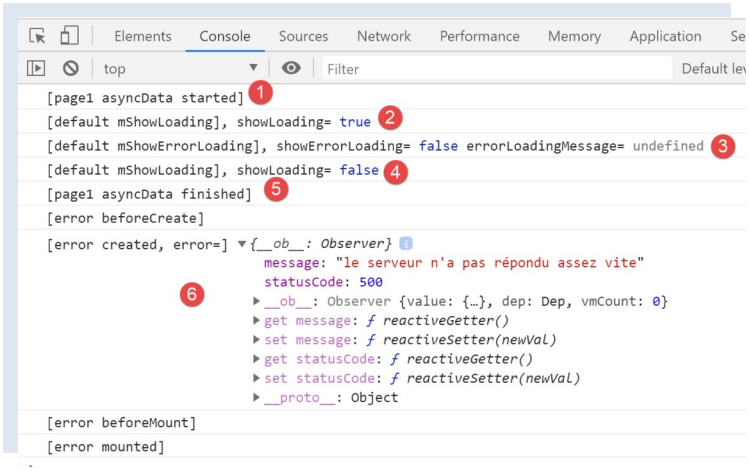


Rappelons la fonction [asyncData] de la page [index] :

```
1.  asyncData(context) {
2.      // log
3.      console.log('[page1 asyncData started]')
4.      // début attente
5.      context.app.$eventBus().$emit('loading', true)
6.      // pas d'erreur
7.      context.app.$eventBus().$emit('errorLoading', false)
8.      // on rend une promesse
9.      return new Promise(function(resolve, reject) {
10.       // on simule une fonction asynchrone
11.       setTimeout(function() {
```

```
12.         // fin attente
13.         context.app.$eventBus().$emit('loading', false)
14.         // log
15.         console.log('[page1 asyncData finished]')
16.         // on rend une erreur
17.         reject(new Error("le serveur n'a pas répondu assez vite"))
18.       }, 5000)
19.     }).catch((e) => context.error({ statusCode: 500, message: e.message }))
20. }
```

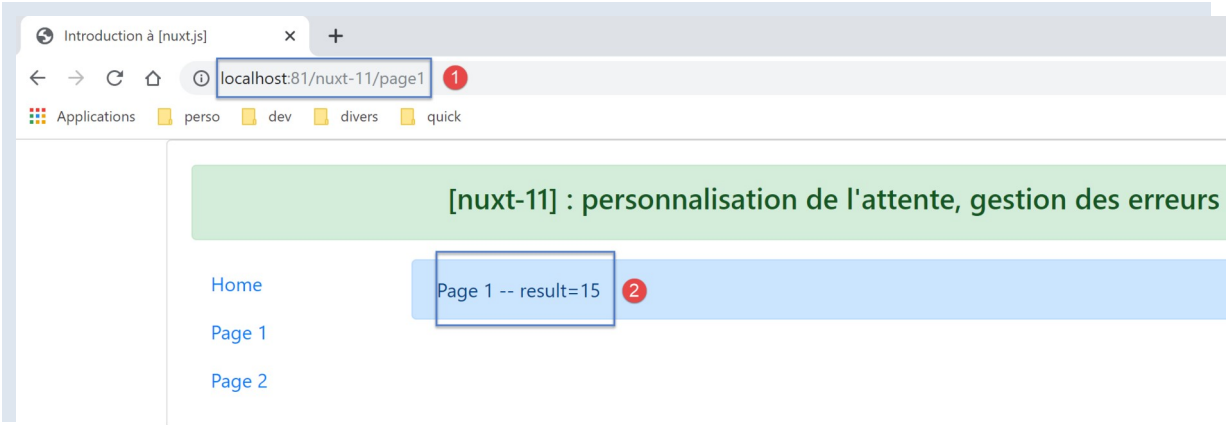
Les logs du client sont les suivants :



- en [1], la fonction [asyncData] démarre ;
- en [2], on met en route l'image d'attente ;
- en [2-3], on voit que la page [default] a reçu les événements [loading, true] [2] et [errorLoading, false] envoyés par la fonction [asyncData] de la page [index] (lignes 5 et 7);
- en [4], fin de l'attente. La page [default] a reçu l'événement [loading, false] envoyé par la page [index] (ligne 13);
- en [5], la fonction [asyncData] a terminé son travail ;
- parce que la fonction [asyncData] a créé une erreur avec [context.error] (ligne 19), la page [error] est affichée [6];

14.6.6 La page [*page1*] exécutée par le client

Après l'attente de 5 secondes, le client affiche la page suivante :

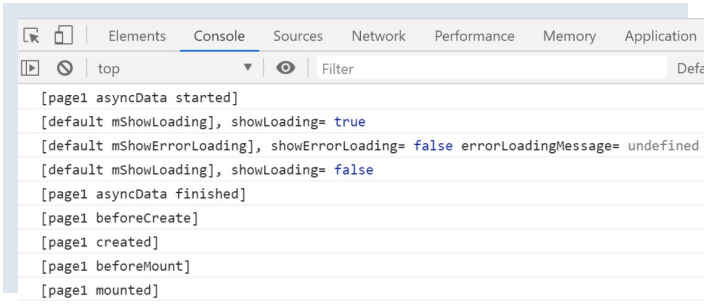


Rappelons le code de la fonction [asyncData] de [page1] :

```
1.  asyncData(context) {
```

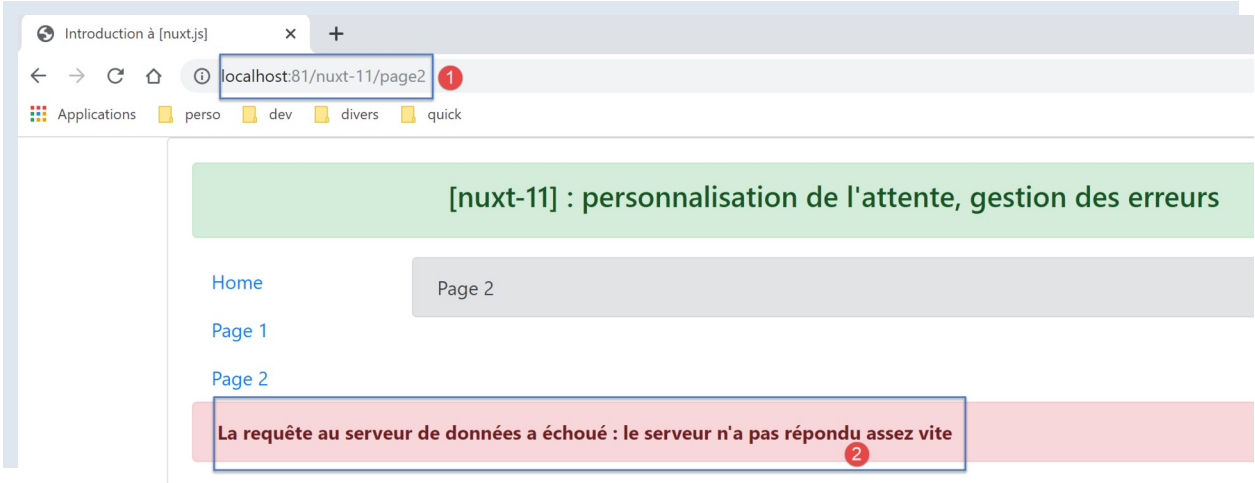
```
2.      // log
3.      console.log('[page1 asyncData started]')
4.      // début attente
5.      context.app.$eventBus().$emit('loading', true)
6.      // pas d'erreur
7.      context.app.$eventBus().$emit('errorLoading', false)
8.      // on rend une promesse
9.      return new Promise(function(resolve, reject) {
10.       // on simule une fonction asynchrone
11.       setTimeout(function() {
12.         // fin attente
13.         context.app.$eventBus().$emit('loading', false)
14.         // log
15.         console.log('[page1 asyncData finished]')
16.         // on rend le résultat asynchrone - un nombre aléatoire ici
17.         resolve({ result: Math.floor(Math.random() * Math.floor(100)) })
18.       }, 5000)
19.     })
20. },
```

Les logs sont les suivants :



14.6.7 La page [*page2*] exécutée par le client

Après l'attente de 5 secondes, le client affiche la page suivante :

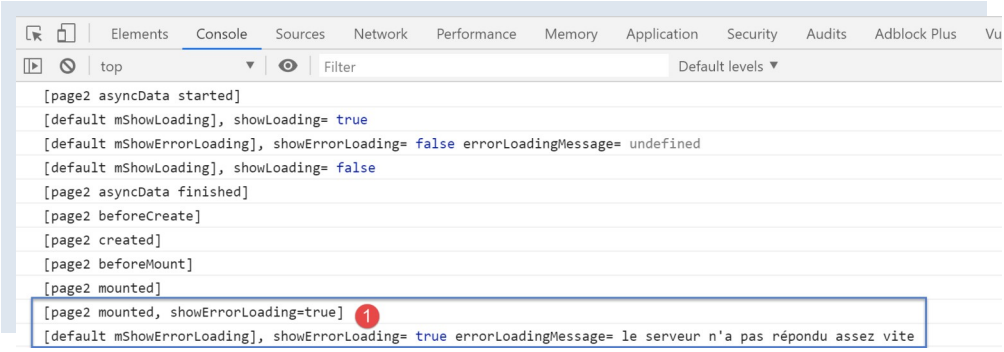


Rappelons le code des fonctions [asyncData] et [mounted] de [page2] :

```
1.  asyncData(context) {
2.      // log
3.      console.log('[page2 asyncData started]')
4.      // début attente
5.      context.app.$eventBus().$emit('loading', true)
6.      // pas d'erreur
7.      context.app.$eventBus().$emit('errorLoading', false)
```

```
8.      // on rend une promesse
9.      return new Promise(function(resolve, reject) {
10.       // on simule une fonction asynchrone
11.       setTimeout(function() {
12.         // fin attente
13.         context.app.$eventBus().$emit('loading', false)
14.         // on génére arbitrairement une erreur
15.         const errorLoadingMessage = "le serveur n'a pas répondu assez vite"
16.         // fin avec succès
17.         resolve({ showErrorLoading: true, errorLoadingMessage })
18.         // log
19.         console.log('[page2 asyncData finished]')
20.       }, 5000)
21.     })
22.   }
23.
24. mounted() {
25.     console.log('[page2 mounted]')
26.     // client
27.     if (this.showErrorLoading) {
28.       console.log('[page2 mounted, showErrorLoading=true]')
29.       this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
30.     }
31. }
```

Les logs sont les suivants :



- en [1], la page [default] a reçu l'événement [showErrorLoading, true] envoyé par [page2] (ligne 29) qui lui demande d'afficher le message d'erreur ;

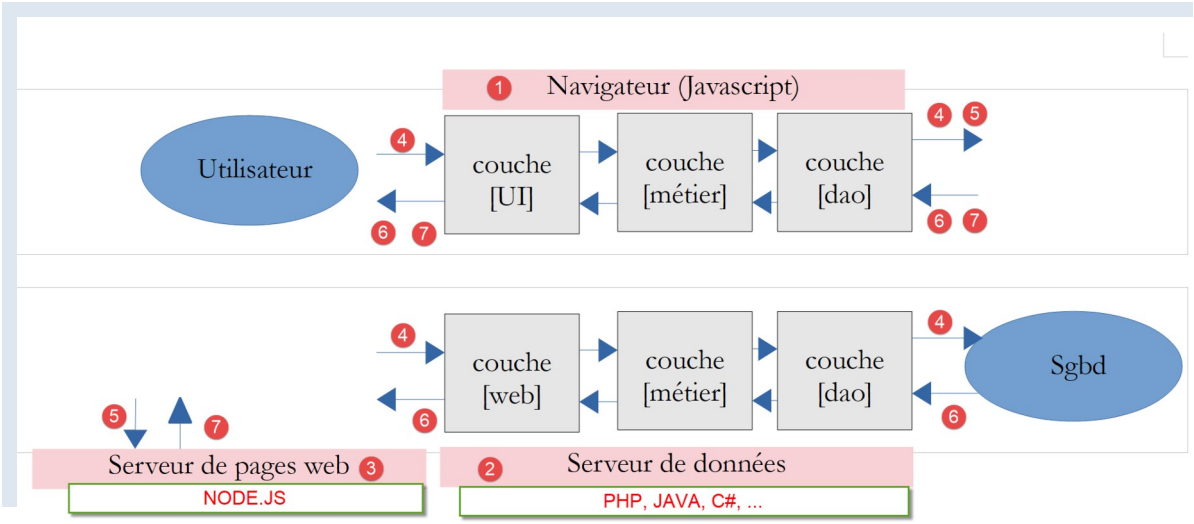
15 Exemple [nuxt-12] : requêtes HTTP avec axios

15.1 Présentation

Dans ce nouvel exemple, nous allons découvrir comment dans les fonctions [asyncData] on peut faire des requêtes HTTP avec la bibliothèque [axios]. Par ailleurs, nous allons utiliser des notions déjà acquises :

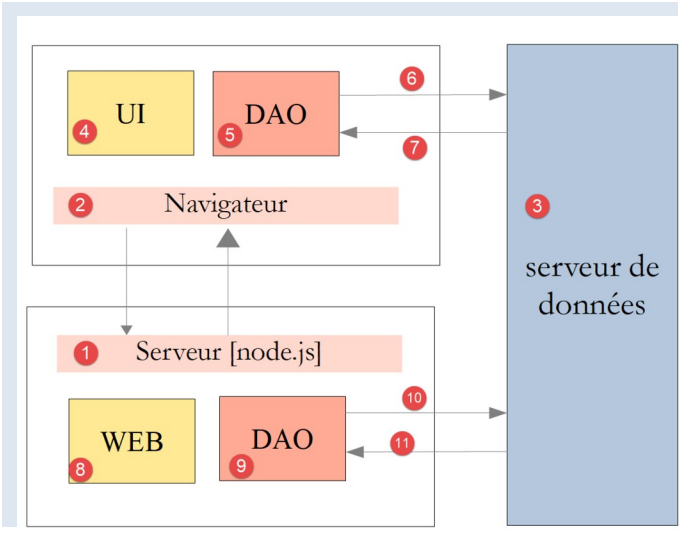
- l'utilisation de plugins de l'exemple [nuxt-06] :
- la persistance du store dans un cookie de session de l'exemple [nuxt-06] ;
- le contrôle de la navigation avec des middlewares de l'exemple [nuxt-09] ;
- la gestion des erreurs de l'exemple [nuxt-11] ;

L'architecture de l'exemple sera le suivant :



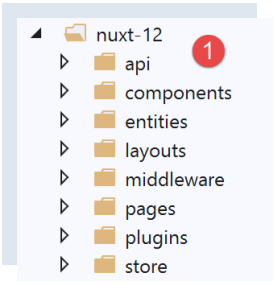
- l'application [nuxt] sera stockée sur le serveur [node.js] [3], téléchargée par le navigateur [1] qui l'exécutera ensuite ;
- le client [nuxt] [1] aussi bien que le serveur [nuxt] [3] feront des requêtes HTTP au serveur de données [2]. Ce serveur sera le serveur de calcul de l'impôt développé dans la partie PHP 7. Nous utiliserons sa dernière version, la version 14, avec les requêtes CORS autorisées ;

L'architecture de l'exemple peut être simplifiée de la façon suivante :



- en [1], le serveur [node.js] délivre les pages [nuxt] au navigateur [2]. C'est la couche [web] [8] du serveur qui délivre ces pages. Pour délivrer la page, le serveur a pu demander des données externes au serveur de données [3]. C'est la couche [DAO] [9] qui fait les requêtes HTTP nécessaires ;
- à chaque appel de page au serveur [node.js][1], le navigateur [2] reçoit la totalité de l'application [nuxt] qui va alors s'exécuter en mode SPA. Le bloc [UI] (User Interface) [4] présente des pages [vue.js] à l'utilisateur. Les actions de celui-ci ou le cycle de vie naturel des pages peuvent provoquer des appels de données externes au serveur de données [3]. C'est la couche [DAO] [5] qui fait alors les requêtes HTTP nécessaires ;

15.2 Arborescence du projet



15.3 Le fichier de configuration [nuxt.config.js]

Le projet sera contrôlé par le fichier [nuxt.config.js] suivant :

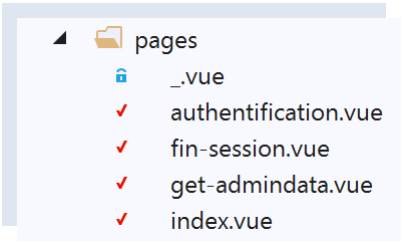
```
1.  export default {
2.    mode: 'universal',
3.    /*
4.     ** Headers of the page
5.     */
6.    head: {
7.      title: 'Introduction à [nuxt.js]',
8.      meta: [
9.        { charset: 'utf-8' },
10.       { name: 'viewport', content: 'width=device-width, initial-scale=1' },
11.       {
12.         hid: 'description',
13.         name: 'description',
14.         content: 'ssr routing loading asyncdata middleware plugins store'
15.       }
16.     ],
17.     link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
18.   },
19.   /*
20.    ** Customize the progress-bar color
21.    */
22.   loading: false,
23.
24.   /*
25.    ** Global CSS
26.    */
27.   css: [],
28.   /*
29.    ** Plugins to load before mounting the App
30.    */
31.   plugins: [
32.     { src: '@/plugins/client/plgSession', mode: 'client' },
33.     { src: '@/plugins/server/plgSession', mode: 'server' },
34.     { src: '@/plugins/client/plgDao', mode: 'client' },
35.     { src: '@/plugins/server/plgDao', mode: 'server' },
36.     { src: '@/plugins/client/plgEventBus', mode: 'client' }
37.   ],
38.   /*
39.    ** Nuxt.js dev-modules
```

```
40.    */
41.   buildModules: [
42.     // Doc: https://github.com/nuxt-community/eslint-module
43.     '@nuxtjs/eslint-module'
44.   ],
45.   /*
46.    ** Nuxt.js modules
47.    */
48.   modules: [
49.     // Doc: https://bootstrap-vue.js.org
50.     'bootstrap-vue/nuxt',
51.     // Doc: https://axios.nuxtjs.org/usage
52.     '@nuxtjs/axios',
53.     // https://www.npmjs.com/package/cookie-universal-nuxt
54.     'cookie-universal-nuxt'
55.   ],
56.   /*
57.    ** Axios module configuration
58.    ** See https://axios.nuxtjs.org/options
59.    */
60.   axios: {},
61.   /*
62.    ** Build configuration
63.    */
64.   build: {
65.     /*
66.      ** You can extend webpack config here
67.      */
68.     extend(config, ctx) { }
69.   },
70.   // répertoire du code source
71.   srcDir: 'nuxt-12',
72.   // routeur
73.   router: {
74.     // racine des URL de l'application
75.     base: '/nuxt-12/',
76.     // middleware de routage
77.     middleware: ['routing']
78.   },
79.   // serveur
80.   server: {
81.     // port de service, 3000 par défaut
82.     port: 81,
83.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
84.     // 0.0.0.0 = toutes les adresses réseau de la machine
85.     host: 'localhost'
86.   },
87.   // environnement
88.   env: {
89.     // configuration axios
90.     timeout: 2000,
91.     withCredentials: true,
92.     baseURL: 'http://localhost/php7/scripts-web/impots/version-14',
93.     // configuration du cookie de session [nuxt]
94.     maxAge: 60 * 5
95.   }
96. }
```

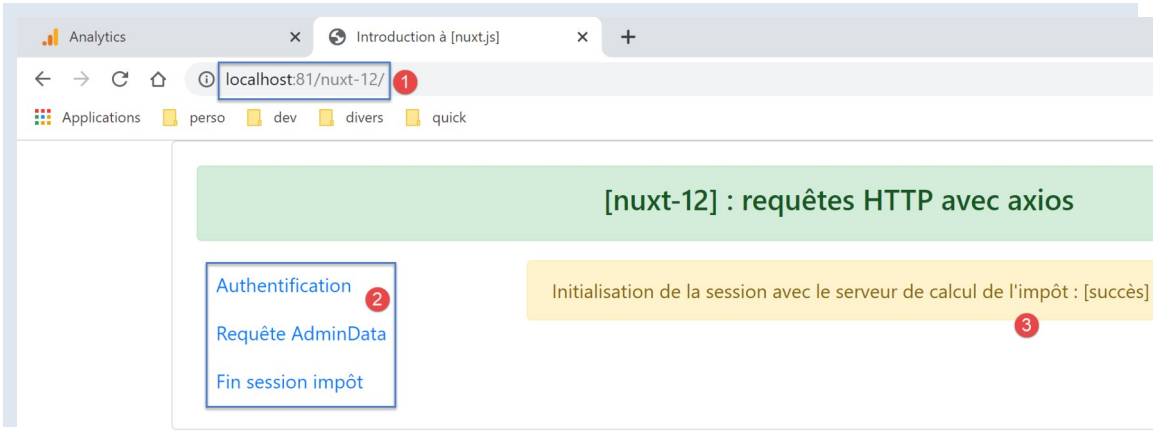
- ligne 22 : nous gérons nous-mêmes l'alerte d'attente de fin d'une action asynchrone ;
- ligne 31 : nous allons utiliser divers plugins qui seront spécialisés soit pour le client soit pour le serveur mais pas pour les deux à la fois ;
- ligne 52 : le module [axios] est intégré à [nuxt]. Cela va avoir pour conséquence que l'objet [axios] qui fera les requêtes HTTP de l'application [nuxt] vers le serveur PHP de calcul de l'impôt sera disponible dans [**context.$axios**] ;
- ligne 54 : le module [cookie-universal-nuxt] va nous permettre de sauvegarder la session [nuxt] dans un cookie ;
- ligne 60 : la propriété [axios] nous permet de configurer le module [`@nuxtjs/axios`] de la ligne 52. Nous n'utiliserons pas cette possibilité à laquelle nous préfèrerons la propriété [env] de la ligne 88 ;
- ligne 90 : durée d'attente maximale de la réponse du serveur de calcul de l'impôt ;
- ligne 91 : nécessaire au client [nuxt] - autorise l'utilisation de cookies dans les échanges avec le serveur de calcul de l'impôt ;

- ligne 92 : l'URL de base du serveur de calcul de l'impôt ;
- ligne 94 : durée de vie de la session nuxt (5 mn) ;
- ligne 77 : la navigation des client et serveur [nuxt] sera contrôlée par un middleware de routage ;

15.4 La couche [UI] de l'application



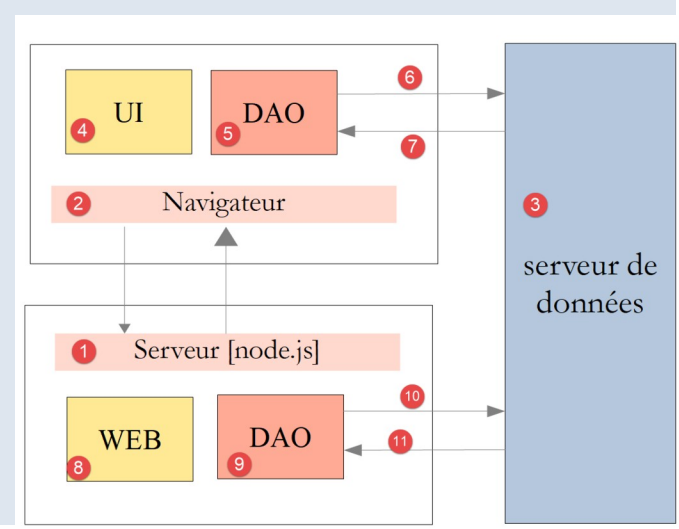
Nous allons donner à l'application [nuxt] l'accès à l'API du serveur de calcul de l'impôt via la vue suivante :



- en [2], le menu qui donne accès à l'API du serveur de calcul de l'impôt :
  - [Authentification] : correspond à la page [authentification]. Cette page fait une requête d'authentification auprès du serveur de calcul de l'impôt avec les identifiants [admin, admin] qui sont pour l'instant les seuls autorisés. Le résultat affiché est analogue à [3] ;
  - [Requête AdminData] : correspond à la page [get-admindata]. Cette page demande au serveur de calcul de l'impôt, les données, appelées ici [adminData], qui permettent le calcul de l'impôt. Le résultat affiché est analogue à [3] ;
  - [Fin session impôt] : correspond à la page [fin-session]. Cette page fait une requête de fin de session PHP auprès du serveur de calcul de l'impôt. Le serveur annule alors la session PHP courante et en initialise une nouvelle vierge ;

15.5 Les couches [dao] de l'application [nuxt]

Comme indiqué plus haut, l'architecture de l'application [nuxt] sera la suivante :



- en [1], le serveur [node.js] délivre les pages [nuxt] au navigateur [2]. C'est la couche [web] [8] du serveur qui délivre ces pages. Pour délivrer la page, le serveur a pu demander des données externes au serveur de données [3]. C'est la couche [DAO] [9] qui fait les requêtes HTTP nécessaires ;
- à chaque appel de page au serveur [node.js][1], le navigateur [2] reçoit la totalité de l'application [nuxt] qui va alors s'exécuter en mode SPA. Le bloc [UI] (User Interface) [4] présente des pages [vue.js] à l'utilisateur. Les actions de celui-ci ou le cycle de vie des pages peuvent provoquer des appels de données externes au serveur de données [3]. C'est la couche [DAO] [5] qui fait alors les requêtes HTTP nécessaires ;

Nous utiliserons la version 14 du serveur de calcul de l'impôt développé dans le document | Introduction au langage **PHP7** par l'exemple|. Nous utiliserons une partie seulement de son API (**A**pplication **P**rogramming **I**nterface) jSON :

Requête

```
[initialisation d'une session jSON avec le serveur de calcul de l'impôt]
[une session PHP est créée avec le serveur de calcul de l'impôt]

GET main.php?action=init-session&type=json

```

Réponse

```
{
    "action": "init-session",
    "état": 700,
    "réponse": "session démarrée avec type [json]"
}
```

Requête

```
[authentification de l'utilisateur]
[l'authentification est stockée dans la session PHP]

POST main.php?action=authentifier-utilisateur
paramètres postés : user, admin
```

Réponse

```
{
    "action": "authentifier-utilisateur",
    "état": 200,
    "réponse": "Authentification réussie [admin, admin]"
}
```

Requête

```
[demande des données de l'administration fiscale]
[les données reçues sont stockées dans la session PHP]

GET main.php?action=get-admindata
```

Réponse

```
{
    "action": "get-admindata",
    "état": 1000,
    "réponse": {
        "limites": [
            9964,
            27519,
            73779,
            156244,
            0
        ],
        "coeffR": [
            0,
            0.14,
            0.3,
            0.41,
            0.45
        ],
```

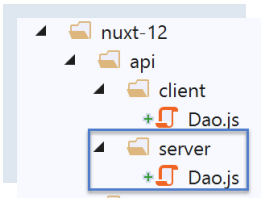
Requête	Réponse

```
[fin de la session PHP avec le serveur de calcul de l'impôt]
[supprime la session PHP courante et crée une nouvelle session PHP. Dans celle-ci, la session jSON reste activée, mais l'utilisateur n'est plus authentifié]

GET main.php?action=fin-session
```

```
        "coeffN": [
            0,
            1394.96,
            5798,
            13913.69,
            20163.45
        ],
        "plafondQfDemiPart": 1551,
        "plafondRevenusCelibataire
PourReduction": 21037,
        "plafondRevenusCouplePourReduction": 42074,
        "valeurReducDemiPart": 3797,
        "plafondDecoteCelibataire": 1196,
        "plafondDecoteCouple": 1970,
        "plafondImpotCouplePourDecote": 2627,
        "plafondImpotCelibatairePourDecote": 1595,
        "abattementDixPourcentMax": 12502,
        "abattementDixPourcentMin": 437
    }
}
```

```
{
    "action": "fin-session",
    "état": 400,
    "réponse": "session supprimée"
}
```

15.5.1 Le couche [dao] du serveur [nuxt]



Le serveur [node.js] [1] va utiliser la couche [dao] décrite dans le document |Introduction au framework **VUE.JS** par l'exemple|. Nous rappelons ici son code :

```
'use strict';

// imports
import qs from 'qs'

class Dao {

  // constructeur
  constructor(axios) {
    this.axios = axios;
    // cookie de session
    this.sessionCookieName = "PHPSESSID";
    this.sessionCookie = '';
  }

  // init session
  async  initSession() {
    // options de la requête HHTP [get /main.php?action=init-session&type=json]
    const options = {
      method: "GET",
      // paramètres de l'URL
```

```
      params: {
        action: 'init-session',
        type: 'json'
      }
    };
    // exécution de la requête HTTP
    return await this.getRemoteData(options);
  }

  async  authentifierUtilisateur(user, password) {
    // options de la requête HHTP [post /main.php?action=authentifier-utilisateur]
    const options = {
      method: "POST",
      headers: {
        'Content-type': 'application/x-www-form-urlencoded',
      },
      // corps du POST
      data: qs.stringify({
        user: user,
        password: password
      }),
      // paramètres de l'URL
      params: {
        action: 'authentifier-utilisateur'
      }
    };
    // exécution de la requête HTTP
    return await this.getRemoteData(options);
  }

  async getAdminData() {
    // options de la requête HHTP [get /main.php?action=get-admindata]
    const options = {
      method: "GET",
      // paramètres de l'URL
      params: {
        action: 'get-admindata'
      }
    };
    // exécution de la requête HTTP
    const data = await this.getRemoteData(options);
    // résultat
    return data;
  }

  async  getRemoteData(options) {
    // pour le cookie de session
    if (!options.headers) {
      options.headers = {};
    }
    options.headers.Cookie = this.sessionCookie;
    // exécution de la requête HTTP
    let response;
    try {
      // requête asynchrone
      response = await this.axios.request('main.php', options);
    } catch (error) {
      // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
      if (error.response) {
        // la réponse du serveur est dans [error.response]
        response = error.response;
      } else {
        // on relance l'erreur
        throw error;
      }
    }
    // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-
  même)
    // on récupère le cookie de session s'il existe
    const setCookie = response.headers['set-cookie'];
    if (setCookie) {
      // setCookie est un tableau
      // on cherche le cookie de session dans ce tableau
      let trouvé = false;
```

```
95.       let i = 0;
96.       while (!trouvé && i < setCookie.length) {
97.         // on cherche le cookie de session
98.         const results = RegExp('^(' + this.sessionCookieName + '.+?);').exec(setCookie[i]);
99.         if (results) {
100.          // on mémorise le cookie de session
101.          // eslint-disable-next-line require-atomic-updates
102.          this.sessionCookie = results[1];
103.          // on a trouvé
104.          trouvé = true;
105.        } else {
106.          // élément suivant
107.          i++;
108.        }
109.      }
110.    }
111.    // la réponse du serveur est dans [response.data]
112.    return response.data;
113.  }
114.}
115.
116.// export de la classe
117.export default Dao;
```

- toutes les méthodes de la couche [dao] rendent l'objet envoyé par le serveur de données [{**action** : 'xx', **état** : nn, **réponse** : {...}] avec :
  - [action] : le nom de l'action exécutée par le serveur de données ;
  - [état] : indicateur numérique :
    - [`initSession`] : **état**=700 pour une réponse sans erreur ;
    - [`authentifierUtilisateur`] : **état**=200 pour une réponse sans erreur ;
    - [`getAdminData`] : **état**=1000 pour une réponse sans erreur ;
    - [fin-session] : **état**=400 pour une réponse sans erreur ;
  - [réponse] : réponse associée à l'indicateur numérique [état]. Peut varier selon cet indicateur numérique ;

Examinons le constructeur de la classe [Dao] :

```
1.  // constructeur
2.    constructor(axios) {
3.      this.axios = axios;
4.      // cookie de session
5.      this.sessionCookieName = "PHPSESSID";
6.      this.sessionCookie = '';
7.  }
```

- ligne 2 : l'objet [axios] fourni en argument au constructeur est fourni par le code appelant. C'est lui qui va faire les requêtes HTTP ;
- ligne 5 : le nom du cookie de session envoyé par le serveur de données écrit en PHP ;
- ligne 6 : le cookie de session qui est échangé entre la couche [dao] et le serveur de données. Celui-ci est initialisé par la fonction [getRemoteData] des lignes 67-113 ;

Pour le cookie de session, il nous faut considérer deux couches [dao] séparées :
- celle du navigateur ;
- celle du serveur ;

Nous allons devoir gérer trois cookies de session :

1. celui échangé entre le client [nuxt] et le serveur PHP 7 ;
2. celui échangé entre le serveur [nuxt] et le serveur PHP 7 ;
3. celui échangé entre le client [nuxt] et le serveur [nuxt] ;

Nous ferons en sorte que le cookie de la session avec le serveur PHP soit le même pour le client et le serveur [nuxt]. Nous appellerons ce cookie, **cookie de la session PHP**. Ce cookie est celui des cas 1 et 2. Nous appellerons **cookie de la session [nuxt]**, le cookie du cas 3. Nous aurons donc deux sessions :

- une session PHP avec le cookie de session PHP ;
- une session [nuxt] avec le cookie de session [nuxt] ;

Pourquoi utiliser le même cookie pour les sessions PHP du client et du navigateur [nuxt] ? Nous voulons que l'application puisse dialoguer avec le serveur PHP 7 indifféremment avec le client ou le serveur [nuxt] :

- si une action A du serveur [nuxt] met le serveur PHP dans un état E, cet état est reflété dans la session PHP entretenue par le serveur PHP ;
- en utilisant le même cookie de session PHP que le serveur, une action B du client [nuxt] qui suivrait l'action A du serveur [nuxt] retrouverait le serveur PHP dans l'état E laissé par le serveur [nuxt] et pourrait donc s'appuyer sur le travail déjà fait par le serveur [nuxt] ;
- si après l'action B du client [nuxt], vient une action C du serveur [nuxt], pour la même raison que précédemment, cette action va pouvoir s'appuyer sur le travail fait par l'action B du client [nuxt] ;

Pour que le navigateur du client [nuxt] puisse dialoguer avec le serveur PHP du calcul de l'impôt, nous utiliserons la version 14 de ce serveur **qui autorise les appels inter-domaines**, ç-à-d ceux d'un navigateur vers le serveur PHP. Les appels du serveur [nuxt] vers le serveur PHP ne sont pas, eux, des appels inter-domaines. Cette notion n'existe que pour les appels faits depuis un navigateur.

