Introduction au framework VUE.JS par l'exemple

Serge Tahé, octobre 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 VUE.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|. **C'est le document présent** ;
4. |Introduction au framework NUXT.JS par l'exemple| ;

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

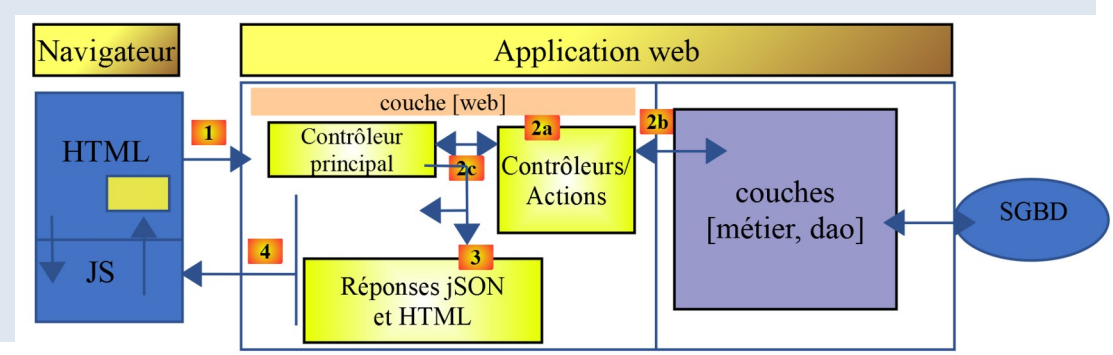
- le document [1] présente le langage PHP 7. Le lecteur seulement intéressé par le langage PHP et pas par le langage Javascript des articles suivants s'arrêtera là ;
- les documents [2-4] visent tous à construire un client Javascript au serveur de calcul de l'impôt développé dans le document [1] ;
- 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] ;

La dernière version du serveur de calcul de l'impôt développée dans le document [1] (cf document |https://tahe.developpez.com/tutoriels-cours/php7|) peut être améliorée de diverses manières :

- la version écrite est centrée sur le serveur. La tendance est désormais (sept 2019) au client / serveur :
  - le serveur fonctionne en service jSON ;
  - une page statique ou non est le point d'entrée de l'application web. Cette page contient du HTML /CSS mais aussi du JavaScript ;
  - les autres pages de l'application web sont obtenues dynamiquement par le JavaScript :
    - la page HTML peut être obtenue par assemblage de fragments statiques ou non, fournis par le même serveur qui a fourni la page d'entrée, ou bien construites par le Javascript du client ;
    - ces différentes pages affichent des données qui sont demandées au service jSON ;

Ainsi le travail est réparti sur le client et le serveur. Le serveur ainsi déchargé peut servir davantage d'utilisateurs.

L'architecture correspondant à ce modèle est le suivant :



JS : JavaScript

Le navigateur est client :

- d'un service de pages ou fragments statiques ou non (non représenté ci-dessus) ;
- d'un service de données jSON ;

Le code JavaScript est donc un client jSON et à ce titre peut être organisé en couches `[UI, métier, dao]` (UI : User Interface) comme l'ont été nos clients jSON écrits en PHP. Au final, le navigateur ne charge qu'une unique page, la page d'accueil. Toutes les autres sont obtenues et construites par le JavaScript. On appelle ce type d'application **SPA** : **S**ingle **P**age **A**pplication ou encore **APU** : **A**pplication à **P**age **U**nique.

Ce type d'application fait également partie des applications dites **AJAX** : **A**synchronous **J**avascript **A**nd **X**M :

- **Asynchronous** : parce que les appels du client Javascript au serveur jSON sont asynchrones ;
- **XML** : parce que XML était la technologie utilisée avant l'avènement du jSON. On a cependant gardé l'acronyme AJAX ;

Nous allons étudier une telle architecture dans ce document. Côté client, nous utiliserons le framework Javascript `[Vue.js]` `[https://vuejs.org/]` pour écrire le client Javascript du serveur jSON PHP que nous avons écrit dans le document [1].

`[Vue.js]` est un framework JavaScript. Nous avons présenté le langage Javascript dans le document [2]. Nous ne ferons pas une étude exhaustive du framework [vue.js]. Nous nous contenterons de présenter les concepts qui seront utilisés ensuite dans l'écriture d'un client `[Vue.js]` pour la version jSON de la version 14 du serveur de calcul de l'impôt (cf |https://tahe.developpez.com/tutoriels-cours/php7|).

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`|.