Revenons au code du constructeur de la classe [Dao] précédente :

```
1.  // constructeur
2.    constructor(axios) {
3.      this.axios = axios;
4.      // cookie de session
5.      this.sessionCookieName = "PHPSESSID";
6.      this.sessionCookie = '';
7.  }
```

- les lignes 5 et 6 correspondent au cookie de la session PHP avec le serveur de calcul de l'impôt ;

La gestion du cookie de la session PHP ci-dessus ne convient pas au serveur [nuxt] : sa couche [dao] **est instanciée à chaque nouvelle requête faite au serveur [nuxt]**. On se rappelle en effet que demander une page au serveur [nuxt] revient à réinitialiser l'application [nuxt]. Ainsi lorsqu'à l'issue de la 1ère requête faite au serveur de données par le serveur [nuxt], le cookie de session PHP de la couche [dao] est initialisé, cette valeur est perdue lors de la requête HTTP suivante du même serveur [nuxt], car entre-temps sa couche [dao] a été recréée, le constructeur réexécuté et le cookie de session PHP réinitialisé avec la chaîne vide (ligne 6) ;

Une solution est d'utiliser un autre constructeur pour la couche [dao] du serveur :

```
1.  // constructeur
2.    constructor(axios, phpSessionCookie) {
3.      // bibliothèque axios
4.      this.axios = axios
5.      // valeur du cookie de session
6.      this.phpSessionCookie = phpSessionCookie
7.      // nom du cookie de session du serveur PHP
8.      this.phpSessionCookieName = 'PHPSESSID'
9.    }
```

- ligne 2 : cette fois-ci le cookie de la session PHP sera fourni au constructeur de la couche [dao] du serveur de données ;

Comment le serveur [nuxt] pourra-t-il fournir ce cookie de session PHP au constructeur de sa couche [dao] ? Nous stockerons le cookie de session PHP dans le cookie de session [nuxt] échangé entre le navigateur et le serveur [nuxt]. Le processus est le suivant :

1. l'application [nuxt] est lancée ;
2. lorsque le serveur [nuxt] fait sa 1ère requête HTTP avec le serveur PHP, il stocke le cookie de la session PHP qu'il a reçu dans le cookie de session [nuxt] qu'il échange avec le client [nuxt] ;
3. le navigateur qui loge le client [nuxt] reçoit ce cookie de session [nuxt] et le renvoie donc systématiquement à chaque nouvelle requête au serveur [nuxt] ;
4. lorsque le serveur [nuxt] devra faire une nouvelle requête au serveur PHP, il retrouvera le cookie de session PHP dans le cookie de session [nuxt] que le navigateur lui aura envoyé. Il l'enverra alors au serveur PHP ;

Il y a bien deux cookies de session et il ne faut pas les confondre :

- le cookie de session [nuxt] échangé entre le serveur [nuxt] et le navigateur du client [nuxt] ;

- le cookie de session PHP échangé entre le serveur [nuxt] et le serveur PHP ou entre le client [nuxt] et le serveur PHP ;

Revenons maintenant sur le code de la méthode de la classe [Dao]. Elle n'inclut pas de fonction pour clôre la session PHP avec le serveur de calcul de l'impôt. Nous ajoutons celle-ci :

```
1.  // fin de la session de calcul de l'impôt
2.    async finSession() {
3.      // options de la requête HHTP  [get /main.php?action=fin-session]
4.      const options = {
5.        method: 'GET',
6.        // paramètres de l'URL
7.        params: {
8.          action: 'fin-session'
9.        }
10.     }
11.     // exécution de la requête HTTP
12.     const data = await this.getRemoteData(options)
13.     // résultat
14.     return data
15.   }
```

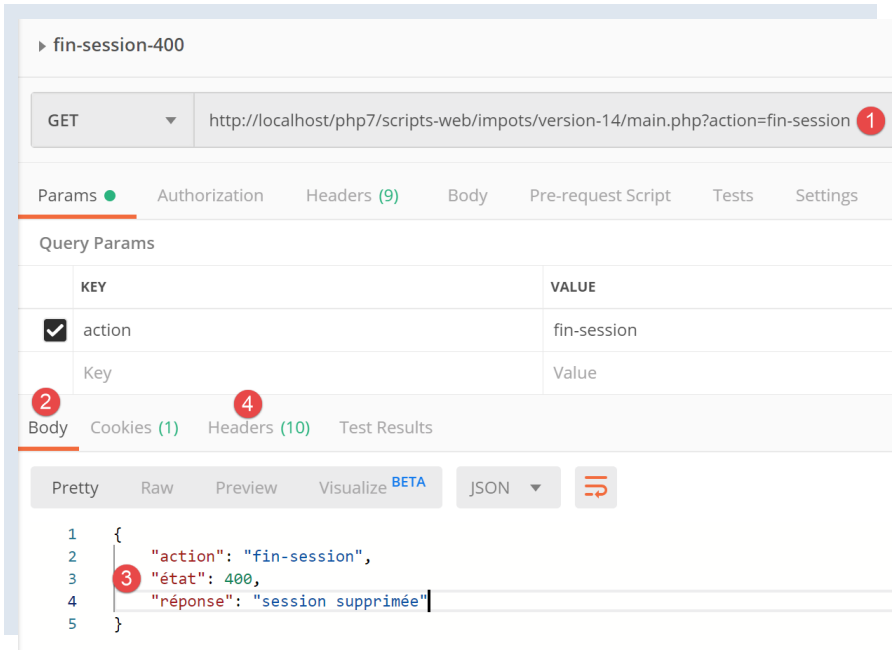
Aux tests, on découvre que la fonction [getRemoteData] appelée ligne 12 ne convient pas pour la méthode [finSession] :

```
1.  async  getRemoteData(options) {
2.      // pour le cookie de session
3.      if (!options.headers) {
4.        options.headers = {};
5.      }
6.      options.headers.Cookie = this.sessionCookie;
7.      // exécution de la requête HTTP
8.      let response;
9.      try {
10.       // requête asynchrone
11.       response = await this.axios.request('main.php', options);
12.     } catch (error) {
13.       // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
14.       if (error.response) {
15.         // la réponse du serveur est dans [error.response]
16.         response = error.response;
17.       } else {
18.         // on relance l'erreur
19.         throw error;
20.       }
21.     }
22.     // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-
    même)
23.     // on récupère le cookie de session s'il existe
24.     const setCookie = response.headers['set-cookie'];
25.     if (setCookie) {
26.       // setCookie est un tableau
27.       // on cherche le cookie de session dans ce tableau
28.       let trouvé = false;
29.       let i = 0;
30.       while (!trouvé && i < setCookie.length) {
31.         // on cherche le cookie de session
32.         const results = RegExp('^(' + this.sessionCookieName + '.+?);').exec(setCookie[i]);
33.         if (results) {
34.           // on mémorise le cookie de session
35.           // eslint-disable-next-line require-atomic-updates
36.           this.sessionCookie = results[1];
37.           // on a trouvé
38.           trouvé = true;
39.         } else {
40.           // élément suivant
41.           i++;
42.         }
43.       }
44.     }
45.     // la réponse du serveur est dans [response.data]
46.     return response.data;
```

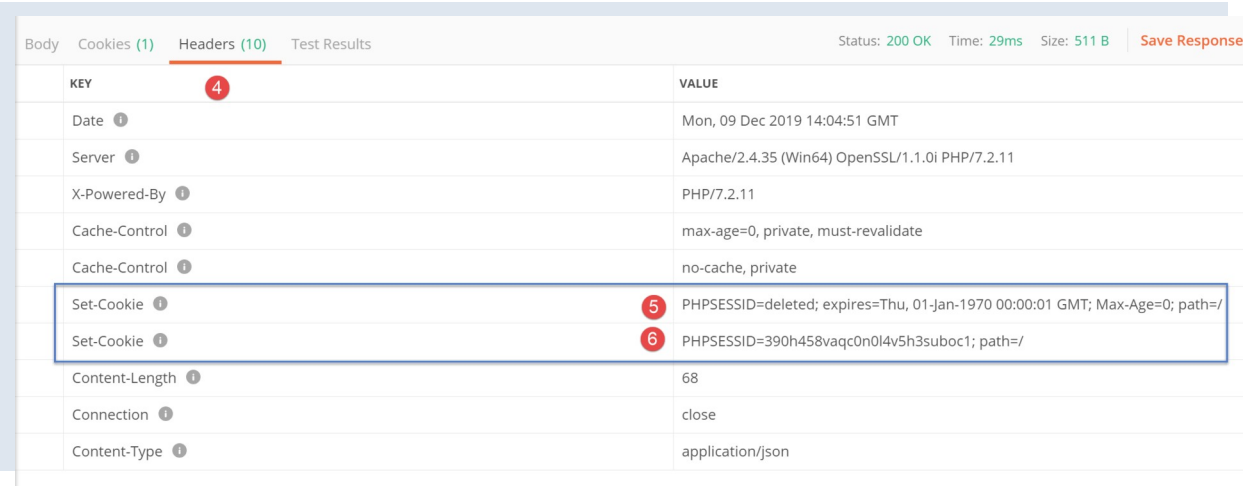
```
47.    }
```

- lignes 30-43 : on recherche le cookie [PHPSESSID=xxx]. Si on le trouve, il est mémorisé dans la classe (ligne 36) ;

Ce code ne convient pas à la nouvelle méthode [finSession] car sur l'action [fin-session], le serveur PHP envoie **deux** cookies avec le nom [PHPSESSID]. Voici un exemple obtenu avec un client [Postman] :



- en [1], la demande du client [Postman] ;
- en [3], la réponse du serveur PHP ;
- en [4], les entêtes HTTP de la réponse du serveur PHP ;



- en [5], le serveur PHP indique d'abord qu'il a supprimé la session PHP courante ;
- en [6], le serveur PHP envoie le cookie de la nouvelle session PHP ;

Avec le code actuel, la fonction [getRemoteData] récupère le cookie [5] alors que c'est le cookie [6] qu'il faut mémoriser.

Il faut donc faire évoluer le code de la fonction [getRemoteData] :

```
1.  async getRemoteData(options) {
2.      // y-a-t-il un cookie de session PHP ?
3.      if (this.phpSessionCookie) {
4.        // y-a-t-il des entêtes ?
5.        if (!options.headers) {
6.          // on crée un objet vide
7.          options.headers = {}
8.        }
9.        // entête du cookie de session PHP
10.       options.headers.Cookie = this.phpSessionCookie
11.     }
12.     // exécution de la requête HTTP
13.     let response
14.     try {
15.       // requête asynchrone
16.       response = await this.axios.request('main.php', options)
17.     } catch (error) {
18.       // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
19.       if (error.response) {
20.         // la réponse du serveur est dans [error.response]
21.         response = error.response
22.       } else {
23.         // on relance l'erreur
24.         throw error
25.       }
26.     }
27.     // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-
    même)
28.     // on cherche le cookie de session PHP dans les cookies reçus
29.     // tous les cookies reçus
30.     const cookies = response.headers['set-cookie']
31.     if (cookies) {
32.       // cookies est un tableau
33.       // on cherche le cookie de session PHP dans ce tableau
34.       let trouvé = false
35.       let i = 0
36.       while (!trouvé && i < cookies.length) {
37.         // on cherche le cookie de session PHP
38.         const results = RegExp('^(' + this.phpSessionCookieName + '.+?)$').exec(cookies[i])
39.         if (results) {
40.           // on mémorise le cookie de session PHP
41.           const phpSessionCookie = results[1]
42.           // y-a-t-il dedans le mot [deleted] ?
43.           const results2 = RegExp(this.phpSessionCookieName +
    '=deleted').exec(phpSessionCookie)
44.           if (!results2) {
45.             // on a le bon cookie de session PHP
46.             this.phpSessionCookie = phpSessionCookie
47.             // on a trouvé
48.             trouvé = true
49.           } else {
50.             // élément suivant
51.             i++
52.           }
53.         } else {
54.           // élément suivant
55.           i++
56.         }
57.       }
58.     }
59.     // la réponse du serveur est dans [response.data]
60.     return response.data
61.   }
```

- ligne 41 : on a trouvé un cookie avec le nom [PHPSESSID]. on le mémorise localement ;
- ligne 43 : on regarde si dans le cookie sauvegardé, il y a la chaîne [PHPSESSID=deleted] ;
- ligne 46 : si la réponse est non, alors c'est qu'on a trouvé le bon cookie [PHPSESSID]. On le mémorise dans la classe ;

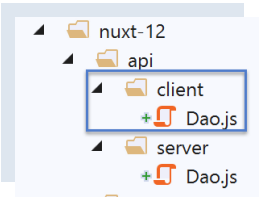
Après la fonction [getRemoteData], le cookie de session PHP est mémorisé dans la classe, dans [this.phpSessionCookie]. On a dit que la classe était instanciée à chaque nouvelle requête HTTP du serveur [nuxt]. Le cookie de session PHP doit donc être exfiltré de la classe. Pour cela, on ajoute une nouvelle méthode à celle-ci :

```
1.  // accès au cookie de la session PHP
2.    getPhpSessionCookie() {
3.      return this.phpSessionCookie
4.  }
```

- le serveur [nuxt] demande une action à sa couche [dao] en fournissant le cookie de la session PHP à son constructeur, s'il en a un ;
- une fois l'action faite, le serveur [nuxt] récupère le cookie de session PHP mémorisé par la couche [dao] à l'aide de la méthode [getPhpSessionCookie] précédente. Ce cookie peut être le même que le précédent ou un autre. Ce dernier cas arrive à deux occasions :
  - lors de l'exécution de la méthode [initSession] (il n'y avait pas de cookie de session PHP avant) ;
  - lors de l'exécution de la méthode [finSession] (le serveur PHP change le cookie de session PHP) ;

Notons une particularité sur le cookie de session PHP. Le serveur [nuxt] ne reçoit pas toujours ce cookie de la part du serveur PHP. En effet, celui-ci ne l'envoie qu'une fois. Ensuite il ne l'envoie plus. Lorsqu'on regarde le code de [getRemoteData] et celui de [getPhpSessionCookie] on verra alors que lorsque le serveur PHP n'envoie pas de cookie de session, la fonction [getPhpSessionCookie] renvoie alors le cookie de session PHP fourni au constucteur. C'est ainsi que le serveur envoie toujours au serveur PHP le dernier cookie de session PHP que celui-ci lui a envoyé.

15.5.2 La couche [dao] du client [nuxt]



Pour le client [nuxt] qui s'exécute dans un navigateur, on reprend le code de la classe [Dao] du document |Introduction au framework **VUE.JS** par l'exemple| :

```
1.  "use strict";
2.
3.  // imports
4.  import qs from "qs";
5.
6.  class Dao {
7.    // constructeur
8.    constructor(axios) {
9.      this.axios = axios;
10.   }
11.
12.   // init session
13.   async initSession() {
14.     // options de la requête HHTP [get /main.php?action=init-session&type=json]
15.     const options = {
16.       method: "GET",
17.       // paramètres de l'URL
18.       params: {
19.         action: "init-session",
20.         type: "json"
21.       }
22.     };
23.     // exécution de la requête HTTP
24.     return await this.getRemoteData(options);
25.   }
26.
27.   async authentifierUtilisateur(user, password) {
28.     // options de la requête HHTP [post /main.php?action=authentifier-utilisateur]
29.     const options = {
30.       method: "POST",
```

```
31.       headers: {
32.         "Content-type": "application/x-www-form-urlencoded"
33.       },
34.       // corps du POST
35.       data: qs.stringify({
36.         user: user,
37.         password: password
38.       }),
39.       // paramètres de l'URL
40.       params: {
41.         action: "authentifier-utilisateur"
42.       }
43.     };
44.     // exécution de la requête HTTP
45.     return await this.getRemoteData(options);
46.   }
47.
48.   async getAdminData() {
49.     // options de la requête HHTP  [get /main.php?action=get-admindata]
50.     const options = {
51.       method: "GET",
52.       // paramètres de l'URL
53.       params: {
54.         action: "get-admindata"
55.       }
56.     };
57.     // exécution de la requête HTTP
58.     const data = await this.getRemoteData(options);
59.     // résultat
60.     return data;
61.   }
62.
63.   async getRemoteData(options) {
64.     // exécution de la requête HTTP
65.     let response;
66.     try {
67.       // requête asynchrone
68.       response = await this.axios.request("main.php", options);
69.     } catch (error) {
70.       // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
71.       if (error.response) {
72.         // la réponse du serveur est dans [error.response]
73.         response = error.response;
74.       } else {
75.         // on relance l'erreur
76.         throw error;
77.       }
78.     }
79.     // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-
    même)
80.     // la réponse du serveur est dans [response.data]
81.     return response.data;
82.   }
83. }
84.
85. // export de la classe
86. export default Dao;
```

Ce code se distingue de la couche [dao] du serveur [nuxt] par le fait qu'il ne gère pas le cookie de la session PHP avec le serveur de calcul de l'impôt : c'est le navigateur qui le fait.

Nous allons, comme nous l'avons fait pour la couche [dao] du serveur [nuxt], ajouter une méthode [finSession] :

```
1.  // fin de la session de calcul de l'impôt
2.    async finSession() {
3.      // options de la requête HHTP  [get /main.php?action=fin-session]
4.      const options = {
5.        method: 'GET',
6.        // paramètres de l'URL
7.        params: {
8.          action: 'fin-session'
9.        }
10.     }
```

```
11.     // exécution de la requête HTTP
12.     const data = await this.getRemoteData(options)
13.     // résultat
14.     return data
15.   }
```

Lorsque le client [nuxt] exécute cette méthode, il reçoit, comme le serveur [nuxt], deux cookies de session PHP. C'est en fait le navigateur qui les reçoit et il gère correctement la situation : il ne garde que le cookie de la nouvelle session PHP qu'a initiée le serveur de calcul de l'impôt. Donc à la prochaine action du client [nuxt] vers le serveur PHP, le cookie de session PHP sera correct car c'est le navigateur qui envoie celui-ci. Il y a cependant un problème : le serveur [nuxt] n'a pas connaissance du fait que le cookie de session PHP a changé. Dans ses échanges avec le serveur PHP, il va alors envoyer un cookie de session PHP qui n'existe plus et on va avoir des problèmes. Il faudrait que le client [nuxt] avertisse le serveur [nuxt] que le cookie de session PHP a changé et lui transmette celui-ci. On sait comment il peut faire cela : via le cookie de session [nuxt], le cookie échangé entre le client et le serveur [nuxt]. Le client [nuxt] a au moins deux façons de récupérer le nouveau cookie de session PHP :

1. en le demandant au navigateur ;
2. en utilisant la méthode [getRemoteData] du serveur qui sait comment récupérer le nouveau cookie de session PHP ;

Nous allons utiliser la 2ième solution car elle est déjà toute prête. La méthode [getRemoteData] du client [nuxt] devient alors la suivante :

```
1.  async getRemoteData(options) {
2.      // exécution de la requête HTTP
3.      let response
4.      try {
5.        // requête asynchrone
6.        response = await this.axios.request('main.php', options)
7.      } catch (error) {
8.        // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
9.        if (error.response) {
10.         // la réponse du serveur est dans [error.response]
11.         response = error.response
12.       } else {
13.         // on relance l'erreur
14.         throw error
15.       }
16.     }
17.     // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-
    même)
18.     // on cherche le cookie de session PHP dans les cookies reçus
19.     // tous les cookies reçus
20.     const cookies = response.headers['set-cookie']
21.     if (cookies) {
22.       // cookies est un tableau
23.       // on cherche le cookie de session PHP dans ce tableau
24.       let trouvé = false
25.       let i = 0
26.       while (!trouvé && i < cookies.length) {
27.         // on cherche le cookie de session PHP
28.         const results = RegExp('^(' + this.phpSessionCookieName + '.+?)$').exec(cookies[i])
29.         if (results) {
30.           // on mémorise le cookie de session PHP
31.           const phpSessionCookie = results[1]
32.           // y-a-t-il dedans le mot [deleted] ?
33.           const results2 = RegExp(this.phpSessionCookieName +
    '=deleted').exec(phpSessionCookie)
34.           if (!results2) {
35.             // on a le bon cookie de session PHP
36.             this.phpSessionCookie = phpSessionCookie
37.             // on a trouvé
38.             trouvé = true
39.           } else {
40.             // élément suivant
41.             i++
42.           }
43.         } else {
44.           // élément suivant
45.           i++
46.         }
```

```
47.       }
48.     }
49.     // la réponse du serveur est dans [response.data]
50.     return response.data
51.   }
```

On a gardé dans [getRemoteData] uniquement le code qui exploite la réponse du serveur PHP à la recherche du cookie de session PHP. On n'a pas gardé le code qui incluait le cookie de session PHP dans la requête au serveur PHP car c'est le navigateur qui abrite le client [nuxt] qui s'en charge.

Une fois le cookie de session PHP obtenu par le client [nuxt], celui-ci doit être mis dans la session [nuxt] pour que le serveur [nuxt] puisse en bébéficier. Ce n'est pas la couche [dao] qui s'occupe de cela mais elle donne accès par une méthode au cookie de session PHP qu'elle a mémorisé :

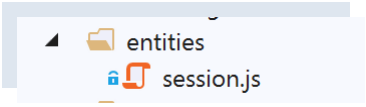
```
1.  // accès au cookie de la session PHP
2.    getPhpSessionCookie() {
3.      return this.phpSessionCookie
4.  }
```

La fonction [`getPhpSessionCookie`] ne rend pas toujours un cookie de session valide :

- il faut se souvenir ici que la couche [dao] du client [nuxt] est persistante. Elle est instanciée une fois et reste ensuite en mémoire ;
- tant que le serveur PHP n'envoie pas un cookie de session PHP au client [nuxt], la fonction [`getPhpSessionCookie`] du client [nuxt] renvoie une valeur [undefined] ;
- lorsque le serveur PHP envoie un cookie de session PHP au client [nuxt], celui-ci est mémorisé dans [this.phpSessionCookie] et le restera tant qu'il ne sera pas changé par un nouveau cookie de session PHP envoyé par le serveur PHP. La fonction [`getPhpSessionCookie`] du client [nuxt] renvoie alors le dernier cookie de session PHP reçu ;

La couche [dao] du client [nuxt] ne diffère de celle du serveur [nuxt] que par un point : elle n'envoie pas le cookie de session PHP elle-même car c'est le navigateur qui le fait. Néanmoins on a préféré garder deux couches [dao] distinctes car les raisonnements qui mènent à leurs écritures respectives sont différents.

15.6 La session [nuxt]



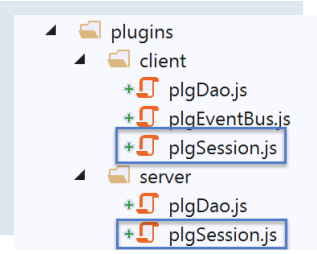
La session [nuxt] (entre client et serveur nuxt) sera encapsulée dans l'objet [session] suivant :

```
1.  /* eslint-disable no-console */
2.  // définition de la session
3.  const session = {
4.    // contenu de la session
5.    value: {
6.      // store non initialisé
7.      initStoreDone: false,
8.      // valeur du store Vuex
9.      store: ''
10.   },
11.   // sauvegarde de la session dans un cookie
12.   save(context) {
13.     // sauvegarde du store en session
14.     this.value.store = context.store.state
15.     console.log('nuxt-session save=', this.value)
16.     // sauvegarde de la valeur de la session
17.     context.app.$cookies.set('nuxt-session', this.value, { path: context.base, maxAge:
    context.env.maxAge })
18.   },
19.   // reset de la session
20.   reset(context) {
21.     console.log('nuxt-session reset')
```

```
22.     // reset du store
23.     context.store.commit('reset')
24.     // sauvegarde du nouveau store en session et sauvegarde de la session
25.     this.save(context)
26.   }
27. }
28. // export de la session
29. export default session
```

- lignes 5-10 : la session n'a qu'une propriété [value] avec deux sous-propriétés :
  - [initStoreDone] qui indique si le store a été initialisé ou pas ;
  - [store] : la valeur [store.state] du store Vuex de l'application ;
- lignes 12-18 : la méthode [save] sert à sauvegarder la session [nuxt] dans un cookie. On utilise ici la bibliothèque [cookie-universal-nuxt] pour gérer le cookie. On notera le nom du cookie de la session [nuxt] : [nuxt-session] (ligne 17) ;
- lignes 20-26 : la méthode [reset] réinitialise la session [nuxt] ;
  - ligne 23 : le store Vuex est réinitialisé puis sauvegardé en session, ligne 25 ;

15.7 Les plugins de gestion de la session [nuxt]



15.7.1 Le plugin de gestion de la session [nuxt] du *serveur* [nuxt]

Au démarrage de l'application, c'est le serveur [nuxt] qui opère le premier. C'est donc lui qui va initialiser la session [nuxt]. Le script [server/plgSession] est le suivant :

```
1.  /* eslint-disable no-console */
2.
3.  // import de la session
4.  import session from '@/entities/session'
5.
6.  export default (context, inject) => {
7.    // gestion de la session serveur
8.    console.log('[plugin server plgSession]')
9.
10.   // y-a-t-il une session existante ?
11.   const value = context.app.$cookies.get('nuxt-session')
12.   if (!value) {
13.     // nouvelle session
14.     console.log("[plugin server plgSession], démarrage d'une nouvelle session")
15.   } else {
16.     // session existante
17.     console.log("[plugin server plgSession], reprise d'une session existante")
18.     session.value = value
19.   }
20.
21.   // on injecte une fonction dans [context, Vue] qui rendra la session courante
22.   inject('session', () => session)
23. }
```

- ligne 4 : on importe le code de la session [nuxt] ;
- ligne 11 : on récupère la valeur du cookie de la session [nuxt] ;
- lignes 12-15 : si le cookie de la session [nuxt] n'existait pas, alors la session [nuxt] importée ligne 4 est suffisante. Il n'y a rien de plus à faire ;

- lignes 15-19 : si le cookie de la session [nuxt] existait, alors ligne 18 on stocke sa valeur dans la session importée ligne 4 ;
- ligne 22 : la session a été soit initialisée soit restaurée. On la rend disponible via la fonction [$session] ;

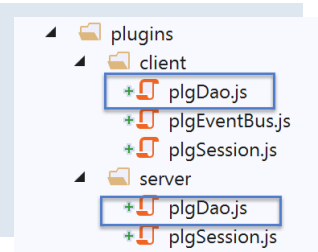
15.7.2 Le plugin de gestion de la session [nuxt] du client [nuxt]

Le script [client/plgSession] est le suivant :

```
1.  /* eslint-disable no-console */
2.
3.  // import de la session
4.  import session from '@/entities/session'
5.
6.  export default (context, inject) => {
7.    // gestion de la session client
8.    console.log('[plugin client plgSession], reprise de la session [nuxt] du serveur')
9.    // on récupère la session existante du serveur nuxt
10.   session.value = context.app.$cookies.get('nuxt-session')
11.
12.   // on injecte une fonction dans [context, Vue] qui rendra la session courante
13.   inject('session', () => session)
14. }
```

- ligne 4 : la session [nuxt] est importée ;
- ligne 10 : on récupère la session [nuxt] courante dans le cookie [nuxt-session] ;
- ligne 13 : on rend la session [nuxt] importée ligne 4 au travers de la fonction injectée [$session] ;

15.8 Les plugins des couches [dao]



15.8.1 Le plugin de la couche [dao] du client [nuxt]

Le script [client/plgDao] est le suivant :

```
1.  /* eslint-disable no-console */
2.  // on crée un point d'accès à la couche [Dao]
3.  import Dao from '@/api/client/Dao'
4.  export default (context, inject) => {
5.    // configuration axios
6.    context.$axios.defaults.timeout = context.env.timeout
7.    context.$axios.defaults.baseURL = context.env.baseURL
8.    context.$axios.defaults.withCredentials = context.env.withCredentials
9.    // instanciation de la couche [dao]
10.   const dao = new Dao(context.$axios)
11.   // injection d'une fonction [$dao] dans le contexte
12.   inject('dao', () => dao)
13.   // log
14.   console.log('[fonction client $dao créée]')
15. }
```

- ligne 3 : la couche [dao] du client [nuxt] est importée ;
- lignes 6-8 : on configure l'objet [context.$axios] qui va faire les requêtes HTTP de la couche [dao] du client [nuxt] avec les informations du fichier [nuxt.config] :

```
1.  // environnement
2.    env: {
3.      // configuration axios
```

```
4.      timeout: 2000,
5.      withCredentials: true,
6.      baseURL: 'http://localhost/php7/scripts-web/impots/version-14',
7.      // configuration du cookie de session [nuxt]
8.      maxAge: 60 * 5
9.    }
```

- ligne 10 : la couche [dao] du client [nuxt] est instanciée ;
- ligne 12 : la fonction [$dao] est injectée dans le contexte et les pages du client. Cette fonction donne accès à la couche [dao] de la ligne 10 ;

On retiendra donc que pour avoir accès à la couche [dao] du client [nuxt] lorsque celui-ci est exécuté, on écrira :

- [**context.app.$dao()**] là où le contexte est connu ;
- [**this.$dao()**] dans une page [Vue.js] ;

15.8.2 Le plugin de la couche [dao] du *serveur* [nuxt]

Le script [server/plgDao] est le suivant :

```
1.  /* eslint-disable no-console */
2.  // on crée un point d'accès à la couche [Dao]
3.  import Dao from '@/api/server/Dao'
4.  export default (context, inject) => {
5.    // configuration axios
6.    context.$axios.defaults.timeout = context.env.timeout
7.    context.$axios.defaults.baseURL = context.env.baseURL
8.    // on récupère le cookie de session
9.    const store = context.app.$session().value.store
10.   const phpSessionCookie = store ? store.phpSessionCookie : ''
11.   console.log('session=', context.app.$session().value, 'phpSessionCookie=', phpSessionCookie)
12.   // instanciation de la couche [dao]
13.   const dao = new Dao(context.$axios, phpSessionCookie)
14.   // injection d'une fonction [$dao] dans le contexte
15.   inject('dao', () => dao)
16.   // log
17.   console.log('[fonction server $dao créée]')
18. }
```

- ligne 3 : la couche [dao] du serveur [nuxt] est importée ;
- lignes 6-7 : on configure l'objet [context.$axios] qui va faire les requêtes HTTP de la couche [dao] du serveur [nuxt] avec les informations du fichier [nuxt.config] :

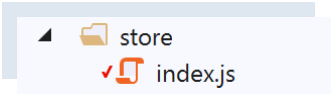
```
1.  // environnement
2.    env: {
3.      // configuration axios
4.      timeout: 2000,
5.      withCredentials: true,
6.      baseURL: 'http://localhost/php7/scripts-web/impots/version-14',
7.      // configuration du cookie de session [nuxt]
8.      maxAge: 60 * 5
9.    }
```

- ligne 9 : on récupère le store de l'application [nuxt] ;
- ligne 10 : si le store existe, on récupère le cookie de la session PHP car on en a besoin pour instancier la couche [dao] du serveur [nuxt] ;
- ligne 13 : on instancie la couche [dao] du serveur [nuxt] ;
- ligne 15 : la fonction [$dao] est injectée dans le contexte et les pages du serveur [nuxt]. Cette fonction donne accès à la couche [dao] de la ligne 13 ;

On retiendra donc que pour avoir accès à la couche [dao] du serveur [nuxt] lorsque celui-ci est exécuté, on écrira :

- [**context.app.$dao()**] là où le contexte est connu ;
- [**this.$dao()**] dans une page [Vue.js] ;

15.9 Le store Vuex



Le store [Vuex] va mémoriser toutes les données qui doivent être partagées par les différentes composantes de l'application [pages, client, serveur] sans que pour autant ces données soient réactives.

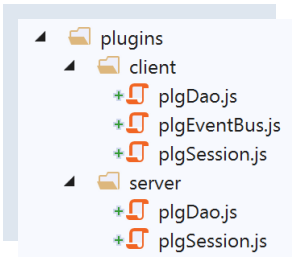
```
1.  /* eslint-disable no-console */
2.
3.  // état du store
4.  export const state = () => ({
5.    // session jSON démarrée
6.    jsonSessionStarted: false,
7.    // utilisateur authentifié
8.    userAuthenticated: false,
9.    // cookie de session PHP
10.   phpSessionCookie: '',
11.   // adminData
12.   adminData: ''
13. })
14.
15. // mutations du store
16. export const mutations = {
17.   // remplacement du state
18.   replace(state, newState) {
19.     for (const attr in newState) {
20.       state[attr] = newState[attr]
21.     }
22.   },
23.   // reset du store
24.   reset() {
25.     this.commit('replace', { jsonSessionStarted: false, userAuthenticated: false,
    phpSessionCookie: '', adminData: '' })
26.   }
27. }
28.
29. // actions du store
30. export const actions = {
31.   nuxtServerInit(store, context) {
32.     // qui exécute ce code ?
33.     console.log('nuxtServerInit, client=', process.client, 'serveur=', process.server, 'env=',
    context.env)
34.     // init session
35.     initStore(store, context)
36.   }
37. }
38.
39. function initStore(store, context) {
40.   // store est le store à initialiser
41.   // on récupère la session
42.   const session = context.app.$session()
43.   // la session a-t-elle été déjà initialisée ?
44.   if (!session.value.initStoreDone) {
45.     // on démarre un nouveau store
46.     console.log("nuxtServerInit, initialisation d'une nouvelle session")
47.     // on met le store dans la session
48.     session.value.store = store.state
49.     // le store est désormais initialisé
50.     session.value.initStoreDone = true
51.   } else {
52.     console.log("nuxtServerInit, reprise d'un store existant")
53.     // on met à jour le store avec le store de la session
54.     store.commit('replace', session.value.store)
55.   }
56.   // on sauvegarde la session
57.   session.save(context)
58.   // log
```

```
59.   console.log('initStore terminé, store=', store.state)
60. }
```

Les données mémorisées dans le store sont les suivantes :

- ligne 6 : [jsonSessionStarted] sera positionnée à vrai dès que l'initialisation d'une session jSON avec le serveur PHP aura été réussie, qu'elle ait été faite par le client ou le serveur [nuxt]. A l'issue de cette initialisation, le cookie de session avec le serveur PHP aura été récupéré et placé dans la propriété [phpSessionCookie], ligne 10 ;
- ligne 8 : [userAuthenticated] sera positionnée à vrai dès que l'authentification auprès du serveur PHP aura été réussie, qu'elle ait été faite par le client ou le serveur [nuxt] ;
- ligne 12 : [adminData] sera la valeur [adminData] obtenue auprès du serveur PHP une fois l'authentification réussie ;
- lignes 18-22 : la mutation [replace] permet d'initialiser les propriétés précédentes avec celles d'un objet passé en paramètre ;
- lignes 24-26 : la mutation [reset] redonne leurs valeurs initiales aux propriétés du store ;
- lignes 31-37 : la fonction [nuxtServerInit] délègue son travail à la fonction [initStore] ;
- lignes 39-60 : la fonction [initStore] a deux rôles :
  - si le store n'a pas été initialisé, il est initialisé et mis en session ;
  - si le store a déjà été initialisé, sa valeur est récupérée dans la session [nuxt] ;
- ligne 42 : on récupère la session nuxt ;
- ligne 44 : on regarde si le store a été initialisé :
  - si ce n'est pas le cas, on met le store initial dans la session (ligne 48) ;
  - puis ligne 50, on indique que le store a été initialisé ;
- lignes 51-55 : si le store était initialisé, on utilise alors celui-ci, ligne 54, pour initialiser le store à la valeur contenue dans la session ;
- ligne 57 : dans tous les cas, la session est sauvegardée dans le cookie [nuxt-session], avec le store qu'elle contient ;

15.10 Le plugin [plgEventBus]



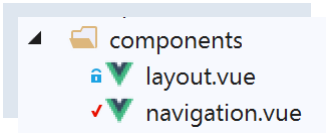
Ce plugin vise à rendre un bus d'événements accessible au client [nuxt] via une fonction [$eventBus] injectée dans le contexte du client [nuxt]. Il est inutile de l'injecter dans le contexte du serveur [nuxt] car celui-ci ne sait pas gérer les événements. Néanmoins nous avons déjà vu que l'injecter côté serveur puis l'utiliser ne provoque pas d'erreur.

```
1.  /* eslint-disable no-console */
2.  // on crée un bus d'événements entre les vues
3.  import Vue from 'vue'
4.  export default (context, inject) => {
5.    // le bus d'événements
6.    const eventBus = new Vue()
7.    // injection d'une fonction [$eventBus] dans le contexte
8.    inject('eventBus', () => eventBus)
9.    // log
10.   console.log('[fonction $eventBus créée]')
11. }
```

Nous avons déjà rencontré ce plugin au paragraphe lien. La fonction [$eventBus] sera disponible au client via les notations :

- [**context.app.$eventBus()**] là où le contexte est disponible ;
- [**this.$eventBus()**] dans les pages [Vue.js] du client ;

15.11 Les composants de l'application [nuxt]



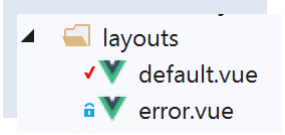
Le composant [layout] est celui des exemples précédents :

```
<!-- disposition des vues -->
<template>
  <!-- ligne -->
  <div>
    <b-row>
      <!-- zone à trois colonnes -->
      <b-col v-if="left" cols="3">
        <slot name="left" />
      </b-col>
      <!-- zone à neuf colonnes -->
      <b-col v-if="right" cols="9">
        <slot name="right" />
      </b-col>
    </b-row>
  </div>
</template>

<script>
export default {
  // paramètres
  props: {
    left: {
      type: Boolean
    },
    right: {
      type: Boolean
    }
  }
}
</script>
```

Le composant [navigation] est le suivant :

```
<template>
  <!-- menu Bootstrap à trois options -->
  <b-nav vertical>
    <b-nav-item to="/authentification" exact exact-active-class="active">
      Authentification
    </b-nav-item>
    <b-nav-item to="/get-admindata" exact exact-active-class="active">
      Requête AdminData
    </b-nav-item>
    <b-nav-item to="/fin-session" exact exact-active-class="active">
      Fin session impôt
    </b-nav-item>
  </b-nav>
</template>
```

15.12 Les layouts de l'application [nuxt]



15.12.1 [default]

Le layout [default] est celui utilisé pour l'exemple [nuxt-11] au paragraphe lien :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <!-- un message -->
5.        <b-alert show variant="success" align="center">
6.          <h4>[nuxt-12] : requêtes HTTP avec axios</h4>
7.        </b-alert>
8.        <!-- la vue courante du routage -->
9.        <nuxt />
10.       <!-- message d'attente -->
11.       <b-alert v-if="showLoading" show variant="light">
12.         <strong>Requête au serveur de données en cours...</strong>
13.         <div class="spinner-border ml-auto" role="status" aria-hidden="true"></div>
14.       </b-alert>
15.       <!-- erreur d'une opération asynchrone -->
16.       <b-alert v-if="showErrorLoading" show variant="danger">
17.         <strong>La requête au serveur de données a échoué : {{ errorLoadingMessage }}</strong>
18.       </b-alert>
19.     </b-card>
20.   </div>
21. </template>
22.
23. <script>
24. /* eslint-disable no-console */
25. export default {
26.   name: 'App',
27.   data() {
28.     return {
29.       showLoading: false,
30.       showErrorLoading: false
31.     }
32.   },
33.   // cycle de vie
34.   beforeCreate() {
35.     console.log('[default beforeCreate]')
36.   },
37.   created() {
38.     console.log('[default created]')
39.     if (process.client) {
40.       // on écoute l'évt [loading]
41.       this.$eventBus().$on('loading', this.mShowLoading)
42.       // ainsi que l'évt [errorLoadingMessage]
43.       this.$eventBus().$on('errorLoading', this.mShowErrorLoading)
44.     }
45.   },
46.   beforeMount() {
47.     console.log('[default beforeMount]')
48.   },
49.   mounted() {
50.     console.log('[default mounted]')
51.   },
52.   methods: {
53.     // gestion du message d'attente
54.     mShowLoading(value) {
55.       console.log('[default mShowLoading], showLoading=', value)
56.       this.showLoading = value
57.     },
58.     // erreur d'une opération asynchrone
59.     mShowErrorLoading(value, errorLoadingMessage) {
60.       console.log('[default mShowErrorLoading], showErrorLoading=', value,
    'errorLoadingMessage=', errorLoadingMessage)
61.       this.showErrorLoading = value
62.       this.errorLoadingMessage = errorLoadingMessage
63.     }
64.   }
65. }
66. </script>
```

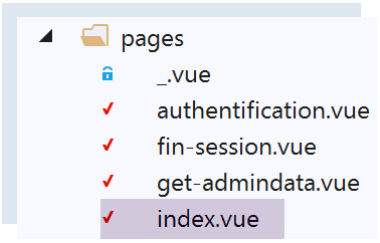
- lignes 10-14 : affichent le message d'attente de la fin d'une opération asynchrone du client [nuxt] ;
- lignes 15-18 : affichent l'éventuel message d'erreur d'une opération asynchrone ;
- ligne 37 : la fonction [created] de la page [default] est exécutée avant la fonction [mounted] des pages ;
- ligne 39 : si l'exécuteur est le client [nuxt], alors la page [default] se met à l'écoute des événements :
  - [loading] qui signale le début ou la fin d'une attente. La fonction [mShowLoading] est alors exécutée ;
  - [errorLoading] qui signale qu'il faut afficher un message d'erreur. La fonction [mShowErrorLoading] est alors exécutée ;
- les pages [nuxt] :
  - font afficher le message d'attente en émettant l'événement ['loading', true] sur le bus d'événements ;
  - cachent le message d'attente en émettant l'événement ['loading', false] sur le bus d'événements ;
  - font afficher un message d'erreur en émettant l'événement ['errorLoading', true] sur le bus d'événements ;
  - cachent le message d'erreur en émettant l'événement ['errorLoading', false] sur le bus d'événements ;

15.12.2 [error]

Le layout [error] affiche un message d'erreur système (non géré par le développeur) :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <!-- mise en page -->
4.    <Layout :left="true" :right="true">
5.      <!-- alerte dans la colonne de droite -->
6.      <template slot="right">
7.        <!-- message sur fond rose -->
8.        <b-alert show variant="danger" align="center">
9.          <h4>L'erreur suivante s'est produite : {{ JSON.stringify(error) }}</h4>
10.       </b-alert>
11.     </template>
12.     <!-- menu de navigation dans la colonne de gauche -->
13.     <Navigation slot="left" />
14.   </Layout>
15. </template>
16.
17. <script>
18. /* eslint-disable no-undef */
19. /* eslint-disable no-console */
20. /* eslint-disable nuxt/no-env-in-hooks */
21.
22. import Layout from '@/components/layout'
23. import Navigation from '@/components/navigation'
24.
25. export default {
26.   name: 'Error',
27.   // composants utilisés
28.   components: {
29.     Layout,
30.     Navigation
31.   },
32.   // propriété [props]
33.   props: { error: { type: Object, default: () => 'waiting ...' } },
34.   // cycle de vie
35.   beforeCreate() {
36.     // client et serveur
37.     console.log('[error beforeCreate]')
38.   },
39.   created() {
40.     // client et serveur
41.     console.log('[error created, error=]', this.error)
42.   },
43.   beforeMount() {
44.     // client seulement
45.     console.log('[error beforeMount]')
46.   },
47.   mounted() {
48.     // client seulement
49.     console.log('[error mounted]')
50.   }
51. }
52. </script>
```

15.13 La page [index] exécutée par le serveur [nuxt]



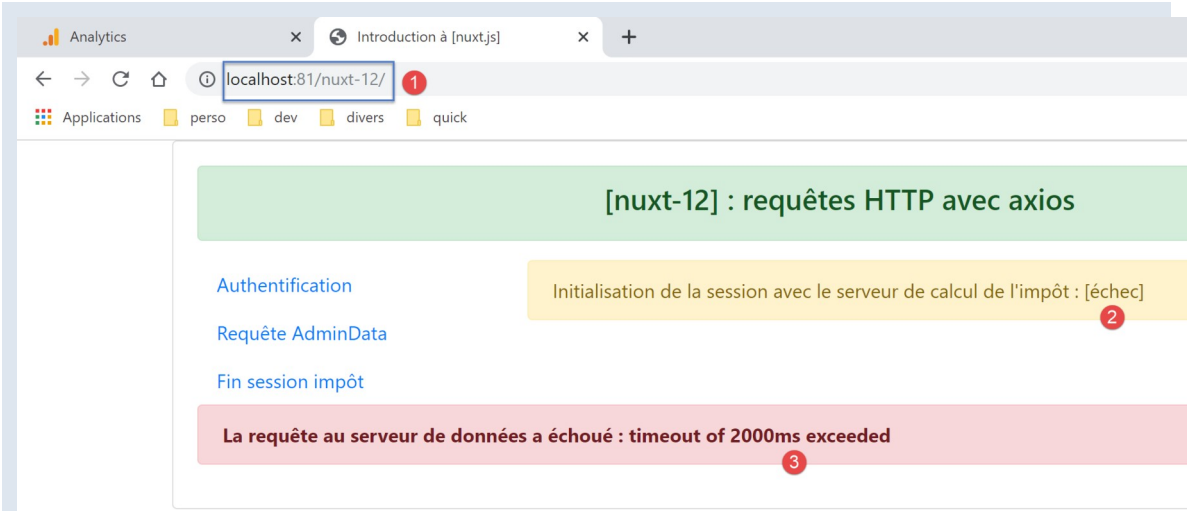
La page [index.vue] a la particularité d'être accessible uniquement via le serveur [nuxt]. Aucun lien n'est présenté à l'utilisateur pour y avoir accès via le client [nuxt]. Son code est le suivant :

```
1.  <!-- page principale -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="warning">Initialisation de la session avec le serveur
    de calcul de l'impôt : {{ result }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13.
14. import Navigation from '@/components/navigation'
15. import Layout from '@/components/layout'
16.
17. export default {
18.   name: 'InitSession',
19.   // composants utilisés
20.   components: {
21.     Layout,
22.     Navigation
23.   },
24.   // données asynchrones
25.   async asyncData(context) {
26.     // log
27.     console.log('[index asyncData started]')
28.     try {
29.       // on démarre une session jSON
30.       const dao = context.app.$dao()
31.       const response = await dao.initSession()
32.       // log
33.       console.log('[index asyncData response=]', response)
34.       // on récupère le cookie de session PHP pour les prochaines requêtes
35.       const phpSessionCookie = dao.getPhpSessionCookie()
36.       // on mémorise le cookie de session PHP dans la session [nuxt]
37.       context.store.commit('replace', { phpSessionCookie })
38.       // y-a-t-il eu erreur ?
39.       if (response.état !== 700) {
40.         // l'erreur se trouve dans response.réponse
41.         throw new Error(response.réponse)
42.       }
43.       // on note le fait que la session jSON a démarré
44.       context.store.commit('replace', { jsonSessionStarted: true })
45.       // on rend le résultat
46.       return { result: '[succès]' }
47.     } catch (e) {
48.       // log
49.       console.log('[index asyncData error=]', e)
50.       // on note le fait que la session jSON n'a pas démarré
51.       context.store.commit('replace', { jsonSessionStarted: false })
52.       // on signale l'erreur
53.       return { result: '[échec]', showErrorLoading: true, errorLoadingMessage: e.message }
```

```
54.     } finally {
55.       // on sauvegarde le store
56.       const session = context.app.$session()
57.       session.save(context)
58.       // log
59.       console.log('[index asyncData finished]')
60.     }
61.   },
62.   // cycle de vie
63.   beforeCreate() {
64.     console.log('[index beforeCreate]')
65.   },
66.   created() {
67.     console.log('[index created]')
68.   },
69.   beforeMount() {
70.     console.log('[index beforeMount]')
71.   },
72.   mounted() {
73.     console.log('[index mounted]')
74.     // client seulement
75.     if (this.showErrorLoading) {
76.       console.log('[index mounted, showErrorLoading=true]')
77.       this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
78.     }
79.   }
80. }
81. </script>
```

- ligne 7 : la page affiche le résultat [result] d'une requête asynchrone (lignes 46 et 51) ;
- ligne 31 : l'opération asynchrone est l'ouverture d'une session jSON avec le serveur de calcul de l'impôt ;
- ligne 25 : on sait que lorsque la page est demandée directement au serveur [nuxt], la fonction [asyncData] n'est exécutée que par le serveur et pas par le client [nuxt] qui s'exécute lorsque le navigateur a reçu la réponse du serveur [nuxt] ;
- ligne 30 : on récupère la couche [dao] dans le contexte du serveur [nuxt] ;
- ligne 35 : si le serveur n'avait pas encore fait de requête au serveur de calcul de l'impôt, il reçoit son premier cookie de session PHP, sinon le dernier cookie de session PHP qu'il a reçu (revoir le code de la couche [dao] du serveur [nuxt] au paragraphe lien) ;
- ligne 37 : on mémorise ce cookie de session PHP dans le store ;
- lignes 39-42 : on regarde si l'opération a réussi. Si ce n'est pas le cas, une exception est lancée qui sera interceptée par le [catch] de la ligne 47 ;
- ligne 44 : on note dans le store que la session jSON avec le serveur PHP est démarrée ;
- ligne 46 : on rend le résultat [result] qui est affiché ligne 7 ;
- lignes 47-54 : on traite une éventuelle exception. Celle-ci peut-être de deux natures :
  - l'opération HTTP de la ligne 31 a échoué sur une erreur de communication serveur [nuxt] / serveur PHP ;
  - l'opération HTTP de la ligne 31 a réussi mais le résultat reçu a signalé une erreur (lignes 39-42) ;
- ligne 51 : on note que la session jSON avec le serveur PHP n'a pas démarré ;
- ligne 53 : on rend le résultat [result] qui est affiché ligne 7. Par ailleurs, on positionne les propriétés [`showErrorLoading`] et [`errorLoadingMessage`] que le client [nuxt] va utiliser pour afficher un message d'erreur lorsqu'il recevra la page envoyée par le serveur [nuxt] (lignes 72-79) ;
- lignes 54-60 : code exécuté dans tous les cas (réussite ou échec) ;
- ligne 56 : on récupère la session [nuxt] dans le contexte du serveur [nuxt] ;
- ligne 57 : on la sauvegarde ;
- lignes 63-68 : une fois la fonction [asyncData] terminée, le serveur [nuxt] exécute les fonctions [beforeCreate] et [create] ;

Note : l'exécution de la page [index] par le serveur [nuxt] peut échouer par exemple si le serveur de calcul de l'impôt n'est pas lancé lorsque l'application [nuxt] est elle lancée :



Dans ce cas, la seule solution est de lancer le serveur de calcul de l'impôt puis l'application [nuxt] elle-même puisque le menu de navigation ne propose pas d'option pour initier une session jSON avec le serveur de calcul de l'impôt ;

15.14 La page [index] exécutée par le client [nuxt]

La page [index] n'est exécutée par le client [nuxt] qu'après que le serveur [nuxt] la lui ait envoyée. Celui-ci lui a envoyé les informations [`result`] et éventuellement [`showErrorLoading`] et [`errorLoadingMessage`].

On sait que la fonction [asyncData] ne sera pas exécutée. Restent alors les fonctions du cycle de vie et notamment la fonction [mounted] :

```
1.  mounted() {
2.      console.log('[index mounted]')
3.      // client seulement
4.      if (this.showErrorLoading) {
5.        console.log('[index mounted, showErrorLoading=true]')
6.        this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
7.      }
8.  }
```

- le client [nuxt] intègre automatiquement dans les propriétés de la page les éléments [result] et éventuellement [showErrorLoading, errorLoadingMessage] que lui a envoyés le serveur [nuxt] :
- la propriété [result] est affichée par la ligne 7 ;
- les propriétés [showErrorLoading, errorLoadingMessage] sont utilisées par la méthode [mounted] : ligne 4, on teste la propriété [showErrorLoading]. Si elle est vraie, on utilise, ligne 6, le bus d'événements du client [nuxt] pour signaler qu'il y a un message d'erreur à afficher ;
- l'événement [errorLoading] lancé ligne 6, est intercepté par la page [layouts/default] décrite au paragraphe lien ;

15.15 La page [authentification] exécutée par le serveur [nuxt]

La page [authentification] est chargée d'identifier un utilisateur auprès du serveur de calcul de l'impôt. Son code est le suivant :

```
1.  <!-- page d'authentification -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="warning">Authentification auprès du serveur de calcul
    de l'impôt : {{ result }} </b-alert>
8.    </Layout>
9.  </template>
10.
```

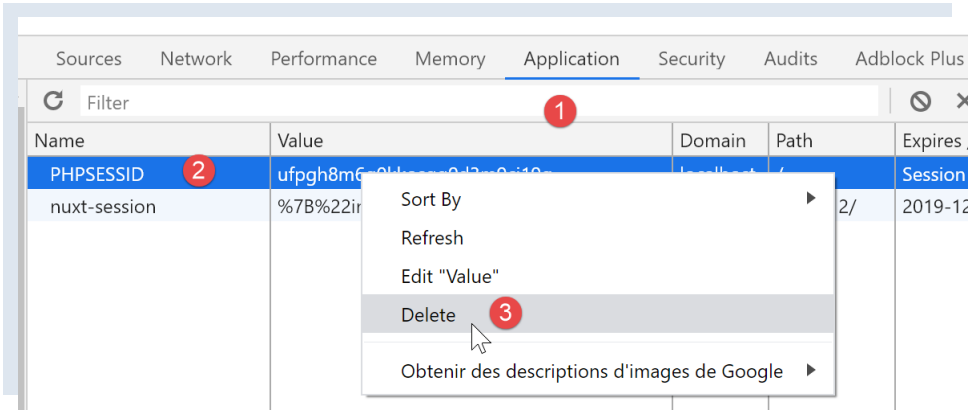
```
<script>
/* eslint-disable no-console */

import Navigation from '@/components/navigation'
import Layout from '@/components/layout'

export default {
  name: 'Authentification',
  // composants utilisés
  components: {
    Layout,
    Navigation
  },
  // données asynchrones
  async asyncData(context) {
    // log
    console.log('[authentification asyncData started]')
    if (process.client) {
      // début attente du client [nuxt]
      context.app.$eventBus().$emit('loading', true)
      // pas d'erreur
      context.app.$eventBus().$emit('errorLoading', false)
    }
    try {
      // on s'authentifie auprès du serveur
      const dao = context.app.$dao()
      const response = await dao.authentifierUtilisateur('admin', 'admin')
      // log
      console.log('[authentification asyncData response=]', response)
      // résultat
      const userAuthenticated = response.état === 200
      // on note le fait que l'utilisateur est authentifié ou pas
      context.store.commit('replace', { userAuthenticated })
      // on sauvegarde le store dans la session [nuxt]
      const session = context.app.$session()
      session.save(context)
      // erreur d'authentification ?
      if (!userAuthenticated) {
        // l'erreur se trouve dans response.réponse
        throw new Error(response.réponse)
      }
      // on rend le résultat
      return { result: '[succès]' }
    } catch (e) {
      // on signale l'erreur
      return { result: '[échec]', showErrorLoading: true, errorLoadingMessage: e.message }
    } finally {
      // log
      console.log('[authentification asyncData finished]')
      if (process.client) {
        // fin attente du client [nuxt]
        context.app.$eventBus().$emit('loading', false)
      }
    }
  },
  // cycle de vie
  beforeCreate() {
    console.log('[authentification beforeCreate]')
  },
  created() {
    console.log('[authentification created]')
  },
  beforeMount() {
    console.log('[authentification beforeMount]')
  },
  mounted() {
    console.log('[authentification mounted]')
    // client seulement
    if (this.showErrorLoading) {
      console.log('[authentification mounted, showErrorLoading=true]')
      this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
    }
  }
}
```

```
85. </script>
```

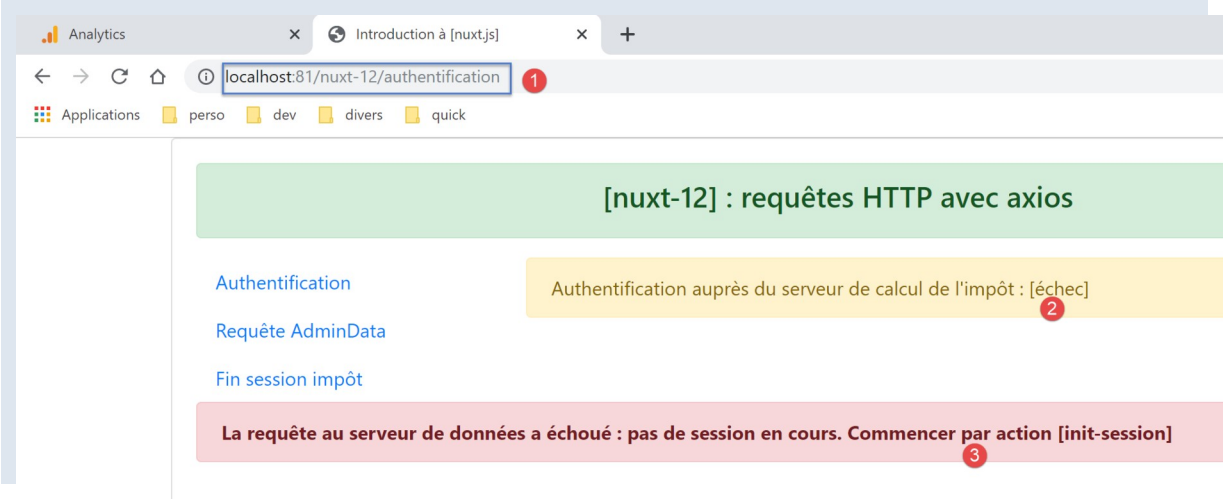
- ligne 7 : la page affiche le résultat [result] de la requête asynchrone [asyncData] des lignes 25-65 ;
- lignes 28-33 : le serveur n'exécute pas ces lignes destinées au client [nuxt] ;
- ligne 36 : on récupère la couche [dao] du serveur [nuxt] ;
- ligne 37 : on s'authentifie auprès du serveur de calcul de l'impôt avec les identifiants de test [admin, admin] qui sont les seuls acceptés par le serveur de calcul de l'impôt ;
- ligne 41 : l'opération d'authentification a réussi si seulement la réponse à l'état 200 ;
- ligne 43 : on met dans le store la propriété [userAuthenticated] ;
- lignes 44-46 : le store est sauvegardé dans la session [nuxt] ;
- lignes 48-51 : si l'authentification a échoué, on lance une exception avec le message d'erreur que le serveur de calcul de l'impôt a envoyé ;
- sinon ligne 53, on retoune un résultat de réussite qui sera affiché ligne 7 ;
- lignes 54-57 : en cas d'erreur on positionne trois propriétés de la page [result, showErrorLoading, errorLoadingMessage]. La propriété [result] sera affichée ligne 7. Les trois propriétés seront envoyées au client [nuxt] ;
- lignes 60-63 : ne sont pas exécutées par le serveur [nuxt] ;
- une fois que [asyncData] a rendu son résultat, celui-ci est affiché ligne 7. Puis les méthodes [beforeCreate] (lignes 67-69) et [created] (lignes 70-72) sont exécutées ;
- c'est fini ;

Note : l'exécution de la page [authentification] par le serveur [nuxt] peut échouer par exemple si la session jSON avec le serveur de calcul de l'impôt n'a pas été initialisée. Cela est possible de la façon suivante :

- supprimez le cookie de session PHP de votre navigateur (pour repartir de zéro) :



- lancez l'application [nuxt] alors que le serveur de calcul n'a pas été lancé : vous obtenez une erreur ;
- lancez le serveur de calcul de l'impôt ;
- demandez l'URL [/authentification] directement dans la barre d'adresses du navigateur :



Dans ce cas, la seule solution est de nouveau de recharger la page [index].

15.16 La page [authentification] exécutée par le client [nuxt]

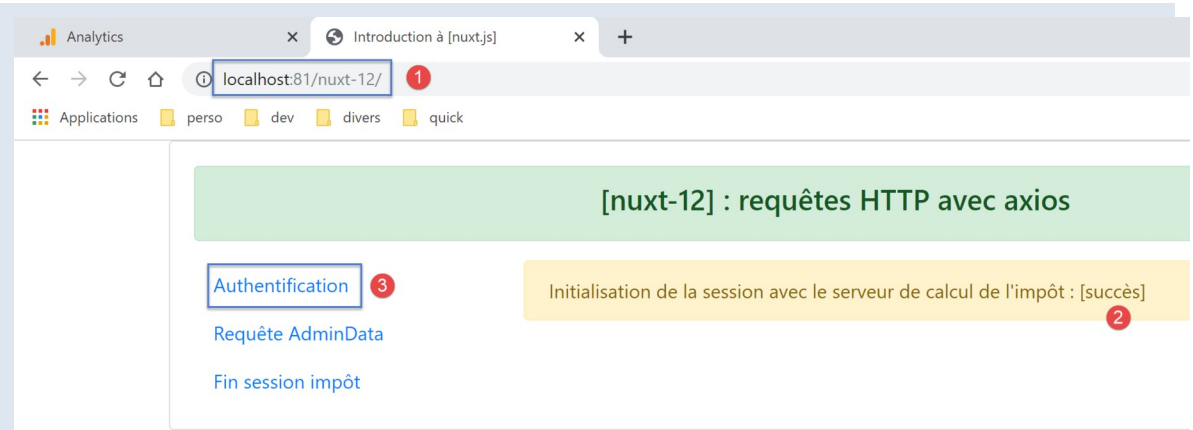
Reprenons le code de la page :

```
1.  <!-- page d'authentification -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="warning">Authentification auprès du serveur de calcul
    de l'impôt : {{ result }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13.
14. import Navigation from '@/components/navigation'
15. import Layout from '@/components/layout'
16.
17. export default {
18.   name: 'Authentification',
19.   // composants utilisés
20.   components: {
21.     Layout,
22.     Navigation
23.   },
24.   // données asynchrones
25.   async asyncData(context) {
26.     // log
27.     console.log('[authentification asyncData started]')
28.     if (process.client) {
29.       // début attente du client [nuxt]
30.       context.app.$eventBus().$emit('loading', true)
31.       // pas d'erreur
32.       context.app.$eventBus().$emit('errorLoading', false)
33.     }
34.     try {
35.       // on s'authentifie auprès du serveur
36.       const dao = context.app.$dao()
37.       const response = await dao.authentifierUtilisateur('admin', 'admin')
38.       // log
39.       console.log('[authentification asyncData response=]', response)
40.       // résultat
41.       const userAuthenticated = response.état === 200
42.       // on note le fait que l'utilisateur est authentifié ou pas
43.       context.store.commit('replace', { userAuthenticated })
```

```
44.       // on sauvegarde le store dans la session [nuxt]
45.       const session = context.app.$session()
46.       session.save(context)
47.       // erreur d'authentification ?
48.       if (!userAuthenticated) {
49.         // l'erreur se trouve dans response.réponse
50.         throw new Error(response.réponse)
51.       }
52.       // on rend le résultat
53.       return { result: '[succès]' }
54.     } catch (e) {
55.       // on signale l'erreur
56.       return { result: '[échec]', showErrorLoading: true, errorLoadingMessage: e.message }
57.     } finally {
58.       // log
59.       console.log('[authentification asyncData finished]')
60.       if (process.client) {
61.         // fin attente du client [nuxt]
62.         context.app.$eventBus().$emit('loading', false)
63.       }
64.     }
65.   },
66.   // cycle de vie
67.   beforeCreate() {
68.     console.log('[authentification beforeCreate]')
69.   },
70.   created() {
71.     console.log('[authentification created]')
72.   },
73.   beforeMount() {
74.     console.log('[authentification beforeMount]')
75.   },
76.   mounted() {
77.     console.log('[authentification mounted]')
78.     // client seulement
79.     if (this.showErrorLoading) {
80.       console.log('[authentification mounted, showErrorLoading=true]')
81.       this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
82.     }
83.   }
84. }
85. </script>
```

Il y a deux cas d'exécution de la page [authentification] par le client [nuxt] :

1. le client [nuxt] s'exécute après que le serveur [nuxt] ait envoyé au navigateur du client [nuxt] la page [authentification] ;
2. le client [nuxt] parce que l'utilisateur a cliqué sur le lien [Authentification] du menu de navigation :



Etudions d'abord le 1er cas. Dans ce cas, le client [nuxt] n'exécute pas la fonction [asyncData]. Il intègre dans les propriétés de la page les éléments [result] et éventuellement [showErrorLoading, errorLoadingMessage] que lui a envoyés le serveur [nuxt] :

- la propriété [result] est affichée par la ligne 7 ;
- les propriétés [showErrorLoading, errorLoadingMessage] sont utilisées par la méthode [mounted] : ligne 79, on teste la propriété [showErrorLoading]. Si elle est vraie, on utilise, ligne 81, le bus d'événements du client [nuxt] pour signaler qu'il y a un message d'erreur à afficher ;

Le mécanisme de l'affichage du message d'erreur a été expliqué pour la page [index] au paragraphe lien.

Le cas 2 est celui du client [nuxt] exécuté lorsque l'utilisateur clique sur le lien [Authentification]. Dans ce cas, le client [nuxt] s'exécute de façon autonome et pas après le serveur [nuxt]. La fonction [asyncData] est alors exécutée. Nous ne donnons que les détails qui diffèrent des explications données pour la page exécutée par le serveur [nuxt] :

- lignes 28-33 : le client [nuxt] demande l'affichage du message d'attente et la disparition d'un éventuel message d'erreur qui aurait été précédemment affiché ;
- ligne 36 : c'est désormais la couche [dao] du client [nuxt] qui est obtenue ici ;
- lignes 60-63 : le client [nuxt] demande la fin de l'affichage du message d'attente ;
- une fois [asyncData] terminée, le cycle de vie de la page va avoir lieu. la fonction [mounted] des lignes 76-83 va être exécutée. S'il y a eu erreur, le message d'erreur va alors être affiché ;

Note : pour provoquer une erreur, suivez la procédure expliquée pour le serveur [nuxt] à la fin du paragraphe lien, mais au lieu de demander la page [authentification] en tapant son URL dans la barre d'adresses, utilisez le lien [Authentification] du menu de navigation. C'est alors le client [nuxt] qui s'exécute.

15.17 La page [get-admindata]

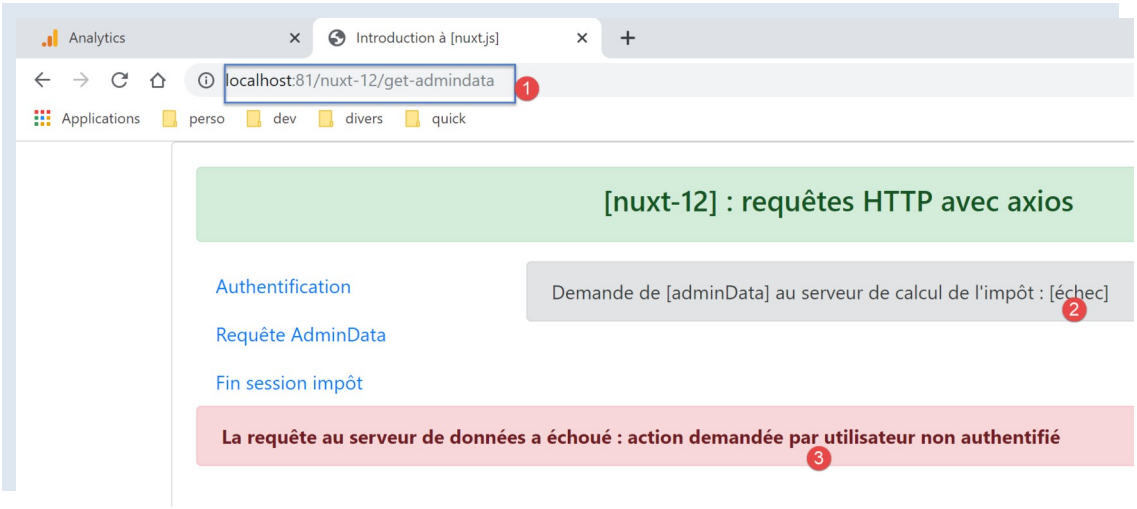
Le code de la page [get-admindata] est le suivant :

```
1.  <!-- vue get-admindata -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-alert slot="right" show variant="secondary"> Demande de [adminData] au serveur de calcul
    de l'impôt : {{ result }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13.
14. import Navigation from '@/components/navigation'
15. import Layout from '@/components/layout'
16.
17. export default {
18.   name: 'GetAdmindata',
19.   // composants utilisés
20.   components: {
21.     Layout,
22.     Navigation
23.   },
24.   // données asynchrones
25.   async asyncData(context) {
26.     // log
27.     console.log('[get-admindata asyncData started]')
28.     if (process.client) {
29.       // début attente
30.       context.app.$eventBus().$emit('loading', true)
31.       // pas d'erreur
32.       context.app.$eventBus().$emit('errorLoading', false)
33.     }
34.     try {
35.       // on demande la donnée [admindata]
36.       const response = await context.app.$dao().getAdminData()
37.       // log
```

```
38.       console.log('[get-admindata asyncData response=]', response)
39.       // résultat
40.       const adminData = response.état === 1000 ? response.réponse : ''
41.       // on met la donnée dans le store
42.       context.store.commit('replace', { adminData })
43.       // on sauvegarde le store dans la session [nuxt]
44.       const session = context.app.$session()
45.       session.save(context)
46.       // y-a-t-il eu erreur ?
47.       if (!adminData) {
48.         // l'erreur se trouve dans response.réponse
49.         throw new Error(response.réponse)
50.       }
51.       // on rend la valeur reçue
52.       return { result: adminData }
53.     } catch (e) {
54.       // on signale l'erreur
55.       return { result: '[échec]', showErrorLoading: true, errorLoadingMessage: e.message }
56.     } finally {
57.       // log
58.       console.log('[get-admindata asyncData finished]')
59.       if (process.client) {
60.         // fin attente
61.         context.app.$eventBus().$emit('loading', false)
62.       }
63.     }
64.   },
65.   // cycle de vie
66.   beforeCreate() {
67.     console.log('[get-admindata beforeCreate]')
68.   },
69.   created() {
70.     console.log('[get-admindata created]')
71.   },
72.   beforeMount() {
73.     console.log('[get-admindata beforeMount]')
74.   },
75.   mounted() {
76.     console.log('[get-admindata mounted]')
77.     // client
78.     if (this.showErrorLoading) {
79.       console.log('[get-admindata mounted, showErrorLoading=true]')
80.       this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
81.     }
82.   }
83. }
84. </script>
```

Cette page est très semblable à la page [authentification]. Les explications sont analogues aussi bien pour son exécution par le serveur [nuxt] que pour son exécution par le client [nuxt]. Notons cependant que la ligne 7 affiche non pas succès / échec comme précédemment mais la valeur de la donnée reçue du serveur de calcul de l'impôt (ligne 52) :

Le résultat ci-dessus est obtenu aussi bien avec le serveur qu'avec le client [nuxt]. Pour provoquer une erreur, demandez la page [get-admindata], via le serveur ou le client [nuxt], sans être authentifié :



15.18 La page [fin-session]

Le code de la page est le suivant :

```
<!-- page principale -->
<template>
  <Layout :left="true" :right="true">
    <!-- navigation -->
    <Navigation slot="left" />
    <!-- message-->
    <b-alert slot="right" show variant="warning">Fin de la session avec le serveur de calcul de l'impôt : {{ result }} </b-alert>
  </Layout>
</template>

<script>
/* eslint-disable no-console */

import Navigation from '@/components/navigation'
import Layout from '@/components/layout'

export default {
  name: 'FinSession',
  // composants utilisés
  components: {
    Layout,
    Navigation
  },
  // données asynchrones
  async asyncData(context) {
    // log
    console.log('[fin-session asyncData started]')
    // cas du client [nuxt]
    if (process.client) {
      // début attente
      context.app.$eventBus().$emit('loading', true)
      // pas d'erreur
      context.app.$eventBus().$emit('errorLoading', false)
    }
    try {
      // on demande une nouvelle session PHP au serveur de calcul de l'impôt
      const dao = context.app.$dao()
      const response = await dao.finSession()
      // log
      console.log('[fin-session asyncData response=]', response)
      // y-at-il eu erreur ?
      if (response.état !== 400) {
        // l'erreur se trouve dans response.réponse
        throw new Error(response.réponse)
      }
      // le serveur a envoyé un nouveau cookie de session PHP
```

```
47.       // on le récupère à la fois pour le serveur et le client nuxt
48.       // si ce code est exécuté par le client [nuxt], le cookie de session PHP doit être mis dans la
    session nuxt
49.       // pour que le plugin [plgDao] du serveur [nuxt] puisse le récupérer et initialiser la couche [dao]
    avec
50.       // si ce code est exécuté par le serveur [nuxt], le cookie de session PHP doit être mis dans la
    session nuxt
51.       // pour que le routing du client [nuxt] le récupère et le passe au navigateur
52.       const phpSessionCookie = dao.getPhpSessionCookie()
53.       // on note dans le store le fait que la session jSON est démarrée et on mémorise le cookie de
    session PHP
54.       context.store.commit('replace', { jsonSessionStarted: true, phpSessionCookie, userAuthenticated:
    false, adminData: '' })
55.       // on sauvegarde le store dans la session [nuxt]
56.       const session = context.app.$session()
57.       session.save(context)
58.       // on rend le résultat
59.       return { result: "[succès]. La session jSON reste initialisée mais vous n'êtes plus authentifié(e)."
    }
60.     } catch (e) {
61.       // log
62.       console.log('[fin-session asyncData error=]', e)
63.       // on signale l'erreur
64.       return { result: '[échec]', showErrorLoading: true, errorLoadingMessage: e.message }
65.     } finally {
66.       // log
67.       console.log('[fin-session asyncData finished]')
68.       if (process.client) {
69.         // fin attente
70.         context.app.$eventBus().$emit('loading', false)
71.       }
72.     }
73.   },
74.   // cycle de vie
75.   beforeCreate() {
76.     console.log('[fin-session beforeCreate]')
77.   },
78.   created() {
79.     console.log('[fin-session created]')
80.   },
81.   beforeMount() {
82.     console.log('[fin-session beforeMount]')
83.   },
84.   mounted() {
85.     console.log('[fin-session mounted]')
86.     // client seulement
87.     if (this.showErrorLoading) {
88.       console.log('[fin-session mounted, showErrorLoading=true]')
89.       this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
90.     }
91.   }
92. }
93. </script>
```

Le code est très analogue à celui des pages précédentes et les explications sont les mêmes. Il faut simplement s'attarder sur un point : l'opération asynchrone de la ligne 38, fait que le serveur de calcul de l'impôt va envoyer un nouveau cookie de session PHP. Les explications pour la gestion de ce cookie diffèrent selon que c'est le serveur ou le client [nuxt] qui exécute ce code.

Commençons par le serveur [nuxt] :

- ligne 37 : c'est la couche [dao] du serveur [nuxt] qui est instanciée. Rappelons le code de son constructeur :

```
1.  // constructeur
2.    constructor(axios, phpSessionCookie) {
3.      // bibliothèque axios
4.      this.axios = axios
5.      // valeur du cookie de session
6.      this.phpSessionCookie = phpSessionCookie
7.      // nom du cookie de session du serveur PHP
8.      this.phpSessionCookieName = 'PHPSESSID'
9.  }
```

  On voit ligne 1, que le constructeur a besoin du cookie de session PHP du moment, le dernier reçu, que ce soit par le serveur ou le client [nuxt] ;
- ligne 52 : le serveur [nuxt] récupère le cookie de la nouvelle session PHP ou bien l'ancien cookie si l'opération de fin de session a échoué ;

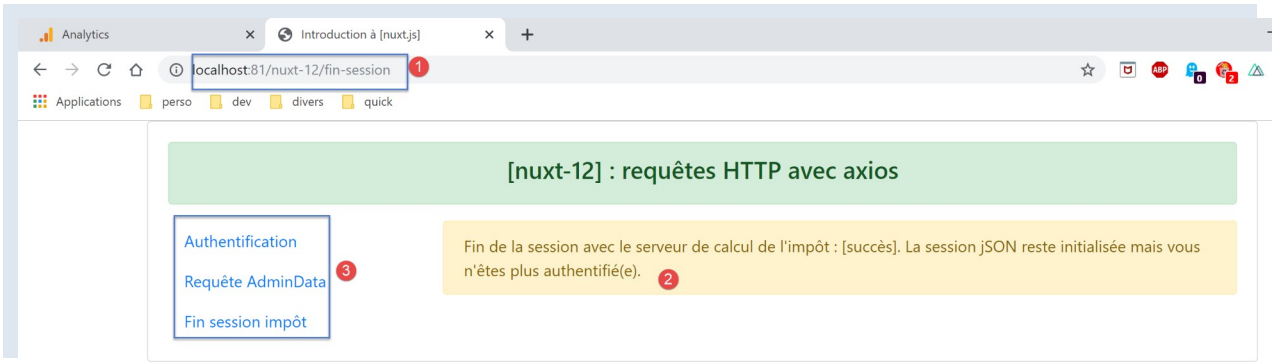
- ligne 54 : le cookie de session PHP est mis dans le store puis sauvegardé dans la session [nuxt] aux lignes 56-57 ;
- après le serveur c'est le client [nuxt] qui exécute la page [fin-session] avec les données envoyées par le serveur. On sait qu'il ne va pas exécuter la fonction [asyncData] ;
- au final, après que serveur et client [nuxt] ont terminé leur travail, on sait que le cookie PHP nécessaire aux échanges avec le serveur de calcul de l'impôt est dans la session [nuxt] ;

Le fait que le cookie PHP soit dans la session [nuxt] est suffisant pour le serveur, car c'est là que va le prendre sa couche [dao]. Dans le plugin [server/plgDao] qui initialise la couche [dao] du serveur, on a écrit :

```
1.  /* eslint-disable no-console */
2.  // on crée un point d'accès à la couche [Dao]
3.  import Dao from '@/api/server/Dao'
4.  export default (context, inject) => {
5.    // configuration axios
6.    context.$axios.defaults.timeout = context.env.timeout
7.    context.$axios.defaults.baseURL = context.env.baseURL
8.    // on récupère le cookie de session
9.    const store = context.app.$session().value.store
10.   const phpSessionCookie = store ? store.phpSessionCookie : ''
11.   console.log('session=', context.app.$session().value, 'phpSessionCookie=', phpSessionCookie)
12.   // instanciation de la couche [dao]
13.   const dao = new Dao(context.$axios, phpSessionCookie)
14.   // injection d'une fonction [$dao] dans le contexte
15.   inject('dao', () => dao)
16.   // log
17.   console.log('[fonction server $dao créée]')
18. }
```

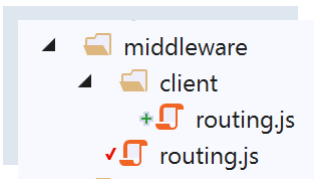
- ligne 13, la couche [dao] du serveur [nuxt] est instanciée avec le cookie de session PHP pris dans la session [nuxt], lignes 9-10 ;

Pour le client [nuxt], c'est une autre histoire. Ce n'est pas lui en effet qui envoie le cookie mais le navigateur qui l'exécute. Or ce navigateur ne connaît pas le cookie de la nouvelle session PHP reçu par le serveur [nuxt]. Si on utilise les liens du menu de navigation [3] :



Le serveur de calcul de l'impôt va recevoir du navigateur un cookie de session PHP obsolète et il va répondre qu'à ce cookie aucune session jSON n'est associée. Il nous faut trouver le moyen de passer au navigateur le nouveau cookie de session PHP.

On peut utiliser un middleware de routing pour ce faire :



Le script [client/routing] est le middleware de routage déclaré dans le fichier [nuxt.config] :

```
1.  // routeur
2.    router: {
3.      // racine des URL de l'application
4.      base: '/nuxt-12/',
5.      // middleware de routage
6.      middleware: ['routing']
7.  },
```

Le script [middleware/routing] est le suivant :

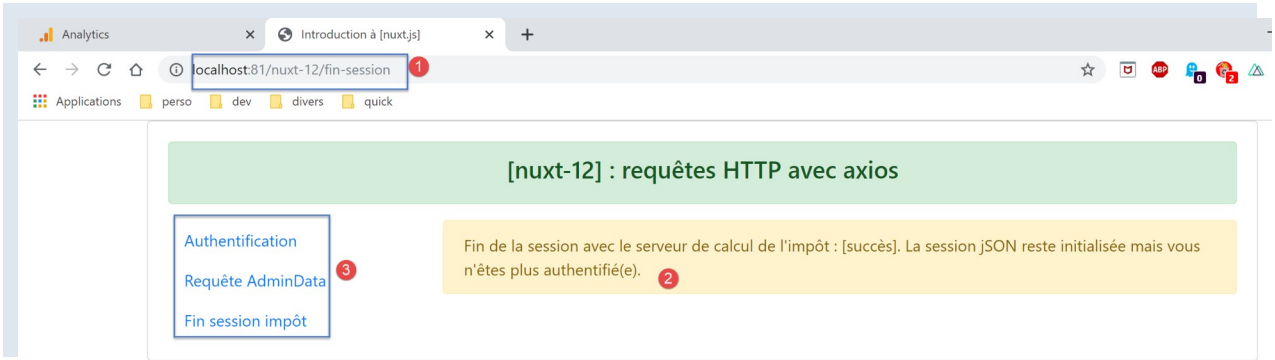
```
1.  /* eslint-disable no-console */
2.
3.  // on importe le middleware du client
4.  import clientRouting from './client/routing'
5.
6.  export default function(context) {
7.    // qui exécute ce code ?
8.    console.log('[middleware], process.server', process.server, ', process.client=',
    process.client)
9.    if (process.client) {
10.     // routage client
11.     clientRouting(context)
12.   }
13. }
```

- lignes 9-12 : on ne route que le client avec une fonction importée ligne 4 ;

Le script [middleware/client/routing] est le suivant :

```
1.  /* eslint-disable no-console */
2.  export default function(context) {
3.    // qui exécute ce code ?
4.    console.log('[middleware client], process.server', process.server, ', process.client=',
    process.client)
5.    // gestion du cookie de la session PHP dans le navigateur
6.    // le cookie de la session PHP du navigateur doit être identique à celui trouvé en session
    nuxt
7.    // l'acion [fin-session] reçoit un nouveau cookie PHP (serveur comme client nuxt)
8.    // si c'est le serveur qui le reçoit, le client doit le transmettre au navigateur
9.    // pour ses propres échanges avec le serveur PHP
10.   // on est ici dans un routing client
11.
12.   // on récupère le cookie de la session PHP
13.   const phpSessionCookie = context.store.state.phpSessionCookie
14.   if (phpSessionCookie) {
15.     // s'il existe, on affecte le cookie de session PHP au navigateur
16.     document.cookie = phpSessionCookie
17.   }
18. }
```

Revenons à la situation juste après l'exécution de la page [fin-session] par le serveur [nuxt] :



Si on clique sur l'un des liens du menu [3], le client [nuxt] va prendre la main. Comme il va y avoir changement de page, le script de routing du client va s'exécuter :

- ligne 13 : le cookie de session PHP est trouvé dans le store de l'application [nuxt] ;
- ligne 14 : s'il n'est pas vide on le transmet au navigateur (ligne 16). A partir de ce moment le navigateur du client [nuxt] a le bon cookie de session PHP ;

Le script [client/routing] est exécuté à chaque changement de page du client [nuxt]. Le code du script est valide quelque soit la page cible : simplement, la plupart du temps, il donne au navigateur un cookie de session PHP qu'il a déjà, sauf dans deux cas :

- juste après le démarrage de l'application, le serveur [nuxt] exécute la page [index] et reçoit un 1er cookie de session PHP que le navigateur du client [nuxt] n'a pas ;
- lorsque le serveur [nuxt] exécute la page [fin-session] comme il vient d'être expliqué ;

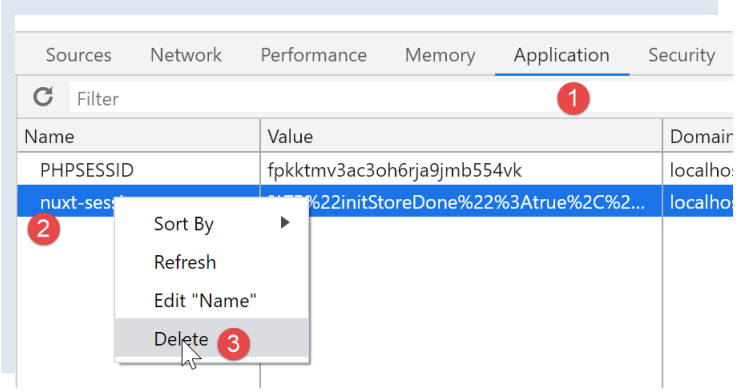
Maintenant étudions le cas où la page [fin-session] est exécutée par le client [nuxt] uniquement, parce qu'on a cliqué sur son lien dans le menu de navigation. C'est désormais le client [nuxt] qui exécute la fonction [asyncData] :

```
1.  try {
2.        // on demande une nouvelle session PHP au serveur de calcul de l'impôt
3.        const dao = context.app.$dao()
4.        const response = await dao.finSession()
5.        // log
6.        console.log('[fin-session asyncData response=]', response)
7.        // y-at-il eu erreur ?
8.        if (response.état !== 400) {
9.          // l'erreur se trouve dans response.réponse
10.         throw new Error(response.réponse)
11.       }
12.       // le serveur a envoyé un nouveau cookie de session PHP
13.       // on le récupère à la fois pour le serveur et le client nuxt
14.       // si ce code est exécuté par le client [nuxt], le cookie de session PHP doit être mis dans la session
    nuxt
15.       // pour que le plugin [plgDao] du serveur [nuxt] puisse le récupérer et initialiser la couche [dao]
    avec
16.       // si ce code est exécuté par le serveur [nuxt], le cookie de session PHP doit être mis dans la
    session nuxt
17.       // pour que le routing du client [nuxt] le récupère et le passe au navigateur
18.       const phpSessionCookie = dao.getPhpSessionCookie()
19.       // on note dans le store le fait que la session jSON est démarrée et on mémorise le cookie de session
    PHP
20.       context.store.commit('replace', { jsonSessionStarted: true, phpSessionCookie, userAuthenticated:
    false, adminData: '' })
21.       // on sauvegarde le store dans la session [nuxt]
22.       const session = context.app.$session()
23.       session.save(context)
24.       // on rend le résultat
25.       return { result: "[succès]. La session jSON reste initialisée mais vous n'êtes plus authentifié(e)." }
26.     } catch (e) {
27.       // log
28.       console.log('[fin-session asyncData error=]', e)
29.       // on signale l'erreur
30.       return { result: '[échec]', showErrorLoading: true, errorLoadingMessage: e.message }
31.     } finally {
32.       // log
33.       console.log('[fin-session asyncData finished]')
34.       if (process.client) {
35.         // fin attente
36.         context.app.$eventBus().$emit('loading', false)
37.       }
38.     }
```

- ligne 3 : c'est la couche [dao] du client [nuxt] qui est obtenu ici ;
- ligne 18 : le cookie de session PHP récupéré par la couche [dao] du client [nuxt] est mémorisé, mis dans le store (ligne 20) puis sauvegardé en session [nuxt] (lignes 22-23) ;
- à partir de là tout va bien car on sait que la couche [dao] du serveur [nuxt] va chercher le cookie de session PHP dans la session [nuxt] ;

15.19 Exécution

Pour exécuter cet exemple, il faut prendre soin avant l'exécution de supprimer le cookie de session [nuxt] et le cookie PHP du navigateur exécutant le client [nuxt] afin de partir d'une situation nette. Ci-dessous un exemple avec le navigateur Chrome :



15.20 Conclusion

Cet exemple a été particulièrement complexe. Il réunissait des connaissances acquises dans les exemples précédents : persistance du store dans une session [nuxt], plugins d'injections de fonctions, middleware de routage, gestion des erreurs des opérations asynchrones. La complexité a été accrue par le fait qu'on voulait que l'utilisateur puisse aussi bien utiliser les liens du menu de navigation que taper des URL à la main sans que ça casse l'application. Pour cela, on a été obligés de regarder comment se comportait chaque page selon qu'elle était exécutée par le client ou le serveur [nuxt].

Cette unité de comportement du client et du serveur [nuxt] n'est pas indispensable. On peut se mettre dans le cas fréquent où :

- la première page est délivrée par le serveur [nuxt] ;
- toutes les pages suivantes sont délivrées par le client [nuxt] qui travaille alors en mode [SPA] ;

Néanmoins même dans ce cas, il faut vérifier ce que donne l'exécution de toutes les pages par le serveur [nuxt] car c'est ce qu'obtiendront les moteurs de recherche qui les demanderont.

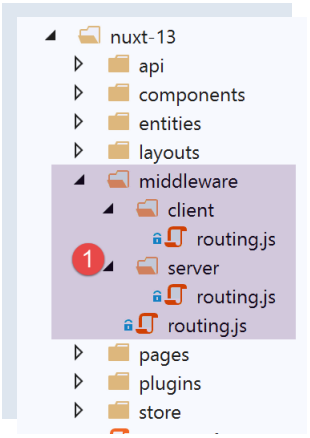
16 Exemple [nuxt-13] : contrôle de la navigation de [nuxt-12]

Dans cet exemple, nous nous intéressons à la navigation de [nuxt-12]. On ne l'a pas fait dans [nuxt-12] parce que le contrôle de la navigation aurait complexifié un exemple déjà complexe.

Objectif : nous voulons que l'utilisateur ne puisse faire que des actions autorisées :

- si la session jSON n'a pas démarré, alors seule l'URL [/] est autorisée ;
- si la session jSON a démarré mais que l'utilisateur n'est pas authentifié, alors seule l'URL [/authentification] est autorisée ;
- si la session jSON a démarré et que l'utilisateur est authentifié, alors seules les URL [/get-admindata, /fin-session] sont autorisées ;
- lorsque la cible du routage du moment n'est pas autorisée, alors on procèdera à une redirection vers une URL autorisée ;

L'exemple [nuxt-13] est obtenu initialement par recopie de l'exemple [nuxt-12] :



C'est dans le dossier du routage [middleware] que vont prendre place les modifications.

16.1 Routage de l'application [nuxt]

Le routage de l'application est configuré de la façon suivante dans le fichier [nuxt.config] :

```
1.  // routeur
2.    router: {
3.      // racine des URL de l'application
4.      base: '/nuxt-13/',
5.      // middleware de routage
6.      middleware: ['routing']
7.  },
```

- ligne 6 : le routage de l'application est contrôlé par le fichier [middleware/routing] ;

Le fichier [middleware/routing] est le suivant :

```
1.  /* eslint-disable no-console */
2.
3.  // on importe les middleware du serveur et du client
4.  import serverRouting from './server/routing'
5.  import clientRouting from './client/routing'
6.
7.  export default function(context) {
8.    // qui exécute ce code ?
9.    console.log('[middleware], process.server', process.server, ', process.client=',
    process.client)
10.   if (process.server) {
11.     // routage serveur
```

```
12.     serverRouting(context)
13.   } else {
14.     // routage client
15.     clientRouting(context)
16.   }
17. }
```

- lignes 10-16 : on traite différemment les routages du client et du serveur [nuxt]. C'est un point où ils diffèrent grandement ;
- ligne 4 : le routage du serveur est implémenté par le script [middleware/server/routing] ;
- ligne 5 : le routage du client est implémenté par le script [middleware/client/routing] ;

16.2 Routage du client [nuxt]

Le routage du client [nuxt] reste ce qu'il était dans [nuxt-12] :

```
1.  /* eslint-disable no-console */
2.  export default function(context) {
3.    // qui exécute ce code ?
4.    console.log('[middleware client], process.server', process.server, ', process.client=',
    process.client)
5.    // gestion du cookie de la session PHP dans le navigateur
6.    // le cookie de la session PHP du navigateur doit être identique à celui trouvé en session
    nuxt
7.    // l'acion [fin-session] reçoit un nouveau cookie PHP (serveur comme client nuxt)
8.    // si c'est le serveur qui le reçoit, le client doit le transmettre au navigateur
9.    // pour ses propres échanges avec le serveur PHP
10.   // on est ici dans un routing client
11.
12.   // on récupère le cookie de la session PHP
13.   const phpSessionCookie = context.store.state.phpSessionCookie
14.   if (phpSessionCookie) {
15.     // s'il existe, on affecte le cookie de session PHP au navigateur
16.     document.cookie = phpSessionCookie
17.   }
18.
19.   ...
20. }
```

Pour éviter que le client aille dans des routes non autorisées on va simplement lui offrir dans le menu de navigation client les seules routes autorisées. Le composant [components/navigation] devient le suivant :

```
1.  <template>
2.    <!-- menu Bootstrap à trois options -->
3.    <b-nav vertical>
4.      <b-nav-item v-if="$store.state.jsonSessionStarted && !$store.state.userAuthenticated"
    to="/authentification" exact exact-active-class="active">
5.        Authentification
6.      </b-nav-item>
7.      <b-nav-item
8.        v-if="$store.state.jsonSessionStarted && $store.state.userAuthenticated && !
    $store.state.adminData"
9.        to="/get-admindata"
10.       exact
11.       exact-active-class="active"
12.     >
13.       Requête AdminData
14.     </b-nav-item>
15.     <b-nav-item v-if="$store.state.jsonSessionStarted && $store.state.userAuthenticated"
    to="/fin-session" exact exact-active-class="active">
16.       Fin session impôt
17.     </b-nav-item>
18.   </b-nav>
19. </template>
```

- ligne 4 : l'option [Authentification] n'est offerte que si la session jSON a démarré mais que l'utilisateur n'est pas authentifié. Si la session jSON n'a pas démarré ou que l'utilisateur est déjà authentifié alors l'option n'est pas offerte ;
- lignes 7-11 : l'option [Requête AdminData] n'est offerte que si la session jSON a démarré, que l'utilisateur est authentifié et qu'on n'a pas encore récupéré la donnée [AdminData]. Si l'une de ces trois conditions n'est pas

satisfaite (session jSON pas démarrée, utilisateur pas authentifié ou la donnée [AdminData] déjà récupérée, l'option n'est pas offerte ;
- ligne 15 : l'option [Fin session impôt] est offerte dès que la session jSON a démarré et que l'utilisateur est authentifié, sinon elle ne l'est pas ;

16.3 Routage du serveur [nuxt]

Le routage du serveur est en général plus complexe que celle du client car l'utilisateur peut taper n'importe quelle URL dans la barre d'adresses de son navigateur. On peut laisser faire (après tout l'utilisateur n'est pas censé faire ça) ou essayer de contrôler les choses. C'est ce que nous allons faire ici, pour l'exemple, car dans le cas de l'application [nuxt-12], on peut très bien s'en passer puisque le serveur de calcul de l'impôt est bien protégé contre ces URL à la main et sait envoyer les messages d'erreur adéquats. Nous l'avons vu dans [next-12] où il n'y avait aucun contrôle de routage.

Le routage d'un serveur [nuxt] est très différent d'un client [nuxt] quant à la notion de redirection :

- lorsque un serveur [nuxt] est redirigé, il envoie un ordre de redirection au navigateur client avec la cible de la redirection. Le navigateur fait alors une nouvelle requête au serveur [nuxt] en lui demandant la cible qui lui a été transmise. Tout se passe comme si l'utilisateur avait tapé à la main l'URL de la cible de la redirection : toute l'application [nuxt] redémarre et donc tout son cycle de vie (plugins serveur, store, routage serveur, pages) ;
- lorsqu'un client [nuxt] est redirigé, rien de tel n'arrive. Il y a un simple changement de page, le même que celui qui aurait été obtenu si l'utilisateur avait cliqué sur un lien menant à la cible de la redirection. Le cycle de vie est alors différent (routage client, affichage cible de la route) ;

Pour cette raison, il est préférable de séparer le routage client du routage serveur même si les deux codes peuvent paraître analogues.

Le script de routage du serveur [middleware/server/routing] sera le suivant :

```
1.  /* eslint-disable no-console */
2.  export default function(context) {
3.    // qui exécute ce code ?
4.    console.log('[middleware server], process.server', process.server, ', process.client=',
    process.client)
5.
6.    // on récupère quelques informations dans le store [nuxt]
7.    const store = context.store
8.    // d'où vient-on ?
9.    const from = store.state.from || 'nowhere'
10.   ...
11. }
```

- dans le routage du client, la fonction de routage reçoit le contexte [context] avec la propriété [context.from] qui est la route de la page d'où l'on vient. La route où l'on va est obtenue par [context.route] ;
- dans le routage du serveur, la fonction de routage reçoit le contexte [context] **sans** la propriété [context.from]. Le routage du serveur n'intervient que lorsqu'une URL est demandée à la main au serveur [nuxt]. On sait qu'alors toute l'application [nuxt] est réinitialisée. C'est comme si on repartait de zéro et il n'y a donc pas de notion de 'page précédente' ;
- grâce à la session [nuxt] on sait que le serveur peut récupérer cette session et donc ne pas repartir de zéro. C'est donc dans cette session [nuxt] et plus particulièrement dans le store de cette session que nous stockerons le nom de la dernière page affichée par le navigateur client avant qu'une URL soit demandée au serveur [nuxt] ;
- lignes 7-9 : on récupère le nom de la dernière page affichée par le navigateur client. Au démarrage de l'application, cette information [from] n'existe pas dans le store. On affecte alors le nom [nowhere] à la variable [from] ;

Pour que le serveur [nuxt] puisse récupérer dans le store le nom de la dernière page affichée par le navigateur client, il faut que le client [nuxt] mette également cette information dans le store. Le script de routage du client [nuxt] est donc complété de la façon suivante :

```
1.  /* eslint-disable no-console */
2.  export default function(context) {
3.    // qui exécute ce code ?
```

```
4.    console.log('[middleware client], process.server', process.server, ', process.client=',
    process.client)
5.    // gestion du cookie de la session PHP dans le navigateur
6.    // le cookie de la session PHP du navigateur doit être identique à celui trouvé en session
    nuxt
7.    // l'acion [fin-session] reçoit un nouveau cookie PHP (serveur comme client nuxt)
8.    // si c'est le serveur qui le reçoit, le client doit le transmettre au navigateur
9.    // pour ses propres échanges avec le serveur PHP
10.   // on est ici dans un routing client
11.
12.   // on récupère le cookie de la session PHP
13.   const phpSessionCookie = context.store.state.phpSessionCookie
14.   if (phpSessionCookie) {
15.     // s'il existe, on affecte le cookie de session PHP au navigateur
16.     document.cookie = phpSessionCookie
17.   }
18.
19.   // on met dans la session le nom de la page où on va - pas de redirection serveur
20.   context.store.commit('replace', { serverRedirection: false, from: context.route.name })
21.   // on sauvegarde le store dans la session [nuxt]
22.   const session = context.app.$session()
23.   session.value.store = context.store.state
24.   session.save(context)
25. }
```

- les lignes 19-24 sont ajoutées ;
- ligne 20 : on met dans le store le nom de la page [`context.route.name`] qui va s'afficher et qui sera donc lors du routage suivant, la page d'où l'on vient. Par ailleurs, on va voir que dans le routage du serveur [nuxt], celui-ci a besoin de savoir si le routage en cours est issu d'une précédente redirection du serveur [nuxt]. Ici ce n'est pas le cas, et on met donc la propriété [`serverRedirection`] à [false] ;
- lignes 22-24 : l'état du store est mis dans la session [nuxt] (ligne 23) puis la session [nuxt] est sauvegardée dans un cookie (ligne 24) qui lui-même sera sauvegardé dans le navigateur du client [nuxt] ;

Revenons sur le script de routage du serveur [nuxt] :

```
1.  /* eslint-disable no-console */
2.  export default function(context) {
3.    // qui exécute ce code ?
4.    console.log('[middleware server], process.server', process.server, ', process.client=',
    process.client)
5.
6.    // on récupère quelques informations dans le store [nuxt]
7.    const store = context.store
8.    // d'où vient-on ?
9.    const from = store.state.from || 'nowhere'
10.   // où va-t-on ?
11.   const to = context.route.name
12.   // éventuelle redirection
13.   let redirection = ''
14.   // gestion du routage terminé
15.   let done = false
16.
17.   // est-on déjà dans une redirection du serveur [nuxt]?
18.   if (store.state.serverRedirection) {
19.     // rien à faire
20.     done = true
21.   }
22.
23.   // est-ce un rechargement de page ?
24.   if (to === from) {
25.     // rien à faire
26.     done = true
27.   }
28.
29.   // contrôle de la navigation du serveur [nuxt]
30.   // on s'inspire de la navigation client dans le composant [navigation]
31.
32.   // cas où la session PHP n'a pas démarré
33.   if (!done && !store.state.jsonSessionStarted && to !== 'index') {
34.     // redirection
35.     redirection = 'index'
36.     // travail terminé
```

```
37.     done = true
38.   }
39.
40.   // cas où l'utilisateur n'est pas authentifié
41.   if (!done && store.state.jsonSessionStarted && !store.state.userAuthenticated && to !==
    'authentification') {
42.     // redirection
43.     redirection = from
44.     // travail terminé
45.     done = true
46.   }
47.
48.   // cas où l'utilisateur a été authentifié
49.   if (!done && store.state.jsonSessionStarted && store.state.userAuthenticated && to !== 'get-
    admindata' && to !== 'fin-session') {
50.     // on reste sur la même page
51.     redirection = from
52.     // travail terminé
53.     done = true
54.   }
55.
56.   // cas où [adminData] a été obtenu
57.   if (!done && store.state.jsonSessionStarted && store.state.userAuthenticated &&
    store.state.adminData && to !== 'fin-session') {
58.     // on reste sur la même page
59.     redirection = from
60.     // travail terminé
61.     done = true
62.   }
63.
64.   // on a fait tous les contrôles ---------------------
65.   // redirection ?
66.   if (redirection) {
67.     // on note la redirection dans le store
68.     store.commit('replace', { serverRedirection: true })
69.   } else {
70.     // pas de redirection
71.     store.commit('replace', { serverRedirection: false, from: to })
72.   }
73.   // on sauvegarde le store dans la session [nuxt]
74.   const session = context.app.$session()
75.   session.value.store = store.state
76.   session.save(context)
77.   // on fait l'éventuelle redirection
78.   if (redirection) {
79.     context.redirect({ name: redirection })
80.   }
81. }
```

- lignes 6-9 : on récupère la valeur de [from] dans le store du serveur [nuxt] ;
- ligne 11 : on note la cible du routage courant ;
- ligne 13 : le routage peut amener à une redirection du navigateur client. [redirection] sera la cible de cette redirection ;
- ligne 15 : [done] à [true] indique que le routage est terminé ;
- lignes 17-21 : on regarde d'abord si le routage courant est issu d'une demande de redirection envoyée au navigateur client. Cette information est stockée dans la propriété [serverRedirection] du store. Si cette propriété est à vrai, alors c'est que le serveur [nuxt] a envoyé une redirection au navigateur client lors de la précédente requête au serveur [nuxt]. Dans ce cas, il n'y a pas de routage à faire. Lors de la précédente requête, le routeur du serveur [nuxt] a décidé que le navigateur client devait être redirigé. Cette décision n'a pas à être remise en cause par un nouveau routage ;
- lignes 23-27 : on regarde si le routage en cours est un rechargement de page. Si oui, on laisse faire ;
- à partir de la ligne 29, on reprend les règles appliquées dans le composant [navigation] du client [nuxt] (cf paragraphe précédent) ;
- lignes 32-38 : on traite le cas où la session jSON n'a pas démarré et que la cible du routage n'est pas la page [index]. Dans ce cas, on redirige le navigateur client vers la page [index] ;
- lignes 40-46 : on traite le cas où la session jSON a démarré, l'utilisateur n'est pas authentifié et la cible du routage courant n'est pas la page [authentification]. Dans ce cas, on refuse le routage et on reste là où on était ;

- lignes 48-54 : on traite le cas où la session jSON a démarré, l'utilisateur est authentifié et la cible du routage courant n'est ni la page [get-admindata], ni la page [fin-session] qui sont alors les seules destinations possibles. Dans ce cas, on refuse le routage demandé et on revient là où on était précédemment ;
- lignes 56-62 : on traite le cas où [adminData] a été obtenu. Dans ce cas, il n'y a qu'une cible possible pour le routage : la page [fin-session]. Si ce n'était pas elle qui était demandée, on refuse le routage et on revient là où on était précédemment ;
- lignes 64-72 : s'il y a eu redirection, on le note dans le store du serveur [nuxt] : [`serverRedirection: true`]. On notera qu'on ne donne pas de valeur à la propriété [from] du store. La raison en est qu'il va y avoir redirection du navigateur client et on a vu que dans ce cas, il n'y avait pas de routage (lignes 17-20) et la propriété [from] du store n'est pas utilisée ;
- lignes 66-69 : s'il n'y a pas de redirection, alors on le note également dans le store du serveur [nuxt] : [`serverRedirection: false`]. Par ailleurs, le routage en cours va afficher la page [to] qui pour la requête suivante (client ou serveur [nuxt]) deviendra la page précédente. C'est pourquoi on écrit [`from: to`] ;
- lignes 73-76 : on sauvegarde le store dans la session [nuxt] elle-même sauvegardée dans un cookie ;
- lignes 77-80 : si [redirection] n'est pas vide, alors on demande au navigateur de se rediriger. Sinon (on ne le voit pas ici), le cycle de vie du serveur [nuxt] va se poursuivre : la page [to] va être traitée par le serveur [nuxt] et envoyée au navigateur du client [nuxt] avec le cookie de session [nuxt] ;

Le routage choisi ici pour le serveur [nuxt] est arbitraire. On aurait pu en choisir un autre ou comme il a été dit ne pas en faire du tout. Celui choisi ci-dessus a le mérite de toujours laisser l'application dans un état stable quelque soit l'URL demandée par l'utilisateur.

On peut améliorer un point lorsque la page chargée au final est la page d'origine. Il y a deux cas :

- l'utilisateur a provoqué un rechargement de la page (to===from) ;
- il y a redirections vers la page d'origine (redirection===from) ;

Dans les deux cas la page d'origine va être de nouveau exécutée avec son appel asynchrone au serveur de calcul de l'impôt. Prenons un exemple. Si une fois authentifié, l'utilisateur recharge la page (F5). Dans ce cas dans le routage ci-dessus, on a : [to]=[from]=[authentification]. Il n'y a pas redirection. La page [to=authentification] va être exécutée par le serveur [nuxt]. Si on ne fait rien, la fonction [asyncData] va s'exécuter de nouveau. C'est inutile puisque l'authentification a déjà été faite.

On peut améliorer les choses en modifiant légèrement la page [authentification] :

```
1.  // données asynchrones
2.    async asyncData(context) {
3.      // log
4.      console.log('[authentification asyncData started]')
5.      // on ne fait pas les choses deux fois si la page a déjà été demandée
6.      if (process.server && context.store.state.userAuthenticated) {
7.        console.log('[authentification asyncData canceled]')
8.        return { result: '[succès]' }
9.      }
10.     // client [nuxt]
11.     if (process.client) {
12.       // début attente
13.       context.app.$eventBus().$emit('loading', true)
14.       // pas d'erreur
15.       context.app.$eventBus().$emit('errorLoading', false)
16.     }
17.     try {
18.       // on s'authentifie auprès du serveur
19. ...
```

- lignes 6-9 : si la page est exécutée par le serveur [nuxt] et qu'on découvre dans le store que l'authentification a déjà été faite, alors on retourne directement le résultat souhaité (ligne 8) ;

On fait la même chose pour toutes les pages :

Page [index] :

```
1.  // données asynchrones
2.    async asyncData(context) {
3.      // log
4.      console.log('[index asyncData started]')
```

```
5.      // on ne fait pas les choses deux fois si la page a déjà été demandée
6.      if (process.server && context.store.state.jsonSessionStarted) {
7.        console.log('[index asyncData canceled]')
8.        return { result: '[succès]' }
9.      }
10.     try {
11. ...
```

Page [get-admindata]

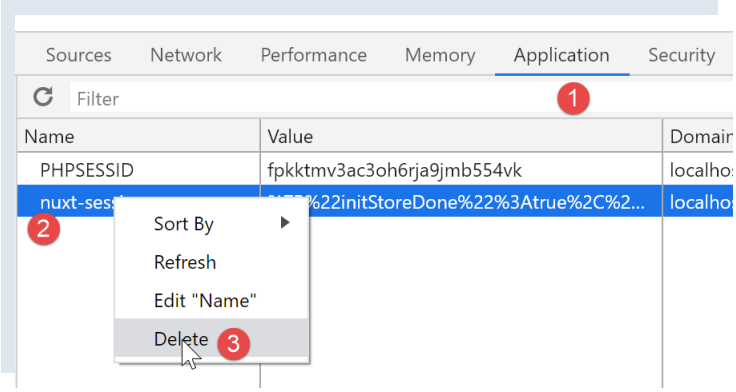
```
1.  // données asynchrones
2.    async asyncData(context) {
3.      // log
4.      console.log('[get-admindata asyncData started]')
5.      // on ne fait pas les choses deux fois si la page a déjà été demandée
6.      if (process.server && context.store.state.adminData) {
7.        console.log('[get-admindata asyncData canceled]')
8.        return { result: context.store.state.adminData }
9.      }
10.     // client
11.     if (process.client) {
12.       // début attente
13.       context.app.$eventBus().$emit('loading', true)
14.       // pas d'erreur
15.       context.app.$eventBus().$emit('errorLoading', false)
16.     }
17.     try {
18.    ...
```

Page [fin-session]

```
1.  // données asynchrones
2.    async asyncData(context) {
3.      // log
4.      console.log('[fin-session asyncData started]')
5.      // on ne fait pas les choses deux fois si la page a déjà été demandée
6.      if (process.server && context.store.state.jsonSessionStarted && !
    context.store.state.userAuthenticated) {
7.        console.log('[fin-session asyncData canceled]')
8.        return { result: "[succès]. La session jSON reste initialisée mais vous n'êtes plus
    authentifié(e)." }
9.      }
10.     // cas du client [nuxt]
11.     if (process.client) {
12.       // début attente
13.       context.app.$eventBus().$emit('loading', true)
14.       // pas d'erreur
15.       context.app.$eventBus().$emit('errorLoading', false)
16.     }
17.     try {
18.
```

16.4 Exécution

Pour exécuter cet exemple, il faut prendre soin avant l'exécution de supprimer le cookie de session [nuxt] et le cookie PHP du navigateur exécutant le client [nuxt] afin de partir d'une situation nette. Ci-dessous un exemple avec le navigateur Chrome :



16.5 Conclusion

Le routage du serveur [nuxt] est complexe car il faut prévoir toutes les URL que peut taper à la main l'utilisateur. C'est un cas d'école. Une application [nuxt] n'est pas destinée à être utilisée de cette façon. Une fois la page [index] servie par le routeur du serveur [nuxt], on pourrait rediriger les appels suivants faits au serveur vers une page d'erreur.

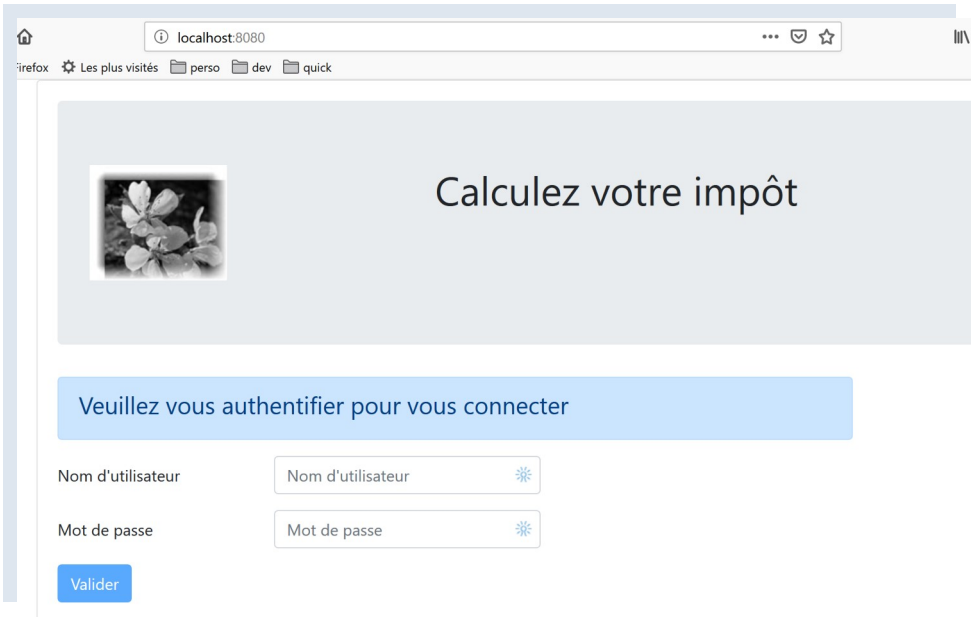
Dans le cas précis de notre exemple [nuxt-13], le routage du serveur [nuxt] était inutile. Celui fait par défaut (absence de routage en fait) dans l'exemple [nuxt-12] convenait très bien.

17 Exemple [nuxt-20] : portage de l'exemple [vuejs-22]

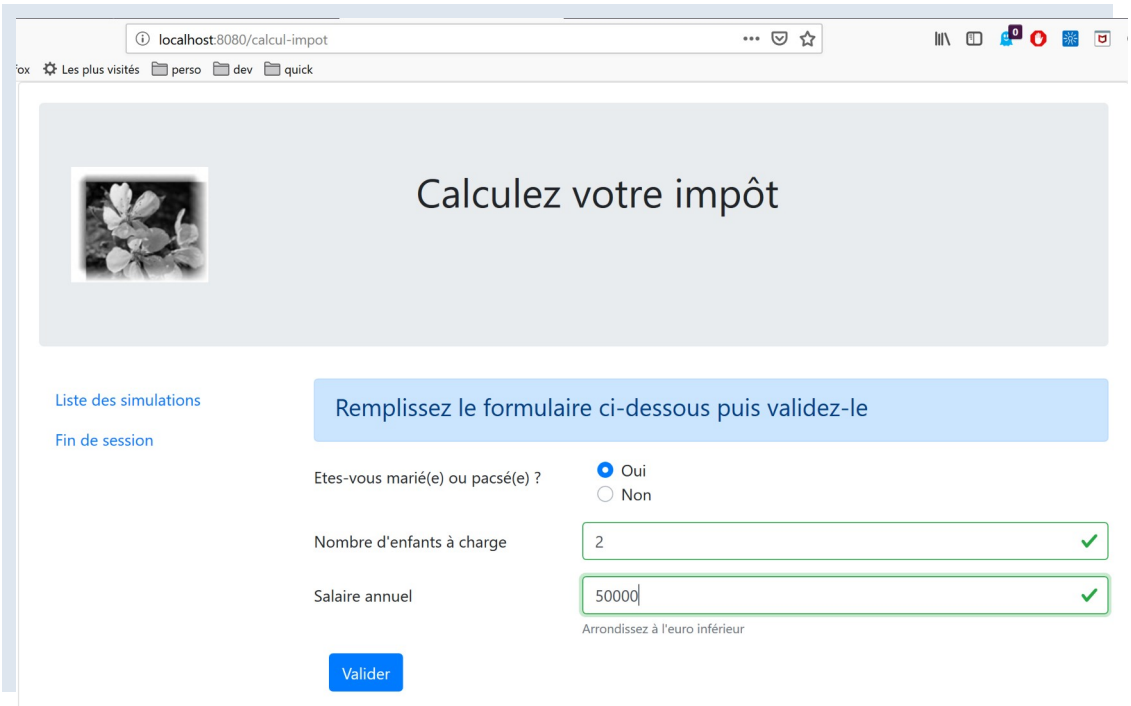
17.1 Présentation

Nous nous proposons ici de porter l'exemple [vuejs-22] qui était une application [vue.js] de type SPA, dans un contexte [nuxt] SSR. [vuejs-22] était une application cliente du serveur de calcul de l'impôt qui présentait les vues suivantes :

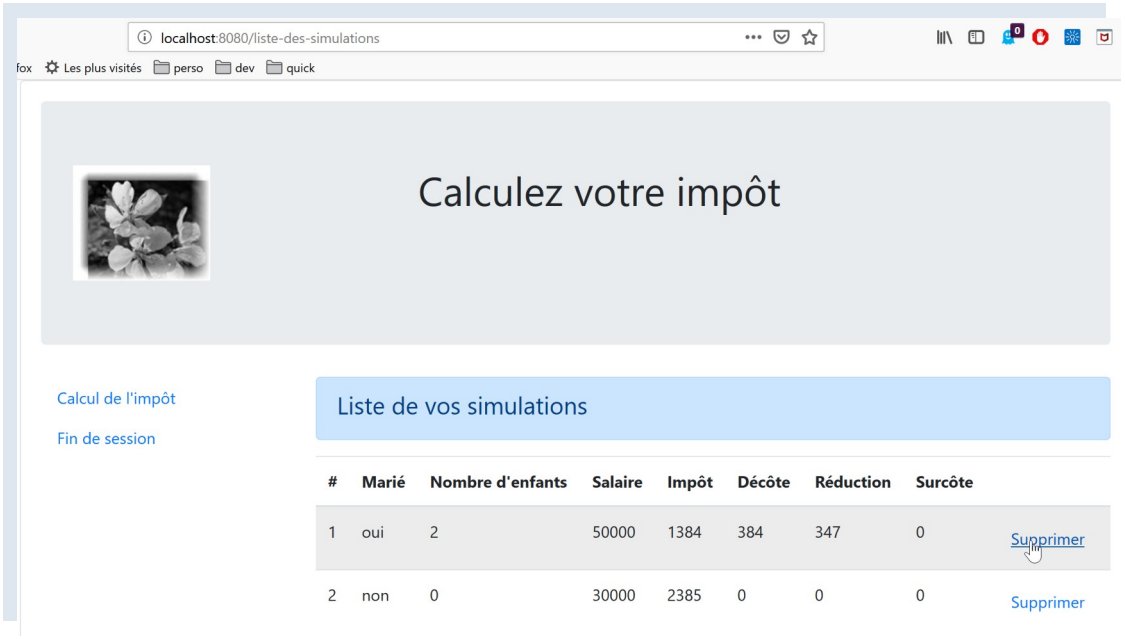
La 1ère vue est la vue d'authentification :



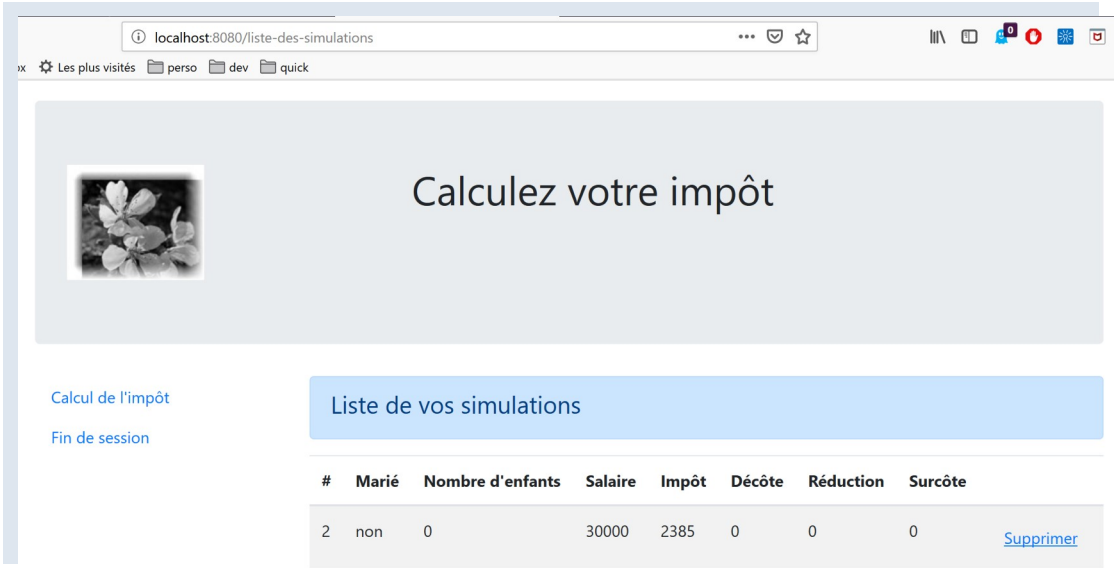
La seconde vue est celle du calcul de l'impôt :



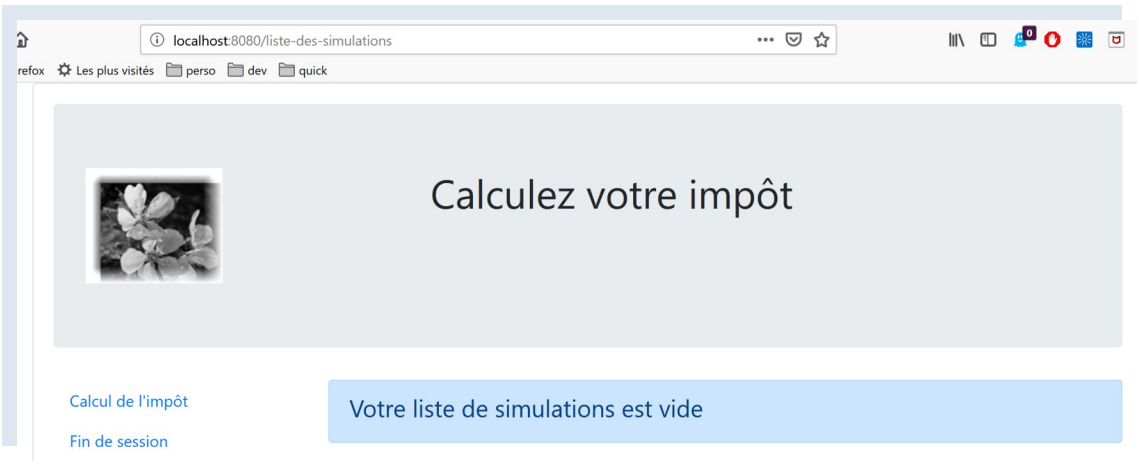
La 3ième vue est celle qui affiche la liste des simulations faites par l'utilisateur :



L'écran ci-dessus montre qu'on peut supprimer la simulation n° 1. On obtient alors la vue suivante :



Si on supprime maintenant la dernière simulation, on obtient la nouvelle vue suivante :



Nous allons porter l'application [vuejs-22] vers l'application [nuxt-20] de façon progressive. Nous n'expliquerons pas de nouveau les codes de [vuejs-22]. Le lecteur est invité à relire le document |Introduction au framework **VUE.JS** par l'exemple|. Les différentes étapes devraient montrer les différences entre une application [vuejs] et une application [nuxt].

17.2 étape 1

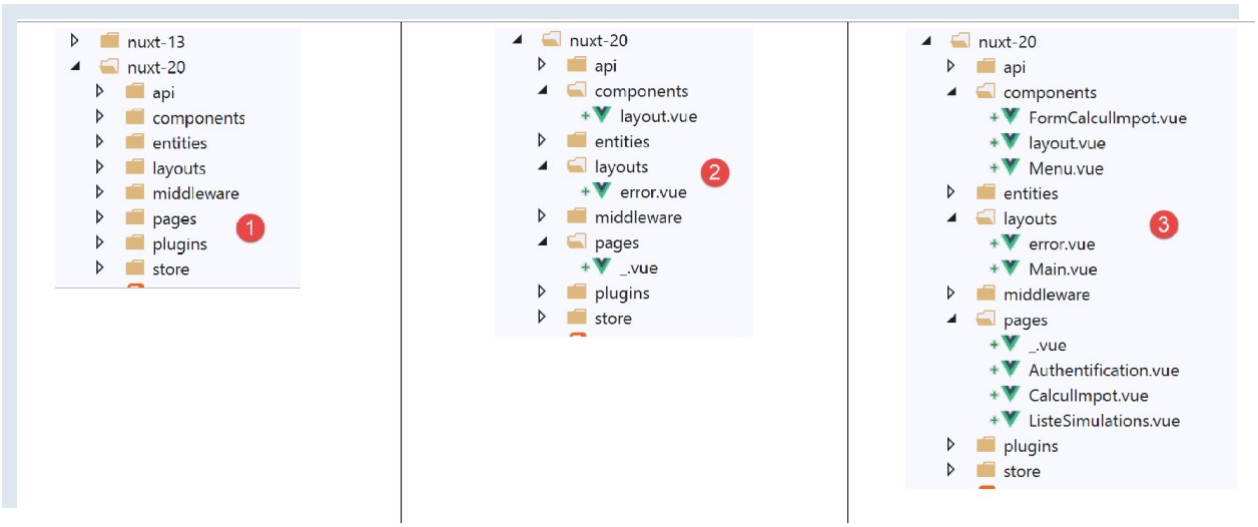
Le projet [nuxt-20] est initialement obtenu par recopie du projet [nuxt-12]. Celui-ci est en effet un bon point de départ :

- il sait dialoguer avec le serveur de calcul de l'impôt ;
- il gère correctement les erreurs que celui-ci envoie ;
- les client et serveur [nuxt] savent communiquer via une session [nuxt] ;

On a donc une bonne infrastructure de départ. Notre principal travail devrait être de modifier :

- les pages. On prendra celles du projet [vuejs-22] qu'il faudra adapter au nouvel environnement ;
- la gestion du store. Des informations supplémentaires devraient apparaître (liste des simulations) et d'autres pourraient devenir inutiles ;
- la gestion du routage du client et du serveur [nuxt] ;

Donc tout d'abord, on crée le projet [nuxt-20] en recopiant le projet [nuxt-12] :



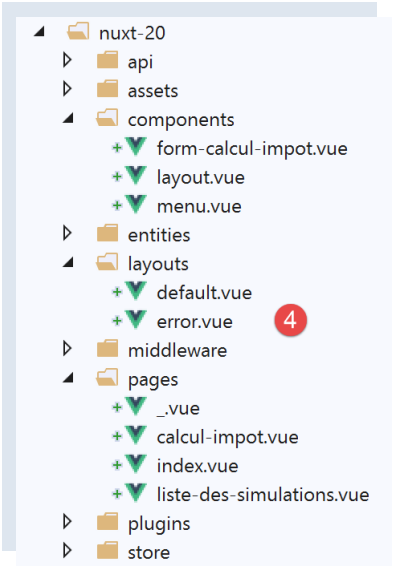
Puis on supprime les pages et composants devenus inutiles [2] :

- le composant [components/navigation] disparaît ;
- le layout [layout/default] disparaît ;
- les pages [index, authentification, get-admindata, fin-session] disparaissent ;

Puis on intègre dans [nuxt-20] des éléments de [vuejs-22] [3] :

- les trois pages [Authentification, CalculImpot, ListeSimulations] de l'application [vuejs-22] vont dans le dossier [pages] ;
- les composants [FormCalculImpot, Menu, Layout] de l'application [vuejs-22] vont dans le dossier [components] ;
- la page [Main] de [vuejs-22] qui servait de [layout] à l'application [vuejs-22] va dans le dossier [layouts] ;

On renomme les éléments intégrés [4] :



- dans [layouts], [Main] est devenue [default] puisque c'est le nom par défaut du layout d'une application [nuxt] ;
- dans [pages], la page [Authentification] est devenue [index], car [Authentification] jouait ce rôle dans l'application [vuejs-22] ;

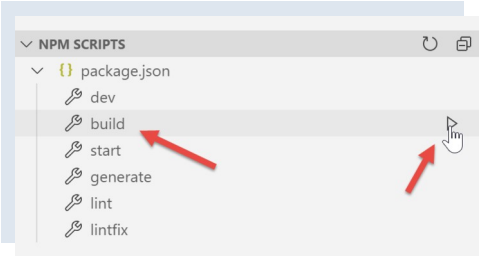
A ce point là, on peut faire une compilation du projet pour voir les premières erreurs. On modifie le fichier [nuxt.config] de l'exemple [nuxt-12] afin d'exécuter désormais [nuxt-20] :

```
1.  export default {
2.    mode: 'universal',
3.    /*
4.     ** Headers of the page
5.     */
6.    head: {
7.      title: 'Introduction à [nuxt.js]',
8.      meta: [
9.        { charset: 'utf-8' },
10.       { name: 'viewport', content: 'width=device-width, initial-scale=1' },
11.       {
12.         hid: 'description',
13.         name: 'description',
14.         content: 'ssr routing loading asyncdata middleware plugins store'
15.       }
16.     ],
17.     link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
18.   },
19.   /*
```

```
    ** Customize the progress-bar color
    */
  loading: false,

  /*
   ** Global CSS
   */
  css: [],
  /*
   ** Plugins to load before mounting the App
   */
  plugins: [
    { src: '@/plugins/client/plgSession', mode: 'client' },
    { src: '@/plugins/server/plgSession', mode: 'server' },
    { src: '@/plugins/client/plgDao', mode: 'client' },
    { src: '@/plugins/server/plgDao', mode: 'server' },
    { src: '@/plugins/client/plgEventBus', mode: 'client' }
  ],
  /*
   ** Nuxt.js dev-modules
   */
  buildModules: [
    // Doc: https://github.com/nuxt-community/eslint-module
    '@nuxtjs/eslint-module'
  ],
  /*
   ** Nuxt.js modules
   */
  modules: [
    // Doc: https://bootstrap-vue.js.org
    'bootstrap-vue/nuxt',
    // Doc: https://axios.nuxtjs.org/usage
    '@nuxtjs/axios',
    // https://www.npmjs.com/package/cookie-universal-nuxt
    'cookie-universal-nuxt'
  ],
  /*
   ** Axios module configuration
   ** See https://axios.nuxtjs.org/options
   */
  axios: {},
  /*
   ** Build configuration
   */
  build: {
    /*
     ** You can extend webpack config here
     */
    extend(config, ctx) {}
  },
  // répertoire du code source
  srcDir: 'nuxt-20',
  // routeur
  router: {
    // racine des URL de l'application
    base: '/nuxt-20/',
    // middleware de routage
    middleware: ['routing']
  },
  // serveur
  server: {
    // port de service, 3000 par défaut
    port: 81,
    // adresses réseau écoutées, par défaut localhost : 127.0.0.1
    // 0.0.0.0 = toutes les adresses réseau de la machine
    host: 'localhost'
  },
  // environnement
  env: {
    // configuration axios
    timeout: 2000,
    withCredentials: true,
    baseURL: 'http://localhost/php7/scripts-web/impots/version-14',
    // configuration du cookie de session [nuxt]
```

```
94.     maxAge: 60 * 5
95.   }
96. }
```

On fait ensuite un [build] du projet :

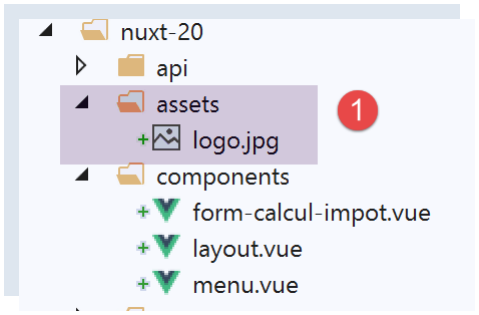


Les erreurs signalées sont les suivantes :

```
1.  Module not found: Error: Can't resolve '../assets/logo.jpg' @
    ./nuxt-20/layouts/default.vue?...
2.  Module not found: Error: Can't resolve './FormCalculImpot' @ ./nuxt-20/pages/calcul-
    impot.vue?...
3.  Module not found: Error: Can't resolve './Layout' @ ./nuxt-20/pages/_.vue?...
4.  Module not found: Error: Can't resolve './Layout' @ ./nuxt-20/pages/liste-des-simulations...
5.  Module not found: Error: Can't resolve './Layout' @ ./nuxt-20/pages/calcul-impot.vue?...
6.  Module not found: Error: Can't resolve './Menu' @ ./nuxt-20/pages/_.vue?...
7.  Module not found: Error: Can't resolve './Menu' @ ./nuxt-20/pages/calcul-impot.vue
```

- l'erreur de la ligne 1 indique qu'on référence une image inexistante. On la récupèrera dans [vuejs-22] ;
- l'erreur de la ligne 2 montre que le composant [./FormCalculImpot] n'existe pas. Effectivement, ce composant est désormais dans [@/components/form-calcul-impot] ;
- les erreurs des lignes [3-5] montrent que le composant [./Layout] n'existe pas. Effectivement, ce composant est désormais dans [@/components/layout] ;
- les erreurs des lignes [6-7] montrent que le composant [./Menu] n'existe pas. Effectivement, il s'appelle maintenant [@/components/menu] ;

On ajoute l'image [`assets/logo.jpg`] au projet [nuxt-20] :



Par ailleurs, dans toutes les pages on va corriger le chemin des composants. Prenons l'exemple de la page [calcul-impot] :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <div>
4.      <Layout :left="true" :right="true">
5.        <!-- formulaire de calcul de l'impôt à droite -->
6.        <FormCalculImpot slot="right" @resultatObtenu="handleResultatObtenu" />
7.        <!-- menu de navigation à gauche -->
8.        <Menu slot="left" :options="options" />
9.      </Layout>
10.     <!-- zone d'affichage des résultat du calcul de l'impôt sous le formulaire -->
```

```
11.      <b-row v-if="résultatObtenu" class="mt-3">
12.        <!-- zone de trois colonnes vide -->
13.        <b-col sm="3" />
14.        <!-- zone de neuf colonnes -->
15.        <b-col sm="9">
16.          <b-alert show variant="success">
17.            <span v-html="résultat"></span>
18.          </b-alert>
19.        </b-col>
20.      </b-row>
21.    </div>
22. </template>
23.
24. <script>
25. // imports
26. import FormCalculImpot from './FormCalculImpot'
27. import Menu from './Menu'
28. import Layout from './Layout'
29.
30. export default {
31.   // composants utilisés
32.   components: {
33.     Layout,
34.     FormCalculImpot,
35.     Menu
36.   },
37.
```

Les trois [import] des lignes 26-28 deviennent :

```
1.  // imports
2.  import FormCalculImpot from '@/components/form-calcul-impot'
3.  import Menu from '@/components/menu'
4.  import Layout from '@/components/layout'
```

On vérifie et éventuellement corrige ainsi les [import] de tous les composants, layouts et pages. Une fois ces corrections faites, on peut tenter un nouveau [build]. Normalement il n'y a plus d'erreurs.

On peut alors tenter une exécution :

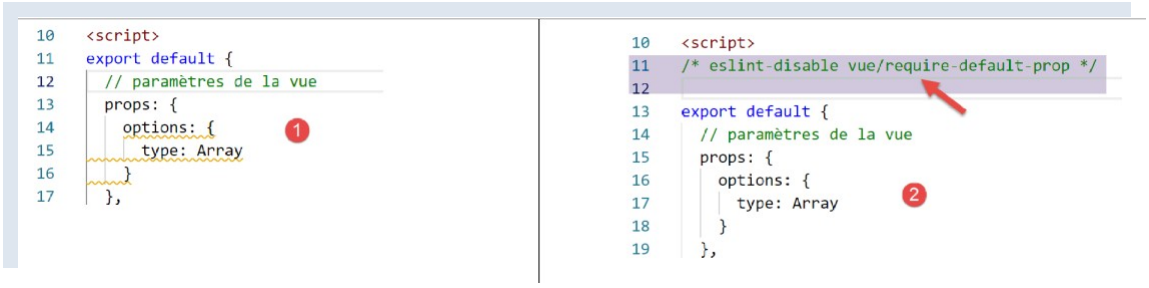
Des erreurs apparaissent :

```
1.  Module Error (from ./node_modules/eslint-loader/dist/cjs.js):
2.
3.  c:\Data\st-2019\dev\nuxtjs\dvp\nuxt-20\pages\index.vue
4.     92:29  error  Expected '!==' and instead saw '!='  eqeqeq
5.    129:27  error  Expected '!==' and instead saw '!='  eqeqeq
6.  150:28  error  Expected '!==' and instead saw '!='  eqeqeq
```

On voit que la commande [dev] alliée au module [eslint] est plus stricte, au niveau syntaxique, que la commande [build]. Ici, elle réclame que l'opérateur de comparaison [!=] soit écrit [!==] qui est un opérateur plus strict (il vérifie également le type des opérandes). Ces erreurs se produisent dans la page [index.vue].

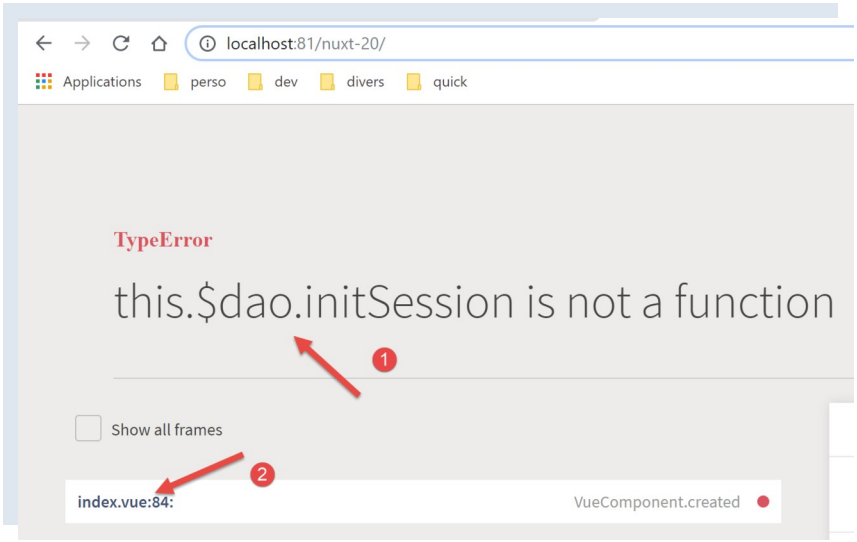
On corrige les erreurs ci-dessus et on relance l'exécution du projet. On a alors un warning du module [eslint] :

```
1.  c:\Data\st-2019\dev\nuxtjs\dvp\nuxt-20\components\menu.vue
2.  14:5  warning  Prop 'options' requires default value to be set  vue/require-default-prop
```



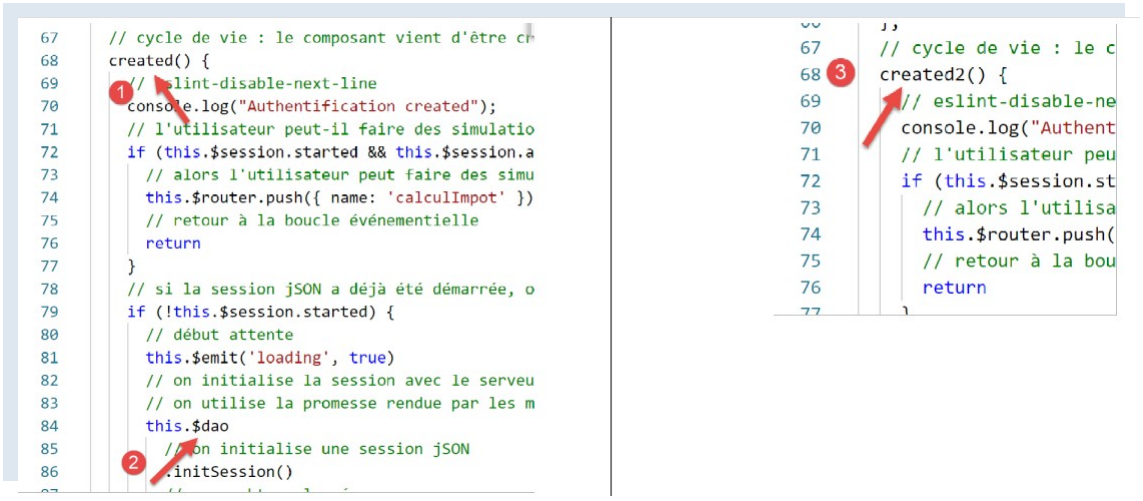
On corrige cette erreur avec le [Quick fix] du module [eslint] [2].

On relance l'exécution du projet. On n'a plus d'erreur de compilation. On demande alors l'URL [http://localhost:81/nuxt-20/] avec un navigateur. On obtient une erreur d'exécution :



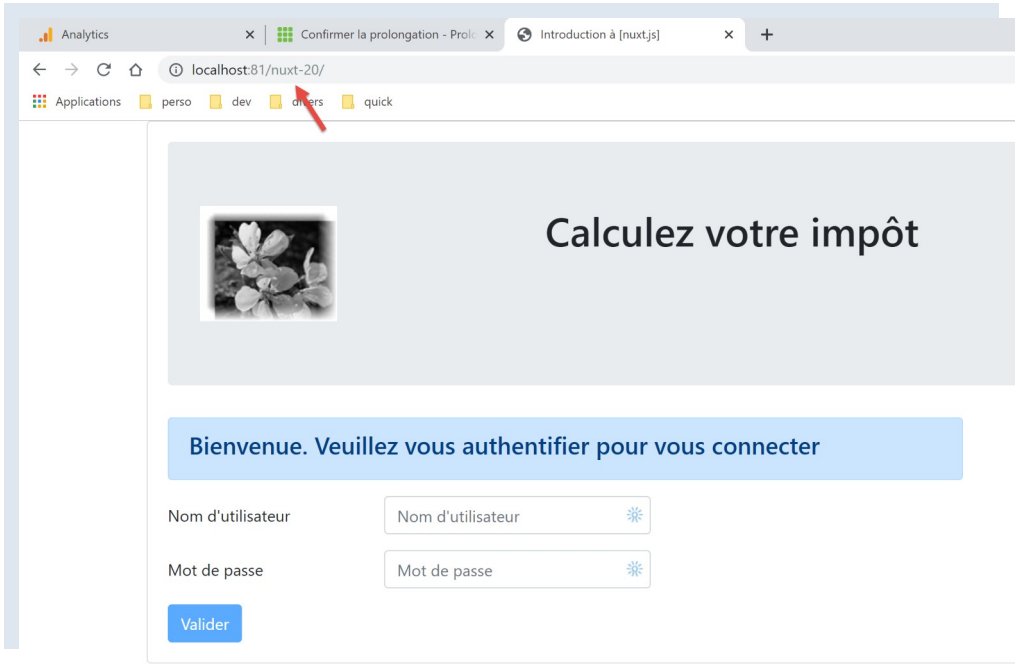
L'erreur se trouve dans [index.vue] [2]. L'erreur [1] vient du fait que dans [vuejs-22], la couche [dao] était disponible dans [this.$dao] alors que dans [nuxt-12] dont nous avons adopté l'infrastructure, elle est disponible dans la fonction [this.$dao()].

L'erreur est dans la fonction [created] du cycle de vie de la page [index] :



Pour l'instant, on se contente de renommer [created] en [created2] pour que la fonction du cycle de vie [created] ne soit pas exécutée [3].

On sauve la modification et on recharge la page [index] dans le navigateur. Cette fois-ci c'est bon :



17.3 étape 2

Les pages du projet [vuejs-22] utilisaient les éléments injectés suivants :

- $dao : pour la couche [dao] du client [vue.js] ;
- $session : pour une session stockée dans le [localStorage] du navigateur ;

Ces éléments n'existent plus dans l'infrastructure du projet [nuxt-12] que nous avons recopiée :

- il y a désormais deux couches [dao], l'une pour le client [nuxt], l'autre pour le serveur [nuxt]. Toutes deux sont disponibles via une **fonction** injectée appelée [$dao]. Cela signifie que dans les pages de l'application [this.$dao] doit être remplacé par [**this.$dao()**] ;
- la session [nuxt] gérée par l'application [nuxt-20] n'a plus rien à voir avec l'objet [$session] de l'application [vuejs-22] où il n'y avait pas de notion de cookie de session. Néanmoins, elles ont une fonctionnalité analogue : stocker des informations persistantes au fil des actions de l'utilisateur. La session [nuxt] stocke les informations dans le store plutôt que directement dans la session. Dans les pages de l'application [this.$session] doit être remplacé par [**this.$store**] lorsqu'il s'agit de mémoriser des informations dans la session et par [**this.$session()**] lorsqu'il s'agit de manipuler la session elle-même ;
- pour connaître l'état d'une propriété P du store, il faudra écrire [**this.$store.state.P**] ;
- pour changer la propriété P du store, il faudra écrire [**this.$store.commit('replace', {P:value}**] ;

Nous faisons ces modifications dans la page [index] :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <Layout :left="false" :right="true">
4.      <template slot="right">
5.        <!-- formulaire HTML - on poste ses valeurs avec l'action [authentifier-utilisateur] -->
6.        <b-form @submit.prevent="login">
7.          <!-- titre -->
```

```
8.          <b-alert show variant="primary">
9.            <h4>Bienvenue. Veuillez vous authentifier pour vous connecter</h4>
10.         </b-alert>
11.         <!-- 1ère ligne -->
12.         <b-form-group label="Nom d'utilisateur" label-for="user" label-cols="3">
13.           <!-- zone de saisie user -->
14.           <b-col cols="6">
15.             <b-form-input id="user" v-model="user" type="text" placeholder="Nom d'utilisateur"
    />
16.           </b-col>
17.         </b-form-group>
18.         <!-- 2ième ligne -->
19.         <b-form-group label="Mot de passe" label-for="password" label-cols="3">
20.           <!-- zone de saisie password -->
21.           <b-col cols="6">
22.             <b-input id="password" v-model="password" type="password" placeholder="Mot de
    passe" />
23.           </b-col>
24.         </b-form-group>
25.         <!-- 3ième ligne -->
26.         <b-alert v-if="showError" show variant="danger" class="mt-3">L'erreur suivante s'est
    produite : {{ message }}</b-alert>
27.         <!-- bouton de type [submit] sur une 3ième ligne -->
28.         <b-row>
29.           <b-col cols="2">
30.             <b-button :disabled="!valid" variant="primary" type="submit">Valider</b-button>
31.           </b-col>
32.         </b-row>
33.       </b-form>
34.     </template>
35.   </Layout>
36. </template>
37.
38. <!-- dynamique de la vue -->
39. <script>
40. /* eslint-disable no-console */
41. import Layout from '@/components/layout'
42. export default {
43.   // composants utilisés
44.   components: {
45.     Layout
46.   },
47.   // état du composant
48.   data() {
49.     return {
50.       // utilisateur
51.       user: '',
52.       // son mot de passe
53.       password: '',
54.       // contrôle l'affichage d'un msg d'erreur
55.       showError: false,
56.       // le message d'erreur
57.       message: ''
58.     }
59.   },
60.
61.   // propriétés calculées
62.   computed: {
63.     // saisies valides
64.     valid() {
65.       return this.user && this.password && this.$store.state.started
66.     }
67.   },
68.   // cycle de vie : le composant vient d'être créé
69.   mounted() {
70.     // eslint-disable-next-line
71.     console.log("Authentification mounted");
72.     // l'utilisateur peut-il faire des simulations ?
73.     if (this.$store.state.started && this.$store.state.authenticated && this.
    $métier.taxAdminData) {
74.       // alors l'utilisateur peut faire des simulations
75.       this.$router.push({ name: 'calculImpot' })
76.       // retour à la boucle événementielle
77.       return
```

```
78.     }
79.     // si la session jSON a déjà été démarrée, on ne la redémarre pas de nouveau
80.     if (!this.$store.state.started) {
81.       // début attente
82.       this.$emit('loading', true)
83.       // on initialise la session avec le serveur - requête asynchrone
84.       // on utilise la promesse rendue par les méthodes de la couche [dao]
85.       this.$dao()
86.         // on initialise une session jSON
87.         .initSession()
88.         // on a obtenu la réponse
89.         .then((response) => {
90.           // fin attente
91.           this.$emit('loading', false)
92.           // analyse de la réponse
93.           if (response.état !== 700) {
94.             // on affiche l'erreur
95.             this.message = response.réponse
96.             this.showError = true
97.             // retour à la boucle événementielle
98.             return
99.           }
100.          // la session a démarré
101.          this.$store.commit('replace', { started: true })
102.          console.log('[authentification], session=', this.$session())
103.        })
104.        // en cas d'erreur
105.        .catch((error) => {
106.          // on remonte l'erreur à la vue [Main]
107.          this.$emit('error', error)
108.        })
109.        // dans tous les cas
110.        .finally(() => {
111.          // on sauvegarde la session
112.          this.$session().save()
113.        })
114.    }
115.  },
116.
117.  // gestionnaires d'évts
118.  methods: {
119.    // ----------- authentification
120.    async login() {
121.      try {
122.        // début attente
123.        this.$emit('loading', true)
124.        // on n'est pas encore authentifié
125.        this.$store.commit('replace', { authenticated: false })
126.        // authentification bloquante auprès du serveur
127.        const response = await this.$dao().authentifierUtilisateur(this.user, this.password)
128.        // fin du chargement
129.        this.$emit('loading', false)
130.        // analyse de la réponse du serveur
131.        if (response.état !== 200) {
132.          // on affiche l'erreur
133.          this.message = response.réponse
134.          this.showError = true
135.          // retour à la boucle événementielle
136.          return
137.        }
138.        // pas d'erreur
139.        this.showError = false
140.        // on est authentifié
141.        this.$store.commit('replace', { authenticated: true })
142.        // --------- on demande maintenant les données de l'administration fiscale
143.        // au départ, pas de donnée
144.        this.$métier.setTaxAdminData(null)
145.        // début attente
146.        this.$emit('loading', true)
147.        // demande bloquante auprès du serveur
148.        const response2 = await this.$dao().getAdminData()
149.        // fin du chargement
150.        this.$emit('loading', false)
151.        // analyse de la réponse
```

```
152.         if (response2.état !== 1000) {
153.           // on affiche l'erreur
154.           this.message = response2.réponse
155.           this.showError = true
156.           // retour à la boucle événementielle
157.           return
158.         }
159.         // pas d'erreur
160.         this.showError = false
161.         // on mémorise dans la couche [métier] la donnée reçue
162.         this.$métier.setTaxAdminData(response2.réponse)
163.         // on peut passer au calcul de l'impôt
164.         this.$router.push({ name: 'calculImpot' })
165.       } catch (error) {
166.         // on remonte l'erreur au composant principal
167.         this.$emit('error', error)
168.       } finally {
169.         // maj store
170.         this.$store.commit('replace', { métier: this.$métier })
171.         // on sauvegarde la session
172.         this.$session().save()
173.       }
174.     }
175.   }
176.}
177.</script>
```

Notons les points suivants :

- ligne 69 : la fonction [created2] a été renommée [mounted], ceci pour que le serveur [nuxt] ne l'exécute pas (il n'exécute ni [beforeMount] ni [mounted]). Seul le client [nuxt] l'exécutera comme c'était le cas avec l'exemple [vuejs-22] ;
- ligne 73 : on référence [this.$métier] qui pour l'instant n'existe pas ;
- ligne 75 : nous n'avons jamais utilisé cette méthode dans une application [nuxt]. Il faudra voir si elle fonctionne dans un contexte [nuxt] ;
- ligne 112, 172 : dans [vuejs-22], la session du projet était sauvegardée de cette façon. Avec le projet [nuxt-20], la méthode [save] doit recevoir le contexte courant. On sait que dans une page [nuxt], l'objet [context] est disponible dans [**this.$nuxt.context**] ;

Les lignes 112 et 172 sont donc réécrites de la façon suivante :

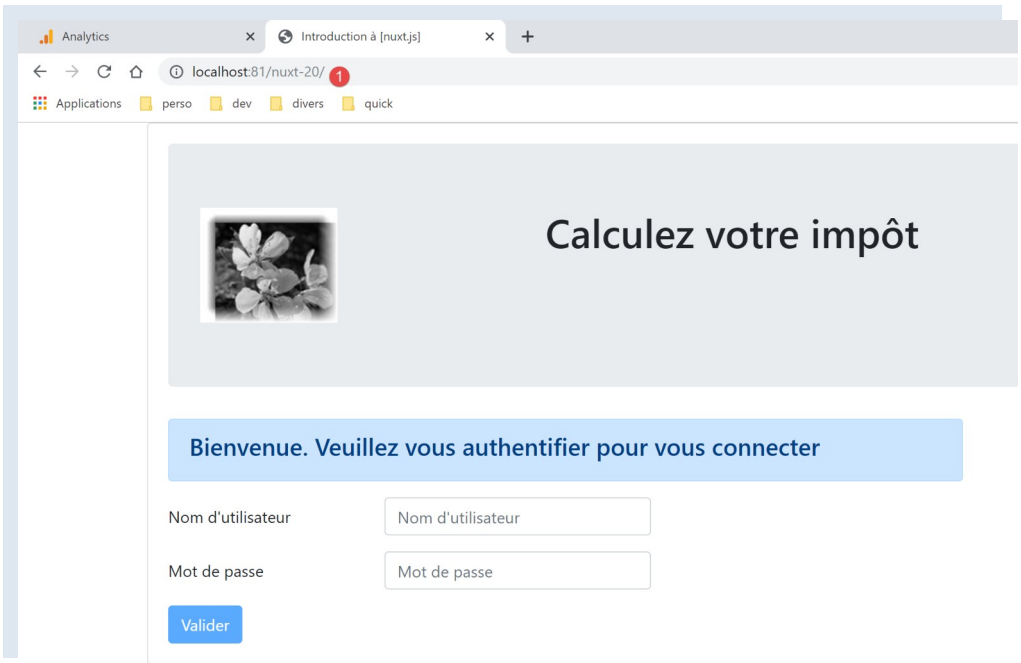
```
1.  this.$session().save(this.$nuxt.context)
```

On notera que ce code n'est pas optimisé. Plutôt que d'utiliser plusieurs fois la fonction [`this.$session()`], il serait préférable d'écrire :

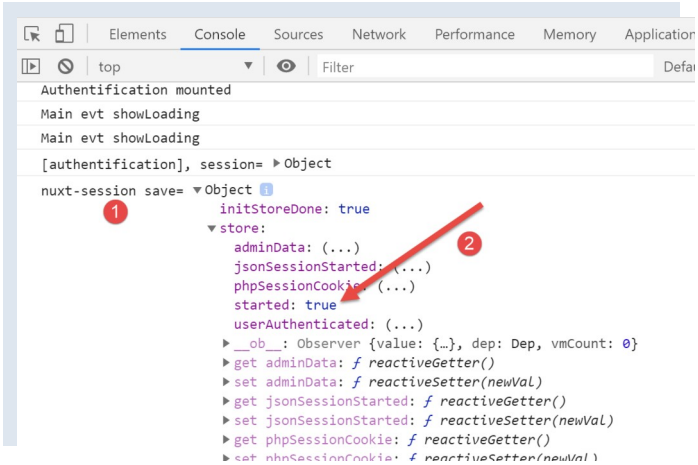
```
1.  const session=this.$session()
```

puis utiliser ensuite la variable [session]. On peut tenir le même raisonnement pour la fonction [**this.$dao()**].

Ces corrections faites, nous pouvons recharger l'URL [http://localhost:81/nuxt-20/] avec un navigateur. Nous obtenons toujours la même page que précédemment :

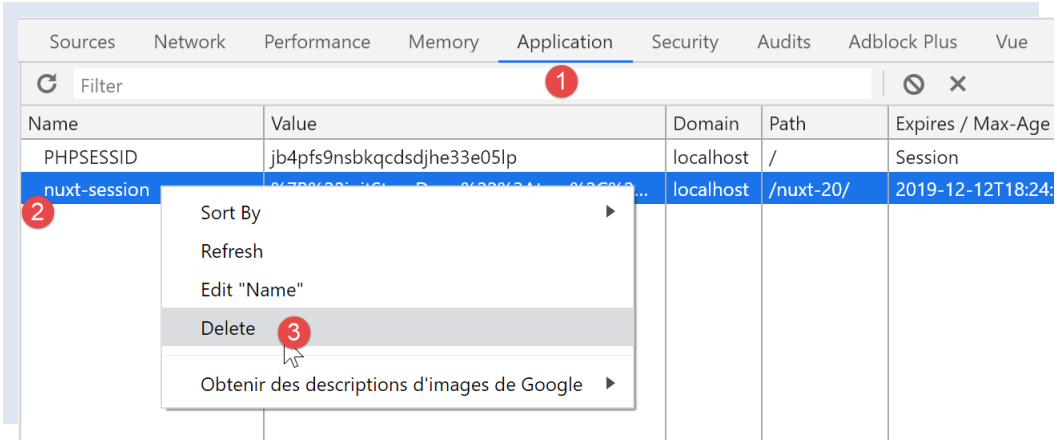


Regardons les logs du navigateur :

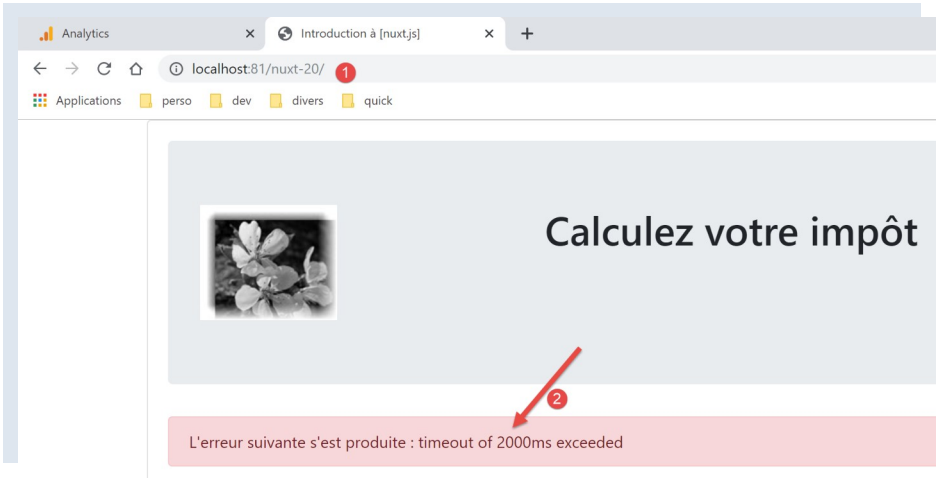


Le log [1] est le dernier log fait par le client [nuxt]. En [2], on voit que la propriété [started] est à [vrai], ce qui veut dire que la fonction [mounted] a réussi à démarrer une session jSON avec le serveur de calcul de l'impôt. On voit également que le store a des propriétés qu'il faudra soit abandonner soit renommer. Rappelons que nous utilisons le store de l'exemple [nuxt-12].

Maintenant redemandons l'URL [http://localhost:81/nuxt-20/] alors que le serveur de calcul de l'impôt n'est pas lancé. On prend soin tout d'abord de supprimer le cookie de la session [nuxt] :



La copie d'écran ci-dessus est une copie d'écran Chrome. Ceci fait, l'URL [http://localhost:81/nuxt-20/] donne le résultat suivant :



L'erreur était correctement gérée par le projet [vuejs-22]. Elle reste correctement gérée par le projet [nuxt-20].

17.4 étape 3

Maintenant que nous avons la page d'authentification, il faut regarder le code exécuté lorsque l'utilisateur clique sur le bouton [Valider] :

```
1.  // gestionnaires d'évts
2.    methods: {
3.      // ---------- authentification
4.      async login() {
5.        try {
6.          // début attente
7.          this.$emit('loading', true)
8.          // on n'est pas encore authentifié
9.          this.$store.commit('replace', { authenticated: false })
10.         // authentification bloquante auprès du serveur
11.         const response = await this.$dao().authentifierUtilisateur(this.user, this.password)
12.         // fin du chargement
13.         this.$emit('loading', false)
14.         // analyse de la réponse du serveur
15.         if (response.état !== 200) {
16.           // on affiche l'erreur
17.           this.message = response.réponse
18.           this.showError = true
```

```
19.           // retour à la boucle événementielle
20.           return
21.         }
22.         // pas d'erreur
23.         this.showError = false
24.         // on est authentifié
25.         this.$store.commit('replace', { authenticated: true })
26.         // --------- on demande maintenant les données de l'administration fiscale
27.         // au départ, pas de donnée
28.         this.$métier.setTaxAdminData(null)
29.         // début attente
30.         this.$emit('loading', true)
31.         // demande bloquante auprès du serveur
32.         const response2 = await this.$dao().getAdminData()
33.         // fin du chargement
34.         this.$emit('loading', false)
35.         // analyse de la réponse
36.         if (response2.état !== 1000) {
37.           // on affiche l'erreur
38.           this.message = response2.réponse
39.           this.showError = true
40.           // retour à la boucle événementielle
41.           return
42.         }
43.         // pas d'erreur
44.         this.showError = false
45.         // on mémorise dans la couche [métier] la donnée reçue
46.         this.$métier.setTaxAdminData(response2.réponse)
47.         // on peut passer au calcul de l'impôt
48.         this.$router.push({ name: 'calculImpot' })
49.       } catch (error) {
50.         // on remonte l'erreur au composant principal
51.         this.$emit('error', error)
52.       } finally {
53.         // maj store
54.         this.$store.commit('replace', { métier: this.$métier })
55.         // on sauvegarde la session
56.         this.$session().save(this.$nuxt.context)
57.       }
58.     }
59.   }
```

Le principal problème ici semble être l'absence de la donnée [this.$métier]. Pour y remédier nous allons :

- inclure la classe [Métier] de l'exemple [vuejs-22]. Nous la mettrons dans le dossier [api] ;
- injecter une fonction [$métier] dans le contexte du client [nuxt] qui donnera accès à cette classe ;

Tout d'abord la copie de la classe [Métier] dans le dossier [api] :

Une fois la classe [Métier] présente dans le projet, on crée un nouveau plugin pour le client [nuxt]. Ce plugin appelé [pluginMétier] va injecter une fonction [$métier] qui donnera accès à la classe [Métier] :

```
1.  /* eslint-disable no-console */
2.  // on crée un point d'accès à la couche [métier]
3.  import Métier from '@/api/client/Métier'
4.  export default (context, inject) => {
5.    // instanciation de la couche [métier]
6.    const métier = new Métier()
7.    // injection d'une fonction [$métier] dans le contexte
8.    inject('métier', () => métier)
9.    // log
```

```
10.   console.log('[fonction client $métier créée]')
11. }
```

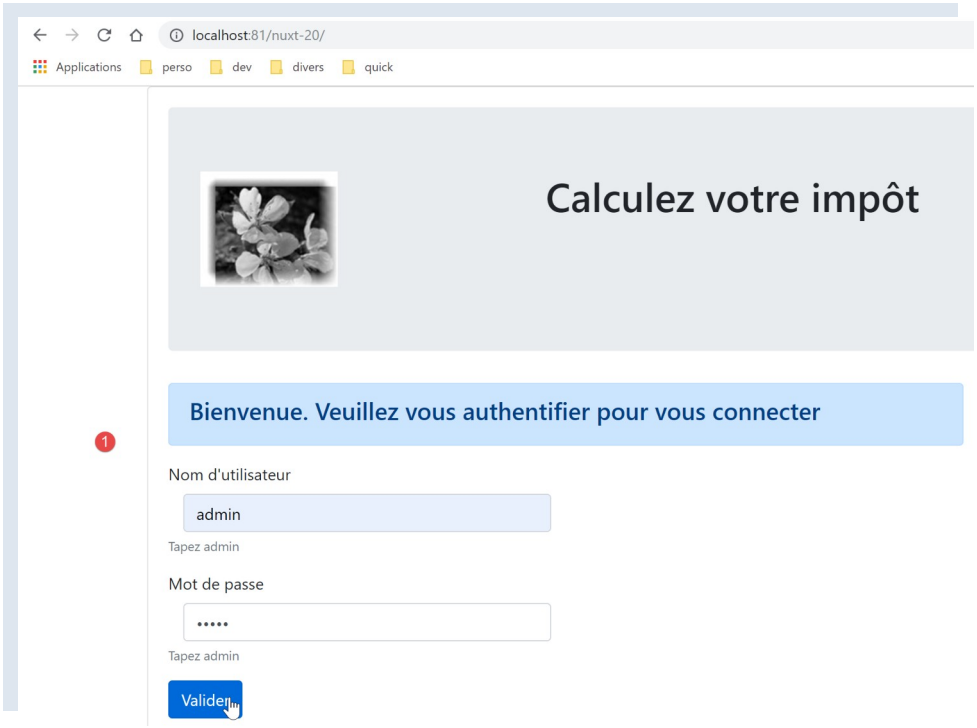
Ceci fait, nous pouvons corriger la page [index] :

```
1.  // cycle de vie : le composant vient d'être créé
2.    mounted() {
3.      // eslint-disable-next-line
4.      console.log("Authentification mounted");
5.      // l'utilisateur peut-il faire des simulations ?
6.      if (this.$store.state.started && this.$store.state.authenticated && this.
    $métier().taxAdminData) {
7.        // alors l'utilisateur peut faire des simulations
8.        this.$router.push({ name: 'calcul-impot' })
9.        // retour à la boucle événementielle
10.       return
11.     }
12.     // si la session jSON a déjà été démarrée, on ne la redémarre pas de nouveau
13.     ...
14.   },
15.
16.   // gestionnaires d'évts
17.   methods: {
18.     // ----------- authentification
19.     async login() {
20.       try {
21.         // début attente
22.         this.$emit('loading', true)
23.         // on n'est pas encore authentifié
24.         this.$store.commit('replace', { authenticated: false })
25.         // authentification bloquante auprès du serveur
26.         const response = await this.$dao().authentifierUtilisateur(this.user, this.password)
27.         // fin du chargement
28.         this.$emit('loading', false)
29.         // analyse de la réponse du serveur
30.         if (response.état !== 200) {
31.           // on affiche l'erreur
32.           this.message = response.réponse
33.           this.showError = true
34.           // retour à la boucle événementielle
35.           return
36.         }
37.         // pas d'erreur
38.         this.showError = false
39.         // on est authentifié
40.         this.$store.commit('replace', { authenticated: true })
41.         // --------- on demande maintenant les données de l'administration fiscale
42.         // au départ, pas de donnée
43.         this.$métier().setTaxAdminData(null)
44.         // début attente
45.         this.$emit('loading', true)
46.         // demande bloquante auprès du serveur
47.         const response2 = await this.$dao().getAdminData()
48.         // fin du chargement
49.         this.$emit('loading', false)
50.         // analyse de la réponse
51.         if (response2.état !== 1000) {
52.           // on affiche l'erreur
53.           this.message = response2.réponse
54.           this.showError = true
55.           // retour à la boucle événementielle
56.           return
57.         }
58.         // pas d'erreur
59.         this.showError = false
60.         // on mémorise dans la couche [métier] la donnée reçue
61.         this.$métier().setTaxAdminData(response2.réponse)
62.         // on peut passer au calcul de l'impôt
63.         this.$router.push({ name: 'calcul-impot' })
64.       } catch (error) {
65.         // on remonte l'erreur au composant principal
66.         this.$emit('error', error)
67.       } finally {
68.         // maj store
```

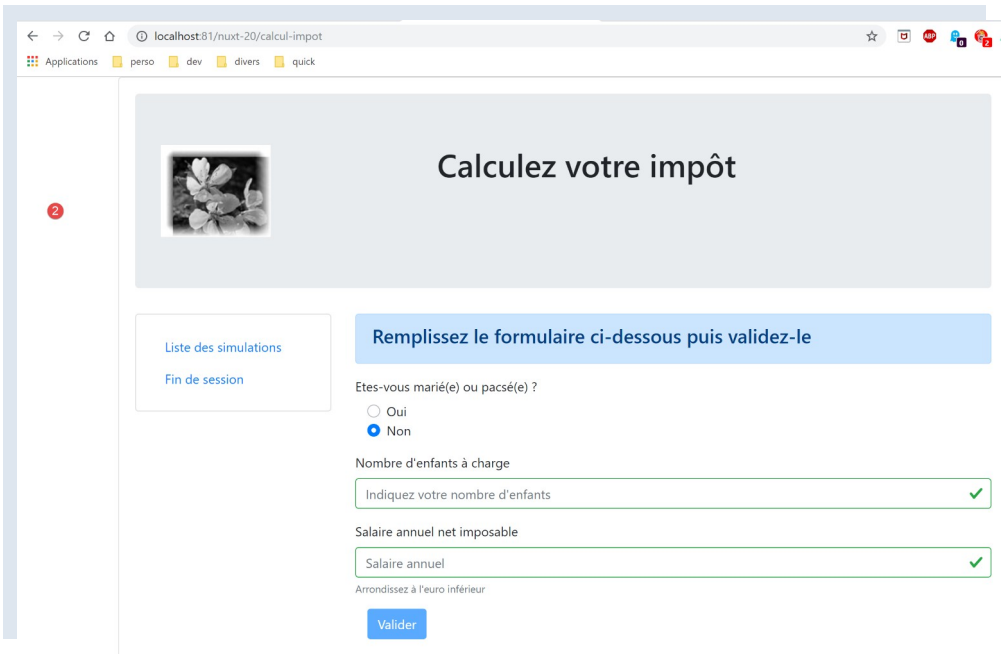
```
69.           this.$store.commit('replace', { métier: this.$métier() })
70.           // on sauvegarde la session
71.           this.$session().save(this.$nuxt.context)
72.         }
73.       }
74.     }
```

- lignes 43, 61, 69, [this.$métier] a été remplacé par [this.$métier()] ;
- lignes 8, 63 : le nom de la page [CalculImpot] du projet [vuejs-22] est devenue la page [calcul-impot] dans le projet [nuxt-20] ;

Ces corrections faites, on peut tenter de valider la page d'authentification :



La page obtenue est la suivante :



On a bien obtenu la page de calcul de l'impôt. Maintenant regardons les logs :

En [2], on voit que le fait d'être authentifié a été correctement mémorisé. En [3-4], on voit qu'on a récupéré la donnée [taxAdminData] qui permet le calcul de l'impôt par la classe [Métier].

17.5 étape 4

Examinons la page [calcul-impot] que nous avons obtenue :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <div>
4.      <Layout :left="true" :right="true">
5.        <!-- formulaire de calcul de l'impôt à droite -->
6.        <FormCalculImpot slot="right" @resultatObtenu="handleResultatObtenu" />
7.        <!-- menu de navigation à gauche -->
8.        <Menu slot="left" :options="options" />
9.      </Layout>
10.     <!-- zone d'affichage des résultat du calcul de l'impôt sous le formulaire -->
11.     <b-row v-if="résultatObtenu" class="mt-3">
12.       <!-- zone de trois colonnes vide -->
13.       <b-col sm="3" />
14.       <!-- zone de neuf colonnes -->
15.       <b-col sm="9">
```

```
16.         <b-alert show variant="success">
17.           <span v-html="résultat"></span>
18.         </b-alert>
19.       </b-col>
20.     </b-row>
21.   </div>
22. </template>
23.
24. <script>
25. // imports
26. import FormCalculImpot from '@/components/form-calcul-impot'
27. import Menu from '@/components/menu'
28. import Layout from '@/components/layout'
29.
30. export default {
31.   // composants utilisés
32.   components: {
33.     Layout,
34.     FormCalculImpot,
35.     Menu
36.   },
37.   // état interne
38.   data() {
39.     return {
40.       // options du menu
41.       options: [
42.         {
43.           text: 'Liste des simulations',
44.           path: '/liste-des-simulations'
45.         },
46.         {
47.           text: 'Fin de session',
48.           path: '/fin-session'
49.         }
50.       ],
51.       // résultat du calcul de l'impôt
52.       résultat: '',
53.       résultatObtenu: false
54.     }
55.   },
56.   // cycle de vie
57.   created() {
58.     // eslint-disable-next-line
59.     console.log("CalculImpot created");
60.   },
61.   // méthodes de gestion des évts
62.   methods: {
63.     // résultat du calcul de l'impôt
64.     handleResultatObtenu(résultat) {
65.       // on construit le résultat en chaîne HTML
66.       const impôt = "Montant de l'impôt : " + résultat.impôt + ' euro(s)'
67.       const décôte = 'Décôte : ' + résultat.décôte + ' euro(s)'
68.       const réduction = 'Réduction : ' + résultat.réduction + ' euro(s)'
69.       const surcôte = 'Surcôte : ' + résultat.surcôte + ' euro(s)'
70.       const taux = "Taux d'imposition : " + résultat.taux
71.       this.résultat = impôt + '<br/>' + décôte + '<br/>' + réduction + '<br/>' + surcôte +
    '<br/>' + taux
72.       // affichage du résultat
73.       this.résultatObtenu = true
74.       // ---- maj du store [Vuex]
75.       // une simulation de +
76.       this.$store.commit('addSimulation', résultat)
77.       // on sauvegarde la session
78.       this.$session.save()
79.     }
80.   }
81. }
82. </script>
```

- lignes 44 et 48 : les liens du menu de navigation sont corrects. La page [/fin-session] n'existe pas. Le projet [vuejs-22] réglait ce problème avec du routage. Nous ferons de même avec le projet [nuxt-20] ;
- ligne 76 : on référence une mutation [addSimulation] qui n'existe pas pour l'instant. Nous allons la créer ;
- ligne 78 : comme dans la page [index], il faut écrire [`this.$session().save(this.$nuxt.context)`] ;

Modifions le store [store/index]. Hérité du projet [nuxt-12], il est pour l'instant le suivant :

```
1.  /* eslint-disable no-console */
2.
3.  // état du store
4.  export const state = () => ({
5.    // session jSON démarrée
6.    jsonSessionStarted: false,
7.    // utilisateur authentifié
8.    userAuthenticated: false,
9.    // cookie de session PHP
10.   phpSessionCookie: '',
11.   // adminData
12.   adminData: ''
13. })
14.
15. // mutations du store
16. export const mutations = {
17.   // remplacement du state
18.   replace(state, newState) {
19.     for (const attr in newState) {
20.       state[attr] = newState[attr]
21.     }
22.   },
23.   // reset du store
24.   reset() {
25.     this.commit('replace', { jsonSessionStarted: false, userAuthenticated: false,
    phpSessionCookie: '', adminData: '' })
26.   }
27. }
28.
29. // actions du store
30. export const actions = {
31.   nuxtServerInit(store, context) {
32.     // qui exécute ce code ?
33.     console.log('nuxtServerInit, client=', process.client, 'serveur=', process.server, 'env=',
    context.env)
34.     // init session
35.     initStore(store, context)
36.   }
37. }
38.
39. function initStore(store, context) {
40.   // store est le store à initialiser
41.   // on récupère la session
42.   const session = context.app.$session()
43.   // la session a-t-elle été déjà initialisée ?
44.   if (!session.value.initStoreDone) {
45.     // on démarre un nouveau store
46.     console.log("nuxtServerInit, initialisation d'un nouveau store")
47.     // on met le store dans la session
48.     session.value.store = store.state
49.     // le store est désormais initialisé
50.     session.value.initStoreDone = true
51.   } else {
52.     console.log("nuxtServerInit, reprise d'un store existant")
53.     // on met à jour le store avec le store de la session
54.     store.commit('replace', session.value.store)
55.   }
56.   // on sauvegarde la session
57.   session.save(context)
58.   // log
59.   console.log('initStore terminé, store=', store.state)
60. }
```

- lignes 3-27 : nous allons reprendre le state et les mutations de l'application [vuejs-22] (cf document [3]) :

```
1.  // état du store
2.  export const state = () => ({
3.    // session jSON démarrée
4.    started: false,
5.    // utilisateur authentifié
```

```
6.    authenticated: false,
7.    // cookie de session PHP
8.    phpSessionCookie: '',
9.    // liste des simulations
10.   simulations: [],
11.   // le n° de la dernière simulation
12.   idSimulation: 0,
13.   // couche [métier]
14.   métier: null
15. })
16.
17. // mutations du store
18. export const mutations = {
19.   // remplacement du state
20.   replace(state, newState) {
21.     for (const attr in newState) {
22.       state[attr] = newState[attr]
23.     }
24.   },
25.   // reset du store
26.   reset() {
27.         this.commit('replace', { started: false, authenticated: false, phpSessionCookie: '',
    idSimulation: 0, simulations: [], métier: null })
28.   },
29.   // suppression ligne n° index
30.   deleteSimulation(state, index) {
31.     // eslint-disable-next-line no-console
32.     console.log('mutation deleteSimulation')
33.     // on supprime la ligne n° [index]
34.     state.simulations.splice(index, 1)
35.     console.log('store simulations', state.simulations)
36.   },
37.   // ajout d'une simulation
38.   addSimulation(state, simulation) {
39.     // eslint-disable-next-line no-console
40.     console.log('mutation addSimulation')
41.     // n° de la simulation
42.     state.idSimulation++
43.     simulation.id = state.idSimulation
44.     // on ajoute la simulation au tableau des simulations
45.     state.simulations.push(simulation)
46.   }
47. }
```

- lignes 4 et 6 : nous introduisons les propriétés déjà utilisées ;
- ligne 8 : nous gardons le cookie de session PHP. Il est fondamental pour que le client et le serveur [nuxt] aient la même session PHP avec le serveur de calcul de l'impôt ;
- ligne 10 : la liste des simulations faites par l'utilisateur ;
- ligne 12 : le n° de la dernière simulation faite par l'utiliasteur ;
- ligne 14 : la couche [métier] ;
- lignes 30-47 : les mutations présentes dans le store du projet [vuejs-22] et référencées par les pages de l'application. Le projet [vuejs-22] avait une mutation appelée [clear] qui vidait la liste des simulations. On ne la met pas car la mutation [reset] déjà présente devrait faire l'affaire ;
- lignes 26-28 : la mutation [reset] est modifiée pour prendre en compte le nouveau contenu du state ;

La page [calcul-impot] utilise le composant [form-calcul-impot] suivant :

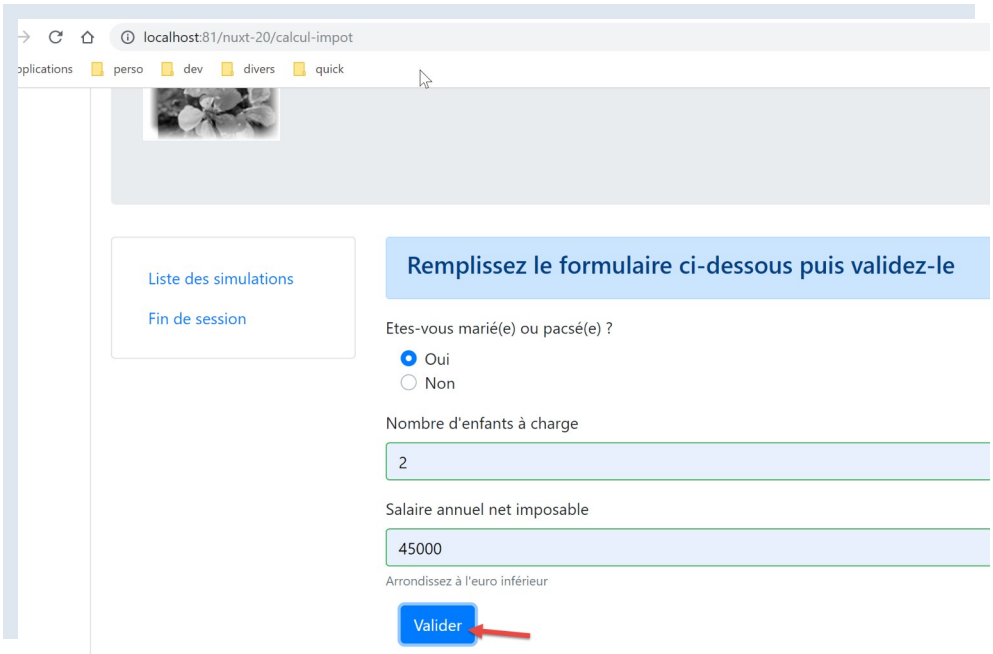
```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <!-- formulaire HTML -->
4.    <b-form @submit.prevent="calculerImpot" class="mb-3">
5.      <!-- message sur 12 colonnes sur fond bleu -->
6.      <b-row>
7.        <b-col sm="12">
8.          <b-alert show variant="primary">
9.            <h4>Remplissez le formulaire ci-dessous puis validez-le</h4>
10.         </b-alert>
11.       </b-col>
12.     </b-row>
13.     <!-- éléments du formulaire -->
14.     <!-- première ligne -->
15.     <b-form-group label="Etes-vous marié(e) ou pacsé(e) ?">
```

```
16.       <!-- boutons radio sur 5 colonnes-->
17.       <b-col sm="5">
18.         <b-form-radio v-model="marié" value="oui">Oui</b-form-radio>
19.         <b-form-radio v-model="marié" value="non">Non</b-form-radio>
20.       </b-col>
21.     </b-form-group>
22.     <!-- deuxième ligne -->
23.     <b-form-group label="Nombre d'enfants à charge" label-for="enfants">
24.       <b-form-input id="enfants" v-model="enfants" :state="enfantsValide" type="text"
    placeholder="Indiquez votre nombre d'enfants"></b-form-input>
25.       <!-- message d'erreur éventuel -->
26.       <b-form-invalid-feedback :state="enfantsValide">Vous devez saisir un nombre positif ou
    nul</b-form-invalid-feedback>
27.     </b-form-group>
28.     <!-- troisème ligne -->
29.     <b-form-group label="Salaire annuel net imposable" label-for="salaire"
    description="Arrondissez à l'euro inférieur">
30.       <b-form-input id="salaire" v-model="salaire" :state="salaireValide" type="text"
    placeholder="Salaire annuel"></b-form-input>
31.       <!-- message d'erreur éventuel -->
32.       <b-form-invalid-feedback :state="salaireValide">Vous devez saisir un nombre positif ou
    nul</b-form-invalid-feedback>
33.     </b-form-group>
34.     <!-- quatrième ligne, bouton [submit] -->
35.     <b-col sm="3">
36.       <b-button :disabled="formInvalide" type="submit" variant="primary">Valider</b-button>
37.     </b-col>
38.   </b-form>
39. </template>
40.
41. <!-- script -->
42. <script>
43. export default {
44.   // état interne
45.   data() {
46.     return {
47.       // marié ou pas
48.       marié: 'non',
49.       // nombre d'enfants
50.       enfants: '',
51.       // salaire annuel
52.       salaire: ''
53.     }
54.   },
55.   // état interne calculé
56.   computed: {
57.     // validation du formulaire
58.     formInvalide() {
59.       return (
60.         // salaire invalide
61.         !this.salaire.match(/^\s*\d+\s*$/) ||
62.         // ou enfants invalide
63.         !this.enfants.match(/^\s*\d+\s*$/) ||
64.         // ou données fiscales pas obtenues
65.         !this.$métier.taxAdminData
66.       )
67.     },
68.     // validation du salaire
69.     salaireValide() {
70.       // doit être numérique >=0
71.       return Boolean(this.salaire.match(/^\s*\d+\s*$/) || this.salaire.match(/^\s*$/))
72.     },
73.     // validation des enfants
74.     enfantsValide() {
75.       // doit être numérique >=0
76.       return Boolean(this.enfants.match(/^\s*\d+\s*$/) || this.enfants.match(/^\s*$/))
77.     }
78.   },
79.   // cycle de vie
80.   created() {
81.     // log
82.     // eslint-disable-next-line
83.     console.log("FormCalculImpot created");
84.   },
```

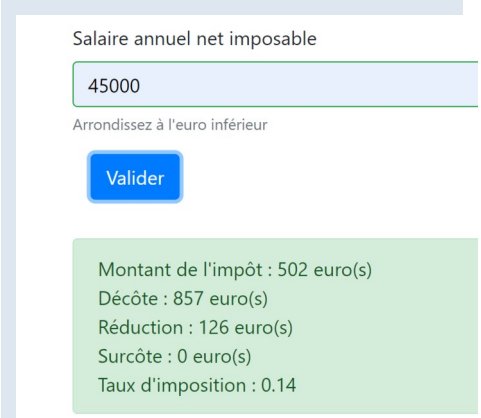
```
85.   // gestionnaire d'évts
86.   methods: {
87.     calculerImpot() {
88.       // on calcule l'impôt à l'aide de la couche [métier]
89.       const résultat = this.$métier.calculerImpot(this.marié, Number(this.enfants),
    Number(this.salaire))
90.       // eslint-disable-next-line
91.       console.log("résultat=", résultat);
92.       // on complète le résultat
93.       résultat.marié = this.marié
94.       résultat.enfants = this.enfants
95.       résultat.salaire = this.salaire
96.       // on émet l'évt [resultatObtenu]
97.       this.$emit('resultatObtenu', résultat)
98.     }
99.   }
100.}
101.</script>
```

- lignes 65, 89 : la référence [this.$métier] doit être changée en [**this.$métier()**] ;

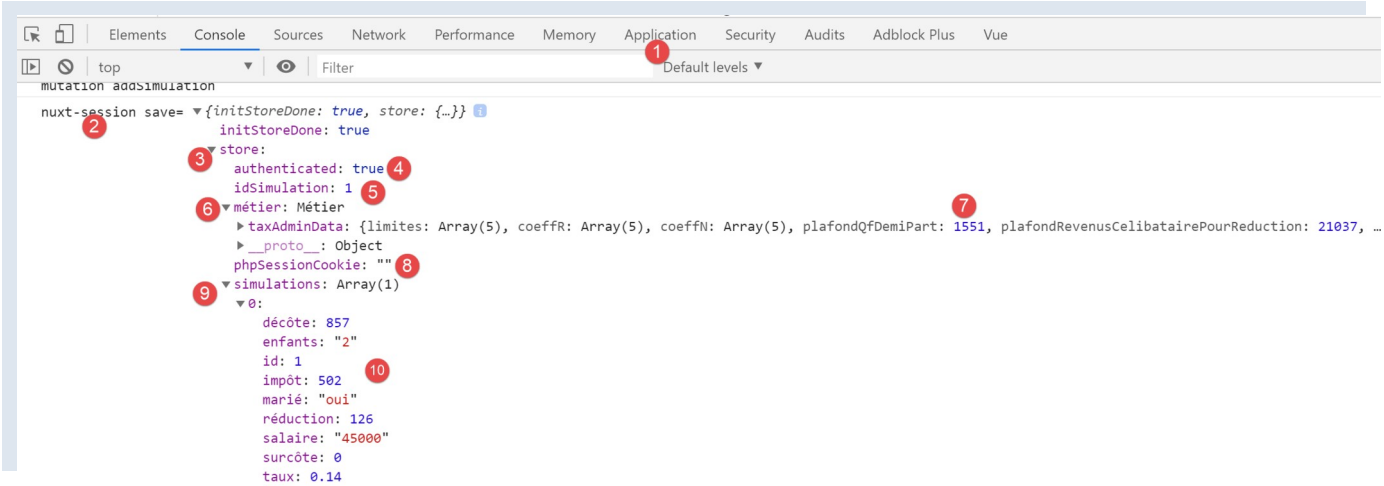
Ces corrections faites, on peut tenter une simulation :



On obtient la réponse suivante :



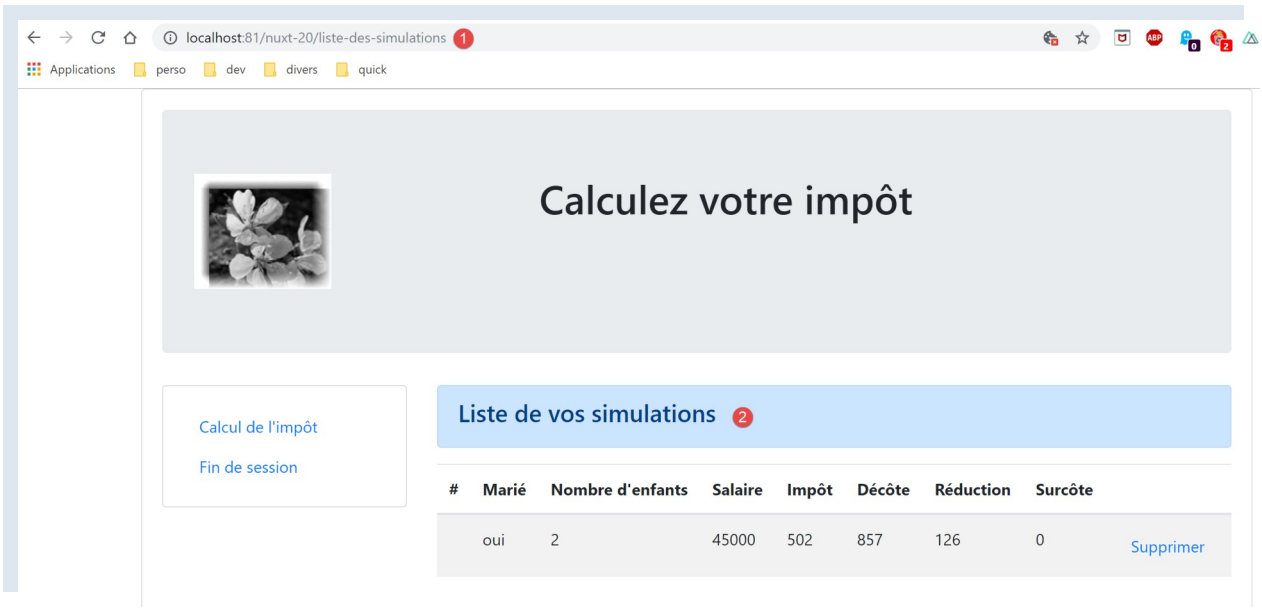
Si on regarde les logs :



- en [9-10], on voit que la 1ère simulation se trouve bien dans le [store] ;
- en [5], le n° de la dernière simulation a bien été incrémenté ;

17.6 étape 5

Maintenant que nous avons fait une simulation, cliquons sur le lien [Liste des simulations]. Nous obtenons la page suivante :



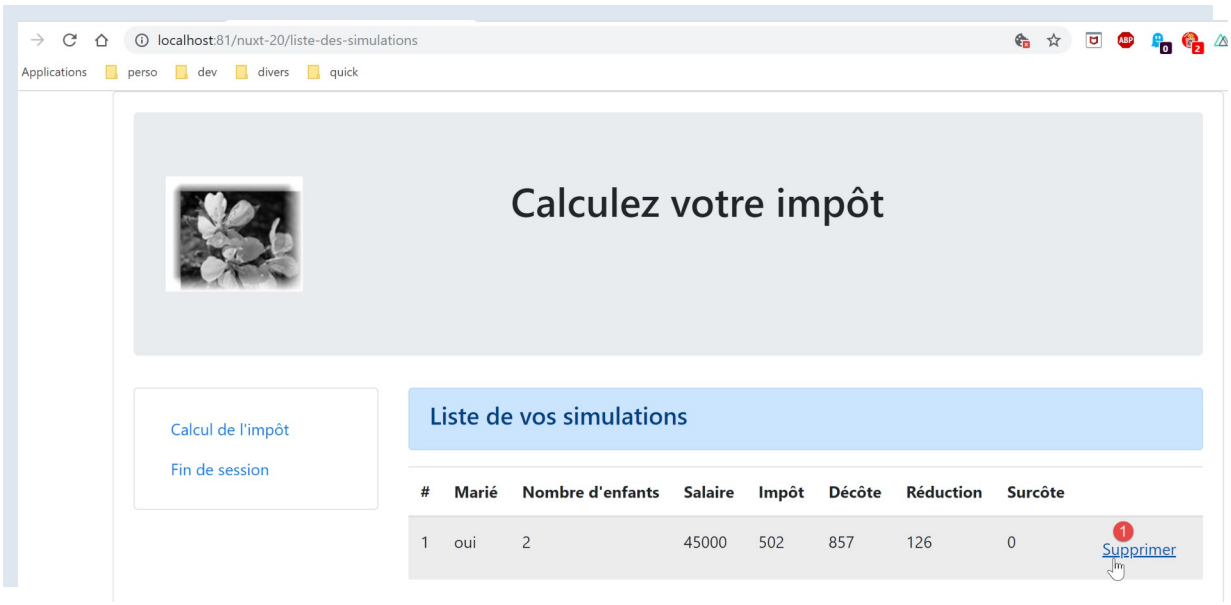
Le routage du client [nuxt] s'est fait correctement. Regardons le code de la page [liste-des-simulations] :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <div>
4.      <!-- mise en page -->
5.      <Layout :left="true" :right="true">
6.        <!-- simulations dans colonne de droite -->
7.        <template slot="right">
8.          <template v-if="simulations.length == 0">
9.            <!-- pas de simulations -->
10.           <b-alert show variant="primary">
11.             <h4>Votre liste de simulations est vide</h4>
12.           </b-alert>
13.         </template>
14.         <template v-if="simulations.length != 0">
15.           <!-- il y a des simulations -->
16.           <b-alert show variant="primary">
17.             <h4>Liste de vos simulations</h4>
18.           </b-alert>
19.           <!-- tableau des simulations -->
20.           <b-table :items="simulations" :fields="fields" striped hover responsive>
21.             <template v-slot:cell(action)="data">
22.               <b-button @click="supprimerSimulation(data.index)" variant="link">Supprimer</b-
    button>
23.             </template>
24.           </b-table>
25.         </template>
26.       </template>
27.       <!-- menu de navigation dans colonne de gauche -->
28.       <Menu slot="left" :options="options" />
29.     </Layout>
30.   </div>
31. </template>
32.
33. <script>
34. // imports
35. import Layout from '@/components/layout'
36. import Menu from '@/components/menu'
37. export default {
38.   // composants
39.   components: {
40.     Layout,
41.     Menu
42.   },
43.   // état interne
44.   data() {
```

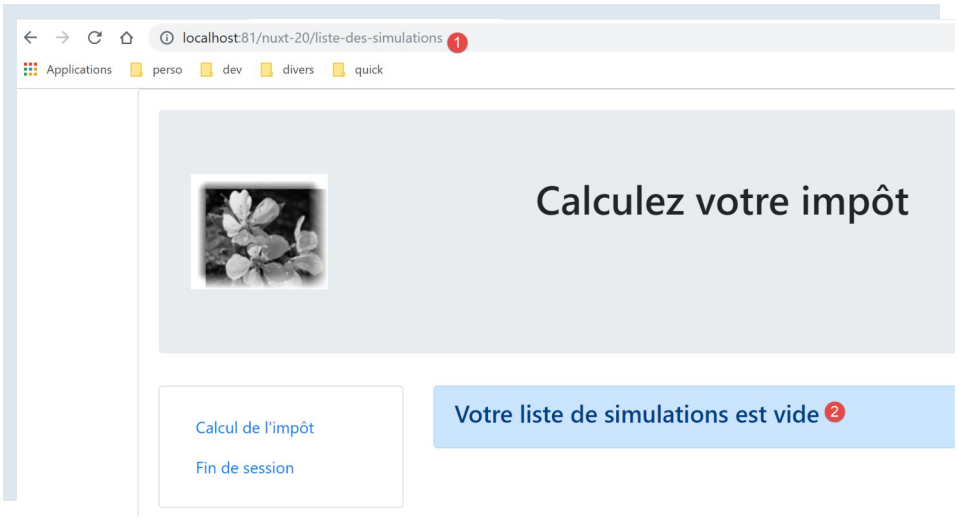
```
45.     return {
46.       // options du menu de navigation
47.       options: [
48.         {
49.           text: "Calcul de l'impôt",
50.           path: '/calcul-impot'
51.         },
52.         {
53.           text: 'Fin de session',
54.           path: '/fin-session'
55.         }
56.       ],
57.       // paramètres de la table HTML
58.       fields: [
59.         { label: '#', key: 'id' },
60.         { label: 'Marié', key: 'marié' },
61.         { label: "Nombre d'enfants", key: 'enfants' },
62.         { label: 'Salaire', key: 'salaire' },
63.         { label: 'Impôt', key: 'impôt' },
64.         { label: 'Décôte', key: 'décôte' },
65.         { label: 'Réduction', key: 'réduction' },
66.         { label: 'Surcôte', key: 'surcôte' },
67.         { label: '', key: 'action' }
68.       ]
69.     }
70.   },
71.   // état interne calculé
72.   computed: {
73.     // liste des simulations prise dans le store Vuex
74.     simulations() {
75.       return this.$store.state.simulations
76.     }
77.   },
78.   // cycle de vie
79.   created() {
80.     // eslint-disable-next-line
81.     console.log("ListeSimulations created");
82.   },
83.   // méthodes
84.   methods: {
85.     supprimerSimulation(index) {
86.       // eslint-disable-next-line
87.       console.log("supprimerSimulation", index);
88.       // suppression de la simulation n° [index]
89.       this.$store.commit('deleteSimulation', index)
90.       // on sauvegarde la session
91.       this.$session.save()
92.     }
93.   }
94. }
95. </script>
```

- lignes 47-56 : les cibles du menu de navigation sont correctes ;
- ligne 75 : le store est correctement référencé ;
- ligne 89 : on utilise une mutation [`deleteSimulation`] que nous avons intégrée lors de l'étape précédente ;
- ligne 91 : cette ligne doit être réécrite comme [`this.$session().save(this.$nuxt.context)`] ;

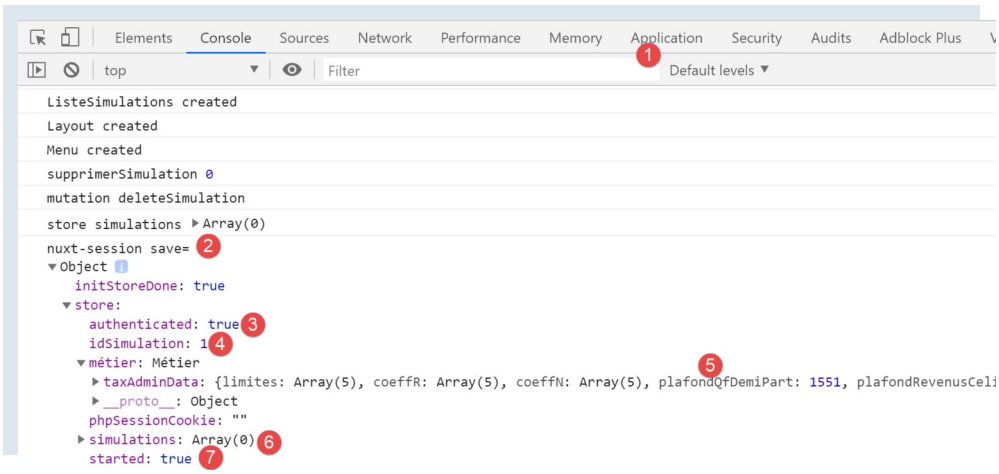
Nous faisons les modifications nécessaires, puis nous tentons de supprimer la simulation affichée :



On obtient alors la page suivante :

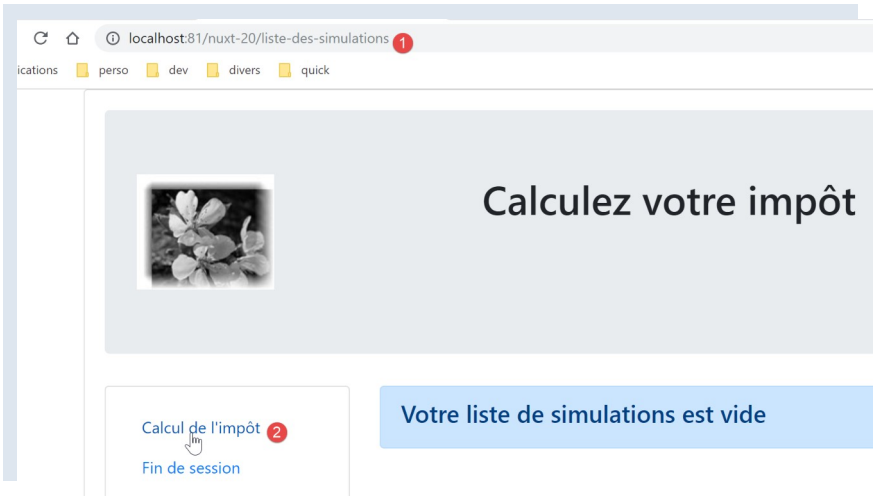


Regardons les logs :

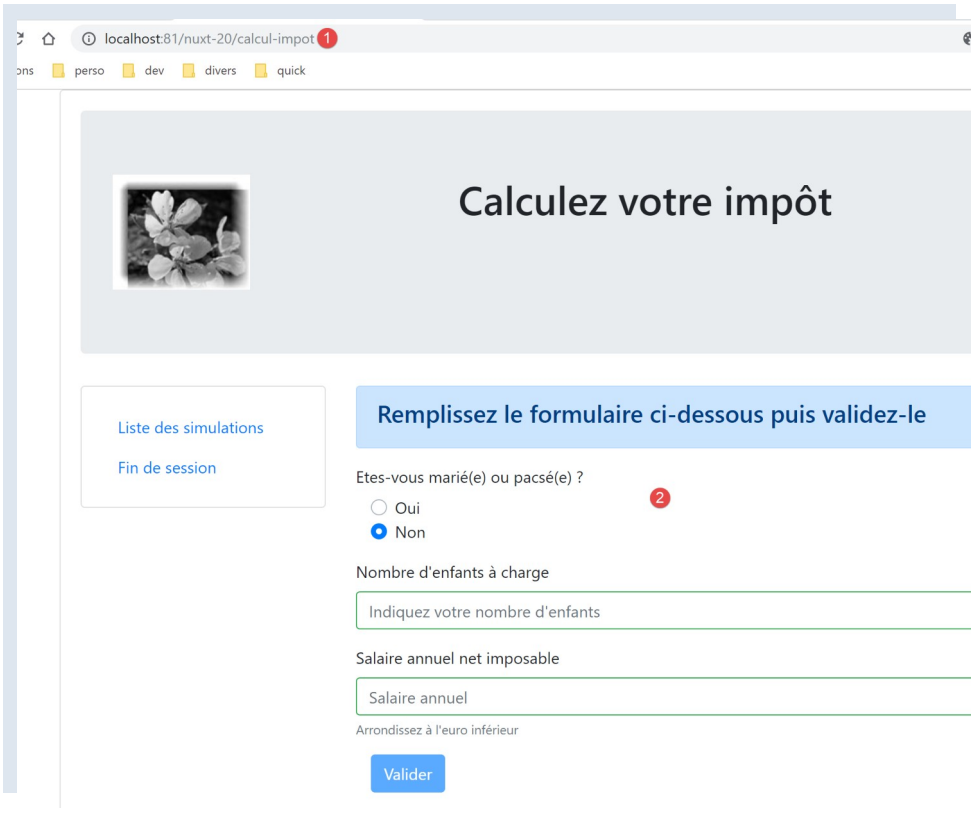


- en [6], on voit que le tableau des simulations est vide ;

Maintenant revenons au formulaire de calcul de l'impôt :



On obtient la page suivante :



Donc le routage a fonctionné.

17.7 étape 6

Il nous reste à gérer l'option de navigation [Fin de session] du menu de navigation :

```
1.  // options du menu
```

```
2.        options: [
3.          {
4.            text: 'Liste des simulations',
5.            path: '/liste-des-simulations'
6.          },
7.          {
8.            text: 'Fin de session',
9.            path: '/fin-session'
10.         }
11. ]
```

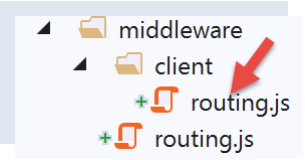
- ligne 9, la page [/fin-session] n'existe pas. Le projet [vuejs-22] gérait ce cas avec des règles de routage dans un fichier [router.js]:

```
1.  // imports
2.  import Vue from 'vue'
3.  import VueRouter from 'vue-router'
4.  // les vues
5.  import Authentification from './views/Authentification'
6.  import CalculImpot from './views/CalculImpot'
7.  import ListeSimulations from './views/ListeSimulations'
8.  import NotFound from './views/NotFound'
9.  // la session
10. import session from './session'
11.
12. // plugin de routage
13. Vue.use(VueRouter)
14.
15. // les routes de l'application
16. const routes = [
17.   // authentification
18.   { path: '/', name: 'authentification', component: Authentification },
19.   { path: '/authentification', name: 'authentification2', component: Authentification },
20.   // calcul de l'impôt
21.   {
22.     path: '/calcul-impot', name: 'calculImpot', component: CalculImpot,
23.     meta: { authenticated: true }
24.   },
25.   // liste des simulations
26.   {
27.     path: '/liste-des-simulations', name: 'listeSimulations', component: ListeSimulations,
28.     meta: { authenticated: true }
29.   },
30.   // fin de session
31.   {
32.     path: '/fin-session', name: 'finSession'
33.   },
34.   // page inconnue
35.   {
36.     path: '*', name: 'notFound', component: NotFound,
37.   },
38. ]
39.
40. // le routeur
41. const router = new VueRouter({
42.   // les routes
43.   routes,
44.   // le mode d'affichage des URL
45.   mode: 'history',
46.   // l'URL de base de l'application
47.   base: '/client-vuejs-impot/'
48. })
49.
50. // vérification des routes
51. router.beforeEach((to, from, next) => {
52.   // eslint-disable-next-line no-console
53.   console.log("router to=", to, "from=", from);
54.   // route réservée aux utilisateurs authentifiés ?
55.   if (to.meta.authenticated && !session.authenticated) {
56.     next({
57.       // on passe à l'authentification
58.       name: 'authentification',
59.     })
60.     // retour à la boucle événementielle
```

```
61.     return;
62.   }
63.   // cas particulier de la fin de session
64.   if (to.name === "finSession") {
65.     // on nettoie la session
66.     session.clear();
67.     // on va sur la vue [authentification]
68.     next({
69.       name: 'authentification',
70.     })
71.     // retour à la boucle événementielle
72.     return;
73.   }
74.   // autres cas - vue suivante normale du routage
75.   next();
76. })
77.
78. // export du router
79. export default router
```

- les lignes 64-76 géraient le cas particulier de la route vers le chemin [/fin-session] ;
- ligne 66 : on vide la session courante ;
- lignes 68-70 : on affiche la vue [authentification] ;

Nous allons essayer de faire quelque chose d'analogue dans le fichier de routage du client [nuxt] :



Le script [client/routing.js] devient le suivant :

```
1.  /* eslint-disable no-console */
2.  export default function(context) {
3.    // qui exécute ce code ?
4.    console.log('[middleware client], process.server', process.server, ', process.client=',
    process.client)
5.    // gestion du cookie de la session PHP dans le navigateur
6.    // le cookie de la session PHP du navigateur doit être identique à celui trouvé en session
    nuxt
7.    // l'action [fin-session] reçoit un nouveau cookie PHP (serveur comme client nuxt)
8.    // si c'est le serveur qui le reçoit, le client doit le transmettre au navigateur
9.    // pour ses propres échanges avec le serveur PHP
10.   // on est ici dans un routing client
11.
12.   // on récupère le cookie de la session PHP
13.   const phpSessionCookie = context.store.state.phpSessionCookie
14.   if (phpSessionCookie) {
15.     // s'il existe, on affecte le cookie de session PHP au navigateur
16.     document.cookie = phpSessionCookie
17.   }
18.
19.   // où va-t-on ?
20.   const to = context.route.path
21.   if (to === '/fin-session') {
22.     // on nettoie la session
23.     const session = context.app.$session()
24.     session.reset(context)
25.     // on redirige vers la page index
26.     context.redirect({ name: 'index' })
27.   }
28. }
```

- on a rajouté les lignes [19-27] au code existant ;
- ligne 20 : on récupère le [path] de la cible de la route courante ;

- ligne 21 : on regarde si c'est [/fin-session]. Si oui :
  - lignes 23-24 : la session est réinitialisée ;
  - ligne 26 : on redirige le client [nuxt] vers la page d'accueil ;

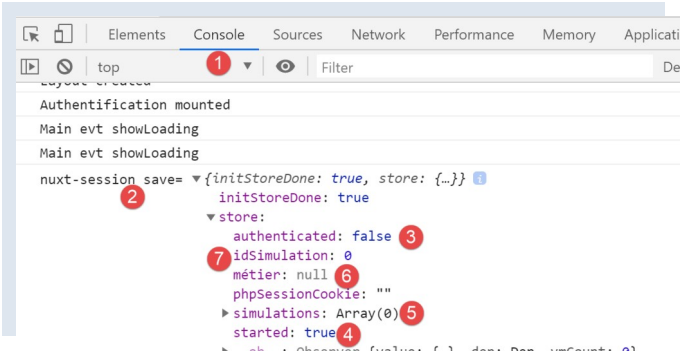
La méthode [`session.reset(context)`] (ligne 24) de la session est la suivante :

```
1.  // reset de la session
2.    reset(context) {
3.      console.log('nuxt-session reset')
4.      // reset du store
5.      context.store.commit('reset')
6.      // sauvegarde du nouveau store en session et sauvegarde de la session
7.      this.save(context)
8.  }
```

La méthode [`context.store.commit('reset')`] (ligne 5) est la suivante :

```
1.  // reset du store
2.    reset() {
3.          this.commit('replace', { started: false, authenticated: false, phpSessionCookie: '',
    idSimulation: 0, simulations: [], métier: null })
4.  }
```

Lorsqu'on utilise maintenant le lien [Fin de session], la page d'accueil est affichée avec les logs suivants :



- en [3], on voit qu'on n'est plus authentifié ;
- en [4], on voit que la session jSON est démarrée ;
- en [6], la couche [métier] n'est plus présente dans le store (elle est toujours présente dans les pages avec [this.$métier()]) ;
- en [5, 7], il n'y a plus de simulations ;

Il faut bien comprendre ce qui se passe dans une fin de session :

- la session [nuxt] est réinitialisée : la propriété [started] du store passe à [false] ;
- il y a redirection vers la page [index] ;
- la méthode [mounted] de la page [index] est exécutée. Celle-ci démarre une nouvelle session jSON avec le serveur de calcul de l'impôt. Si l'opération réussit, la propriété [started] du store passe à [true] ;

17.8 étape 7

A ce stade, l'application [nuxt-20] a toutes les fonctionnalités de l'application [vuejs-22]. Le portage semble terminé.

Nous allons aller un peu plus loin dans un esprit [nuxt]. La méthode [mounted] de la page [index] pose problème. Elle lance une opération asynchrone dont un moteur de recherche n'attendra pas la fin. On sait que dans ce cas là, il faut mettre l'opération asynchrone dans une fonction [asyncData] car alors, le serveur [nuxt] qui l'exécute attend qu'elle soit terminée avant de délivrer la page au moteur de recherche.

Nous nous aidons ici de la fonction [asyncData] écrite dans l'application [nuxt-12] pour la page [index] :

```
1.  export default {
2.    name: 'InitSession',
3.    // composants utilisés
4.    components: {
5.      Layout,
6.      Navigation
7.    },
8.    // données asynchrones
9.    async asyncData(context) {
10.     // log
11.     console.log('[index asyncData started]')
12.     // on ne fait pas les choses deux fois si la page a déjà été demandée
13.     if (process.server && context.store.state.jsonSessionStarted) {
14.       console.log('[index asyncData canceled]')
15.       return { result: '[succès]' }
16.     }
17.     try {
18.       // on démarre une session jSON
19.       const dao = context.app.$dao()
20.       const response = await dao.initSession()
21.       // log
22.       console.log('[index asyncData response=]', response)
23.       // on récupère le cookie de session PHP pour les prochaines requêtes
24.       const phpSessionCookie = dao.getPhpSessionCookie()
25.       // on mémorise le cookie de session PHP dans la session [nuxt]
26.       context.store.commit('replace', { phpSessionCookie })
27.       // y-a-t-il eu erreur ?
28.       if (response.état !== 700) {
29.         // l'erreur se trouve dans response.réponse
30.         throw new Error(response.réponse)
31.       }
32.       // on note le fait que la session jSON a démarré
33.       context.store.commit('replace', { jsonSessionStarted: true })
34.       // on rend le résultat
35.       return { result: '[succès]' }
36.     } catch (e) {
37.       // log
38.       console.log('[index asyncData error=]', e)
39.       // on note le fait que la session jSON n'a pas démarré
40.       context.store.commit('replace', { jsonSessionStarted: false })
41.       // on signale l'erreur
42.       return { result: '[échec]', showErrorLoading: true, errorLoadingMessage: e.message }
43.     } finally {
44.       // on sauvegarde le store
45.       const session = context.app.$session()
46.       session.save(context)
47.       // log
48.       console.log('[index asyncData finished]')
49.     }
50.   },
51.   // cycle de vie
52.   beforeCreate() {
53.     console.log('[index beforeCreate]')
54.   },
55.   created() {
56.     console.log('[index created]')
57.   },
58.   beforeMount() {
59.     console.log('[index beforeMount]')
60.   },
61.   mounted() {
62.     console.log('[index mounted]')
63.     // client seulement
64.     if (this.showErrorLoading) {
65.       console.log('[index mounted, showErrorLoading=true]')
66.       this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
67.     }
68. }
```

- lignes 13, 33, 40 : il faut changer la propriété [`jsonSessionStarted`] en [`started`] ;
- ligne 13 : dans l'application [nuxt-12], seul le serveur [nuxt] exécutait la page [index] et sa fonction [asyncData]. Le client [nuxt] n'exécutait la page [index] qu'après l'avoir reçue du serveur [nuxt] et alors il n'exécutait pas la fonction [asyncData]. Dans [nuxt-20], c'est différent : le lien [Fin de session] va afficher la

page [index] dans l'environnement du client [nuxt]. La fonction [asyncData] va alors être exécutée. Cependant, lorsqu'on arrive à la page [index] de cette façon, la session [nuxt] a été réinitialisée entre-temps et la propriété [started] du store vaut [false] et la condition de la ligne 13 sera forcément fausse. On peut donc laisser [process.server] et ainsi le client [nuxt] ne fera pas ce test ;

- lignes 15, 35, 42 : une propriété [result] est mise dans les propriétés [data] de la page [index]. Dans [nuxt-20], cette propriété ne sera pas utilisée et nous l'enlèverons du résultat rendu par la fonction ;
- lignes 61-67 : cette méthode [mounted] doit être conservée car c'est elle qui permet au client [nuxt] d'afficher le message d'erreur. Néanmoins la façon de gérer l'erreur sera modfiée ;

Dans la page [index] actuelle, nous intégrons la fonction [asyncData] ci-dessus en lieu et place de l'ancienne fonction [mounted] et nous ajoutons une nouvelle fonction [mounted]. Le code de la page [index] de l'exemple [nuxt-20] devient alors le suivant :

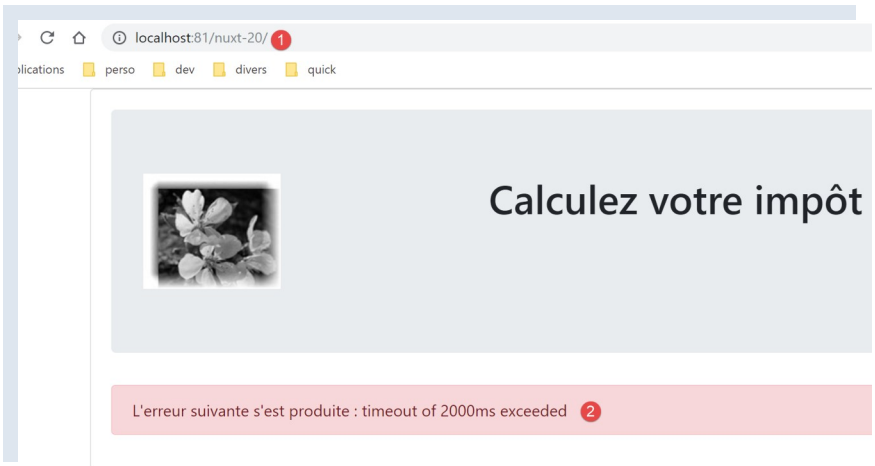
```
1.  ...
2.
3.  <!-- dynamique de la vue -->
4.  <script>
5.  /* eslint-disable no-console */
6.  import Layout from '@/components/layout'
7.  export default {
8.    // composants utilisés
9.    components: {
10.     Layout
11.   },
12.   // état du composant
13.   data() {
14.     return {
15.       // utilisateur
16.       user: '',
17.       // son mot de passe
18.       password: '',
19.       // affichage erreur
20.       showError: false
21.     }
22.   },
23.
24.   // propriétés calculées
25.   computed: {
26.     // saisies valides
27.     valid() {
28.       return this.user && this.password && this.$store.state.started
29.     }
30.   },
31.   // données asynchrones
32.   async asyncData(context) {
33.     // log
34.     console.log('[index asyncData started]')
35.     // on ne fait pas les choses deux fois si la page a déjà été demandée
36.     if (process.server && context.store.state.started) {
37.       console.log('[index asyncData canceled]')
38.       return
39.     }
40.     try {
41.       // on démarre une session jSON
42.       const dao = context.app.$dao()
43.       const response = await dao.initSession()
44.       // log
45.       console.log('[index asyncData response=]', response)
46.       // on récupère le cookie de session PHP pour les prochaines requêtes
47.       const phpSessionCookie = dao.getPhpSessionCookie()
48.       // on mémorise le cookie de session PHP dans la session [nuxt]
49.       context.store.commit('replace', { phpSessionCookie })
50.       // y-a-t-il eu erreur ?
51.       if (response.état !== 700) {
52.         // l'erreur se trouve dans response.réponse
53.         throw new Error(response.réponse)
54.       }
55.       // on note le fait que la session jSON a démarré
56.       context.store.commit('replace', { started: true })
57.       // pas de résultat
58.       return
59.     } catch (e) {
```

```
60.       // log
61.       console.log('[index asyncData error=]', e.message)
62.       // on note le fait que la session jSON n'a pas démarré
63.       context.store.commit('replace', { started: false })
64.       // on signale l'erreur
65.       return { showErrorLoading: true, errorLoadingMessage: e.message }
66.     } finally {
67.       // on sauvegarde le store
68.       const session = context.app.$session()
69.       session.save(context)
70.       // log
71.       console.log('[index asyncData finished]')
72.     }
73.   },
74.   // cycle de vie
75.   beforeCreate() {
76.     console.log('[index beforeCreate]')
77.   },
78.   created() {
79.     console.log('[index created]')
80.   },
81.   beforeMount() {
82.     // client seulement
83.     console.log('[index beforeMount]')
84.     // gestion de l'erreur éventuelle
85.     if (this.showErrorLoading) {
86.       // log
87.       console.log('[index beforeMount, showErrorLoading=true]')
88.       // on remonte l'erreur au composant principal [default]
89.       this.$emit('error', new Error(this.errorLoadingMessage))
90.     }
91.   },
92.   mounted() {
93.     console.log('[index mounted]')
94.   },
95.
96.   // gestionnaires d'évts
97.   methods: {
98.     // ---------- authentification
99.     async login() {
100.      ...
101.}
102.</script>
```

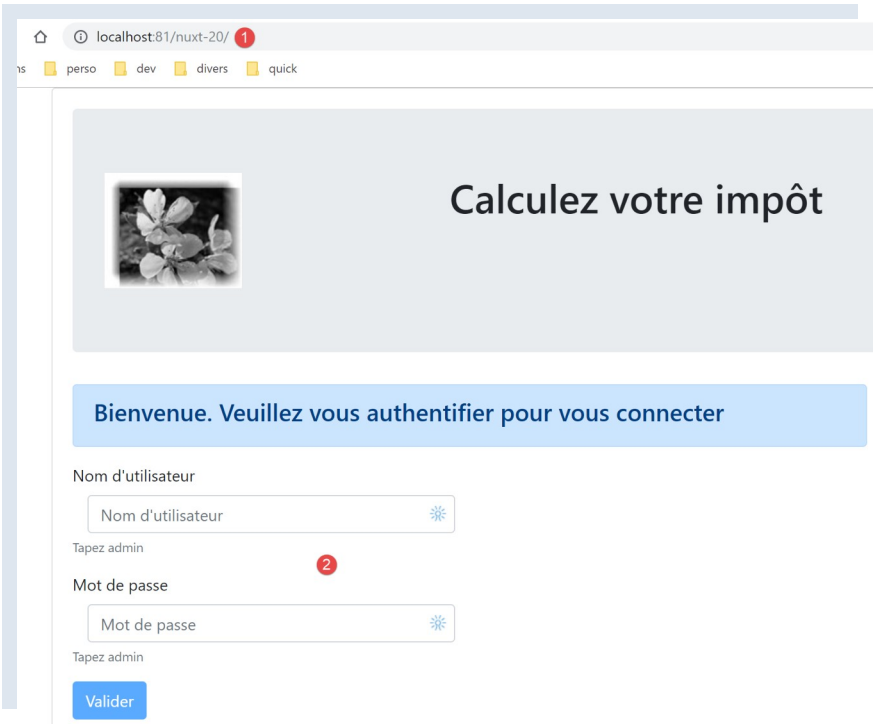
- lignes 58, 65 : la fonction [asyncData] ne rend plus la propriété [result] inutilisée ici ;
- ligne 81 : la méthode [beforeMount] du client [nuxt]. Elle a été préférée à la méthode [mounted] pour gérer l'erreur éventuelle de [asyncData] ;
- ligne 85 : on teste si la propriété [errorLoading] a été positionnée. Elle ne peut l'être que par la fonction [asyncData] ;
- lignes 85-90 : si fonction [asyncData] a signalé une erreur, on la passe à la page [default] via l'événement [error]. C'était comme cela que l'ancienne fonction [created] que nous venons de remplacer, gérait l'éventuelle erreur ;

Faisons quelques tests.

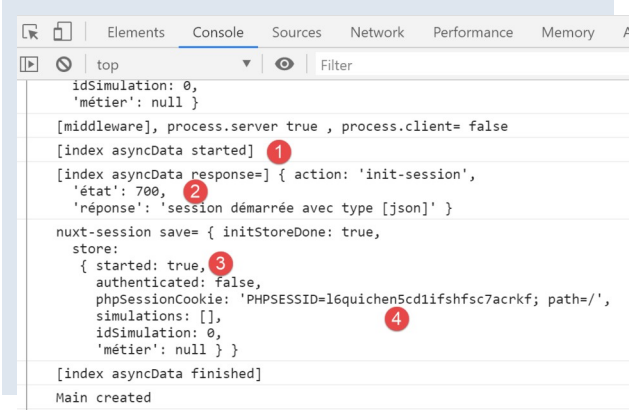
Nous supprimons d'abord et le cookie de la session [nuxt] et le cookie de session PHP s'ils existent. Nous demandons ensuite la page [http://localhost:81/nuxt-20/] alors que le serveur de calcul de l'impôt n'est pas lancé. Nous obtenons la page suivante :



Nous rechargeons la même page après avoir lancé le serveur de calcul de l'impôt :

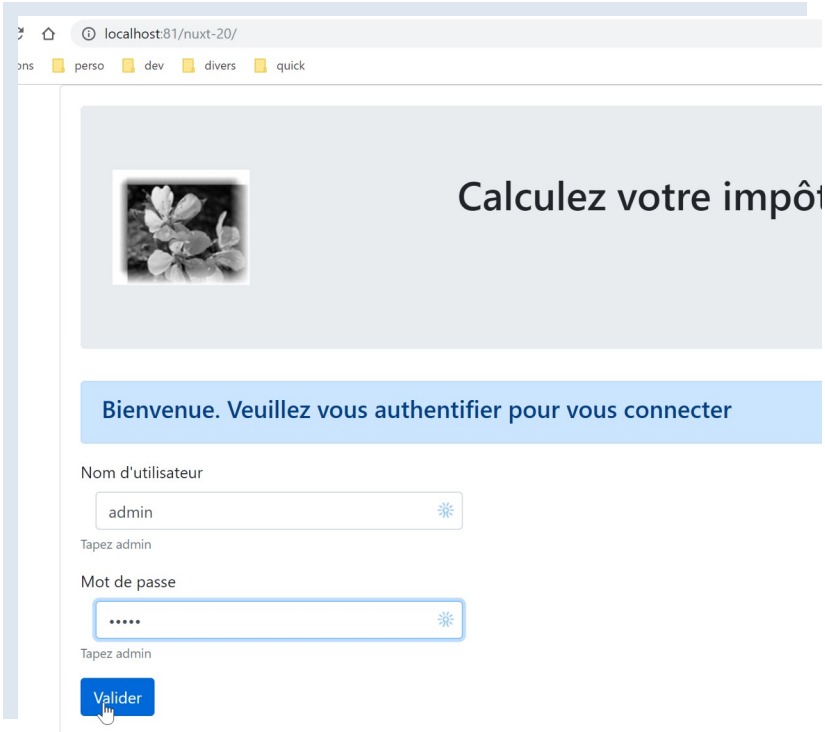


Regardons les logs :

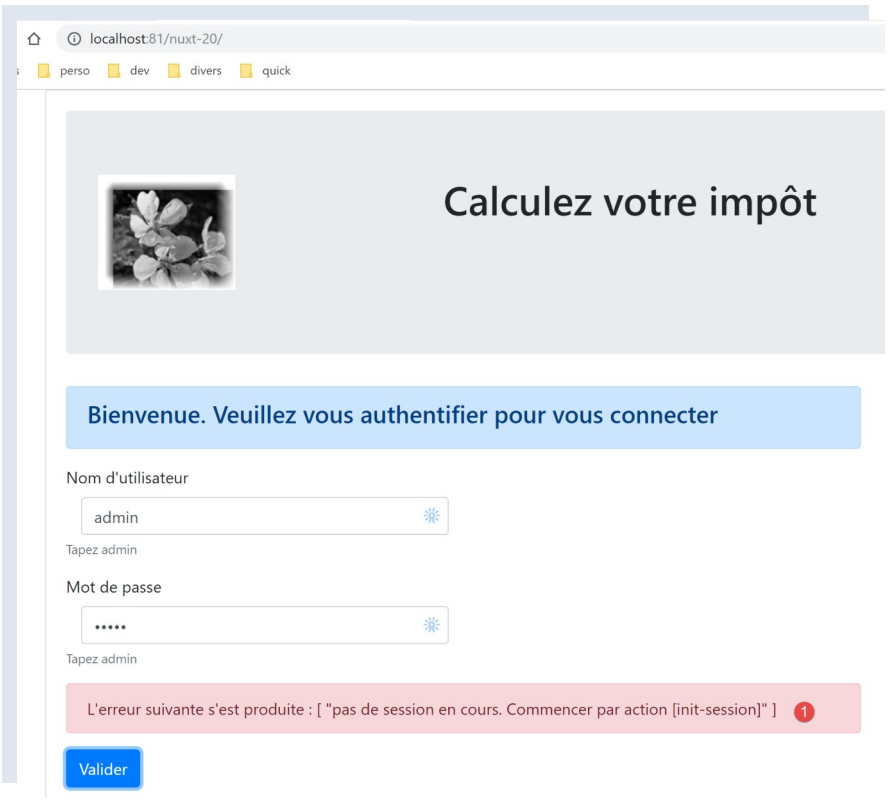


- en [2-3], on voit que la session jSON a été démarrée ;
- en [4], on voit le cookie de session PHP que le serveur [nuxt] a récupéré lors de son échange avec le serveur de calcul de l'impôt. Le client [nuxt] va désormais l'utiliser ;

Maintenant identifions-nous :



Nous obtenons la page suivante :



En [1], nous avons obtenu un message d'erreur. Cela veut dire que le navigateur n'a pas envoyé le bon cookie de la session PHP démarrée par le serveur [nuxt] à l'étape précédente. Dans [nuxt-12], le passage du cookie de session PHP du serveur [nuxt] au client [nuxt] se faisait dans le routage du client [nuxt] du script [middleware/client/routing] :

```
1.  /* eslint-disable no-console */
2.  export default function(context) {  // qui exécute ce code ?
3.    console.log('[middleware client], process.server', process.server, ', process.client=',
    process.client)
4.    // gestion du cookie de la session PHP dans le navigateur
5.    // le cookie de la session PHP du navigateur doit être identique à celui trouvé en session
    nuxt
6.    // l'acion [fin-session] reçoit un nouveau cookie PHP (serveur comme client nuxt)
7.    // si c'est le serveur qui le reçoit, le client doit le transmettre au navigateur
8.    // pour ses propres échanges avec le serveur PHP
9.    // on est ici dans un routing client
10.
11.   // on récupère le cookie de la session PHP
12.   const phpSessionCookie = context.store.state.phpSessionCookie
13.   if (phpSessionCookie) {
14.     // s'il existe, on affecte le cookie de session PHP au navigateur
15.     document.cookie = phpSessionCookie
16.   }
17.
18.   // où va-t-on ?
19.   const to = context.route.path
20.   if (to === '/fin-session') {
21.     // on nettoie la session
22.     const session = context.app.$session()
23.     session.reset(context)
24.     // on redirige vers la page index
25.     context.redirect({ name: 'index' })
26.   }
27. }
```

Ce sont les lignes 13-17 qui permettent au client [nuxt] de récupérer le cookie de la session PHP du serveur [nuxt].

Le problème ici c'est que lorsqu'on clique sur le bouton [Valider], il n'y a pas de routage du client [nuxt]. Sa fonction de routage n'est alors pas appelée. On règle le problème en dupliquant les lignes 12-17 au début de la méthode d'authentification de la page [index] :

```
1.  // gestionnaires d'évts
2.    methods: {
3.      // ----------- authentification
4.      async login() {
5.        // on récupère le cookie de session PHP dans le store
6.        const phpSessionCookie = this.$store.state.phpSessionCookie
7.        if (phpSessionCookie) {
8.          // s'il existe, on affecte le cookie de session PHP au navigateur
9.          document.cookie = phpSessionCookie
10.       }
11.       try {
12.         // début attente
13.         this.$emit('loading', true)
14.         // on n'est pas encore authentifié
```

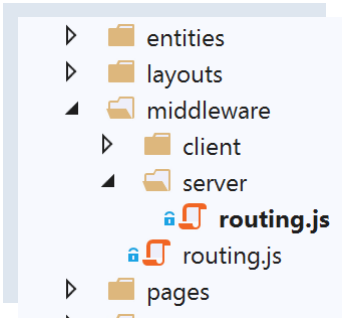
Lignes 5-10, on récupère dans le store le cookie de la session PHP initiée par le serveur [nuxt]. Cette modification faite, on récupère bien la page du calcul de l'impôt, signifiant que l'authentification a fonctionné.

17.9 étape 8

Nous avons une application fonctionnelle et qui travaille dans un esprit [nuxt]. Comme nous l'avons fait pour l'application [nuxt-13], nous allons nous intéresser à la navigation du serveur [nuxt]. Comme il a déjà été dit, l'utilisateur n'est pas censé taper à la main les URL de l'application. Il est censé utiliser les liens qui lui sont présentés et qui sont exécutés par le client [nuxt] qui fonctionne alors en mode SPA. Néanmoins, on va faire en sorte que la navigation du serveur [nuxt] laisse toujours l'application dans un état stable.

De l'étude faite pour [nuxt-13] (cf paragraphe lien), nous savons qu'il faut :

- modifier le script [midleware/routing] ;
- ajouter un script [middleware/server/routing] ;



Le script [middleware/routing] est modifié de la façon suivante :

```
1.  /* eslint-disable no-console */
2.
3.  // on importe les middleware du serveur et du client
4.  import serverRouting from './server/routing'
5.  import clientRouting from './client/routing'
6.
7.  export default function(context) {
8.    // qui exécute ce code ?
9.    console.log('[middleware], process.server', process.server, ', process.client=',
    process.client)
10.   if (process.server) {
11.     // routage serveur
12.     serverRouting(context)
13.   } else {
14.     // routage client
15.     clientRouting(context)
```

```
16.   }
17. }
```

- ligne 4 : on importe le script de routage du serveur [nuxt] ;
- lignes 10-12 : si c'est le serveur [nuxt] qui exécute le code, on utilise sa fonction de routage ;

Le script [middleware/server/routing] est le suivant :

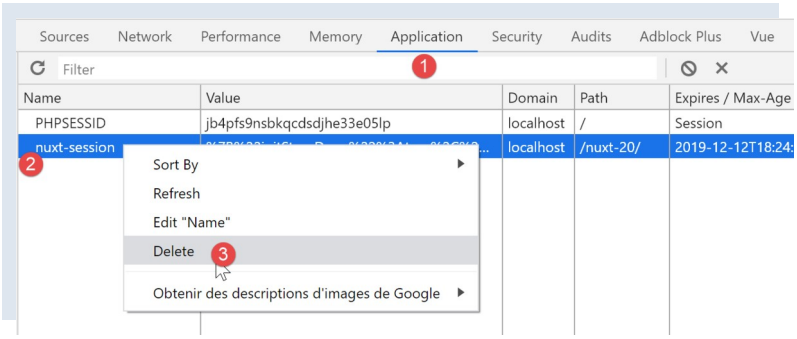
```
1.  /* eslint-disable no-console */
2.  export default function(context) {
3.    // qui exécute ce code ?
4.    console.log('[middleware server], process.server', process.server, ', process.client=',
    process.client)
5.
6.    // on récupère quelques informations ici et là
7.    const store = context.store
8.    // d'où vient-on ?
9.    const from = store.state.from || 'nowhere'
10.   // où va-t-on ?
11.   let to = context.route.name
12.
13.   // cas particulier de /fin-session qui n'a pas d'attribut [name]
14.   if (context.route.path === '/fin-session') {
15.     to = 'fin-session'
16.   }
17.
18.   // éventuelle redirection
19.   let redirection = ''
20.   // gestion du routage terminé
21.   let done = false
22.
23.   // est-on déjà dans une redirection du serveur [nuxt]?
24.   if (store.state.serverRedirection) {
25.     // rien à faire
26.     done = true
27.   }
28.
29.   // s'agit-il d'un rechargement de page ?
30.   if (!done && from === to) {
31.     // rien à faire
32.     done = true
33.   }
34.
35.   // contrôle de la navigation du serveur [nuxt]
36.   // on se calque sur le menu de navigation du client
37.
38.   // on traite d'abord le cas de fin-session
39.   if (!done && store.state.started && store.state.authenticated && to === 'fin-session') {
40.     // on nettoie la session
41.     const session = context.app.$session()
42.     session.reset(context)
43.     // on redirige vers la page index
44.     redirection = 'index'
45.     // travail terminé
46.     done = true
47.   }
48.
49.   // cas où la session PHP n'a pas démarré
50.   if (!done && !store.state.started && to !== 'index') {
51.     // redirection vers [index]
52.     redirection = 'index'
53.     // travail terminé
54.     done = true
55.   }
56.
57.   // cas où l'utilisateur n'est pas authentifié
58.   if (!done && store.state.started && !store.state.authenticated && to !== 'index') {
59.     redirection = 'index'
60.     // travail terminé
61.     done = true
62.   }
63.
64.   // cas où [adminData] n'a pas été obtenu
```

```
65.   if (!done && store.state.started && store.state.authenticated && !
    store.state.métier.taxAdminData && to !== 'index') {
66.     // redirection vers [index]
67.     redirection = 'index'
68.     // travail terminé
69.     done = true
70.   }
71.
72.   // cas où [adminData] a été obtenu
73.   if (
74.     !done &&
75.     store.state.started &&
76.     store.state.authenticated &&
77.     store.state.métier.taxAdminData &&
78.     to !== 'calcul-impot' &&
79.     to !== 'liste-des-simulations'
80.   ) {
81.     // on reste sur la même page
82.     redirection = from
83.     // travail terminé
84.     done = true
85.   }
86.
87.   // on a normalement fait tous les contrôles ---------------------
88.   // redirection ?
89.   if (redirection) {
90.     // on note la redirection dans le store
91.     store.commit('replace', { serverRedirection: true })
92.   } else {
93.     // pas de redirection
94.     store.commit('replace', { serverRedirection: false, from: to })
95.   }
96.   // on sauvegarde le store dans la session [nuxt]
97.   const session = context.app.$session()
98.   session.value.store = store.state
99.   session.save(context)
100.  // on fait l'éventuelle redirection du serveur [nuxt]
101.  if (redirection) {
102.    context.redirect({ name: redirection })
103.  }
104.}
```

- nous reprenons dans ce script les idées déjà développées et utilisées dans le routage du serveur [nuxt] de l'application [nuxt-13] ;
- nous ajoutons deux propriétés au store de l'application :
  - [**from**] : le nom de la dernière page affichée. On sait que le client [nuxt] a cette information mais pas le serveur [nuxt]. On va lui ajouter cette information en stockant dans le store, à chaque routage du serveur [nuxt], le nom de la page qui va être affichée. On fera de même à chaque routage du client [nuxt]. Ainsi au routage suivant du serveur [nuxt], celui-ci trouvera dans le store, le nom de la dernière page affichée par l'application ;
  - [**serverRedirection**] : lorsqu'une destination de routage sera refusée par le serveur [nuxt], celui-ci opèrera une redirection. Il indiquera alors dans le store que la prochaine destination du serveur [nuxt] est une page de redirection. Cette redirection va provoquer une nouvelle exécution du routeur du serveur [nuxt]. Si celui-ci découvre que la destination en cours est issue d'une redirection, il laissera faire ;
- lignes 6-11 : on récupère les informations utiles pour le routage ;
- lignes 13-16 : la cible [/fin-session] n'est pas associée à une page qui s'appellerait [fin-session]. Elle n'a donc pas de nom. On lui en donne un ;
- ligne 19 : la cible d'une éventuelle redirection ;
- ligne 21 : [done=true] lorsque les tests du routage sont terminés ;
- lignes 23-27 : comme il a été dit, si le routage en cours est issu d'une redirection, il n'y a rien à faire. En effet, lors du routage précédent, le routeur a décidé qu'il fallait rediriger le navigateur client. Il n'y a pas lieu de reconsidérer cette décision ;
- lignes 29-33 : s'il s'agit d'un rechargement de page, on laisse faire. Ce n'est pas un axiome valable pour toute application [nuxt] : il faut regarder pour chaque page les effets d'un rechargement. Ici, il se trouve que le rechargement des pages [index, calcul-impot, liste-des-simulations] ne provoque pas d'effets indésirables ;
- lignes 35-85 : le routage du serveur [nuxt] reprend le routage du client [nuxt]. Lorsqu'on est sur une page, le routage du serveur [nuxt] doit refléter le menu de navigation offert par le client [nuxt] lorsqu'on est sur cette page ;

- lignes 38-47 : on traite d'abord le cas de la cible [fin-session] qui ne correspond pas à une page existante. Si les conditions sont réunies (session démarrée, utilisateur authentifié), on nettoie la session et on redirige l'utilisateur vers la page [index] ;
- lignes 49-55 : si la session jSON avec le serveur de calcul de l'impôt n'a pas commencé, alors la seule destination possible est la page [index] ;
- lignes 57-62 : si la session jSON a été démarrée et que l'utilisateur n'est pas authentifié et qu'il n'a pas demandé la page d'authentification, alors on redirige vers la page d'authentification qui est la page [index] ;
- lignes 64-70 : si l'utilisateur est authentifié mais que la donnée [adminData] n'a pas été obtenue, alors on redirige vers la page d'authentification. L'authentification fait deux choses : elle authentifie et si l'authentification a réussi elle demande de plus la donnée [adminData]. Si cette dernière n'a pas été obtenue alors il faut recommencer l'authentification ;
- lignes 72-85 : si la donnée [adminData] a été obtenue, alors les seules cibles possibles sont [calcul-impot] et [liste-des-simulations]. Si ce n'est pas le cas, on refuse le routage ;
- lignes 88-95 : on met à jour le store selon qu'il va y avoir redirection ou non ;
- ligne 94 : il n'y a pas redirection. Aussi le [to] actuel va devenir le [from] du prochain routage ;
- lignes 96-99 : les informations du store sont sauvegardées dans le cookie de la session [nuxt] ;
- lignes 100-103 : si on doit faire une redirection, on la fait ;

Pour faire les tests, il faut prendre soin de partir d'une situation vierge en supprimant le cookie de la session [nuxt] et le cookie de la session PHP avec le serveur de calcul de l'impôt :



Pour tester le routage du serveur [nuxt], sur chaque page essayez toutes les URL possibles [/, /calcul-impot, /liste-des-simulations]. A chaque fois, l'application doit rester dans un état cohérent.

17.10 étape 9

L'étape 9 est celle du déploiement de l'application [nuxt-20]. Celle-ci nécessite un hébergement offrant un environnement [node.js] pour exécuter le serveur [nuxt]. Je ne l'ai pas. Le lecteur pourra suivre les procédures décrites au paragraphe lien, pour déployer l'application [nuxt-20] sur sa machine de développement et la sécuriser avec un protocole HTTPS.

17.11 Conclusion

Le portage de l'application [vuejs-22] vers l'application [nuxt-20] est désormais terminé. Retenons quelques points de ce portage :

- les pages de [vuejs-22] ont été gardées ;
- les opérations asynchrones qui existaient dans les pages de [vuejs-22] ont été migrées dans une fonction [asyncData] ;
- dans [nuxt-20] il a fallu gérer deux entités : le client [nuxt] et le serveur [nuxt]. Cette dernière entité n'existait pas dans [vuejs-22]. Pour garder une cohérence entre les deux entités, nous avons eu besoin d'une session [nuxt] ;
- il nous a fallu gérer le routage du serveur [nuxt] ;

Dans la pratique, il est sans doute préférable de démarrer directement sur une architecture [nuxt] que de faire une architecture [vue.js] portée ensuite dans un environnement [nuxt].