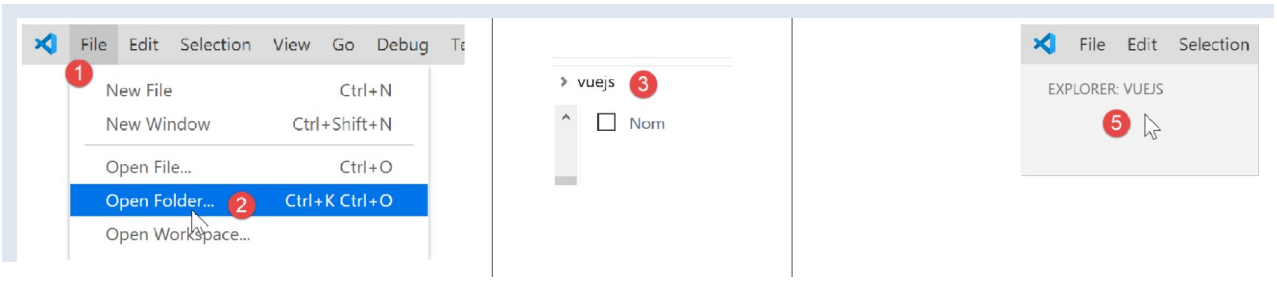
Serge Tahé, octobre 2019

2 Création d'un environnement de travail

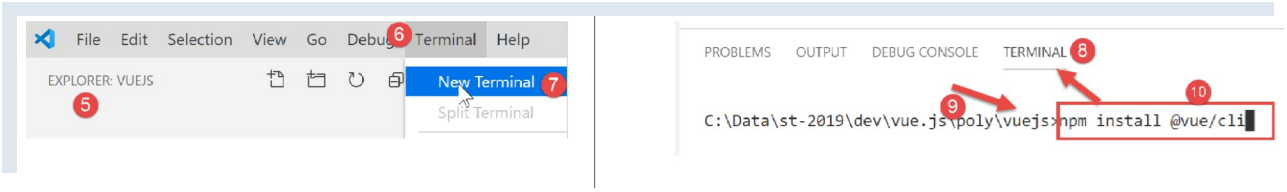
Nous reprenons l'environnement de travail détaillé dans le document [2] :

- Visual Studio Code (VSCode) pour écrire les codes JavaScript ;
- **[node.js]** pour les exécuter ;
- **[npm]** pour télécharger et installer les bibliothèques JavaScript dont nous aurons besoin ;

Nous créons un environnement de travail dans **[VSCode]** :



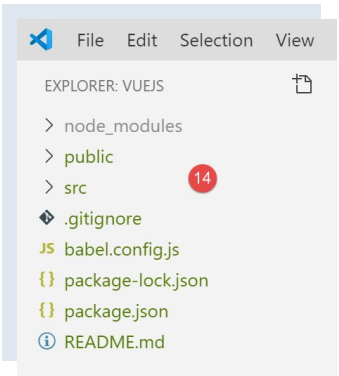
- en **[1-5]**, nous ouvrons un dossier **[vuejs]** vide dans **[VSCode]** ;



- en **[8-10]**, on installe la dépendance **[@vue/cli]** qui va nous permettre d'initialiser un projet **[vue.js]**. Cette dépendance amène un grand nombre de packages (plusieurs centaines) ;

Dans le même terminal, on tape ensuite la commande **[vue create .]** qui demande à créer un projet **[vue.js]** dans le dossier courant (.) :

- en **[13]**, commence une série de questions qui servent à configurer le projet ;



- une fois toutes les questions répondues, de nouveaux packages sont téléchargés et un projet généré dans le dossier courant **[14]**.

Regardons ce qui a été généré. Le fichier **[package.json]** est le suivant :

```
1.  {
2.    "name": "vuejs",
3.    "version": "0.1.0",
4.    "private": true,
5.    "scripts": {
6.      "serve": "vue-cli-service serve vuejs-20/main.js",
7.      "build": "vue-cli-service build vuejs-20/main.js",
8.      "lint": "vue-cli-service lint"
9.    },
10.   "dependencies": {
11.     "axios": "^0.19.0",
12.     "bootstrap": "^4.3.1",
13.     "bootstrap-vue": "^2.0.2",
14.     "core-js": "^2.6.5",
15.     "vue": "^2.6.10",
16.     "vue-router": "^3.1.3",
17.     "vuex": "^3.1.1"
18.   },
19.   "devDependencies": {
20.     "@vue/cli-plugin-babel": "^3.11.0",
21.     "@vue/cli-plugin-eslint": "^3.11.0",
22.     "@vue/cli-service": "^3.11.0",
23.     "babel-eslint": "^10.0.1",
24.     "eslint": "^5.16.0",
25.     "eslint-plugin-vue": "^5.0.0",
26.     "vue-template-compiler": "^2.6.10"
27.   },
28.   "eslintConfig": {
29.     "root": true,
30.     "env": {
31.       "node": true
32.     },
33.     "extends": [
34.       "plugin:vue/essential",
35.       "eslint:recommended"
36.     ],
37.     "rules": {},
38.     "parserOptions": {
39.       "parser": "babel-eslint"
40.     }
41.   },
42.   "postcss": {
43.     "plugins": {
44.       "autoprefixer": {}
45.     }
46.   },
47.   "browserslist": [
48.     "> 1%",
49.     "last 2 versions"
50.   ]
51. }
```

Commentaires

- lignes 14-22 : dans les dépendances nécessaires au développement on voit des références aux deux outils **[eslint, babel]** déjà utilisés dans les deux chapitres précédents. S'y ajoutent des plugins de ces deux outils destinés à leur utilisation au sein de **[vue.js]** ;
- ligne 34 : c'est le package **[babel-eslint]** qui opèrera la transpilation ES6 -> ES5 des codes jS ;
- lignes 5-9 : trois tâches **[npm]** ont été créées :
  - **[build]** : sert à construire la version compilée du projet prête à entrer en production ;
  - **[serve]** : exécute le projet sur un serveur web. C'est avec cet outil que sont faits les tests lors du développement. Comme avec **[webpack-dev-server]**, une modification d'un code source du projet provoque automatiquement la recompilation du projet et son rechargement par le serveur web ;
  - **[lint]** : sert à analyser les codes jS et délivre des rapports. Nous n'utiliserons pas cet outil ici ;

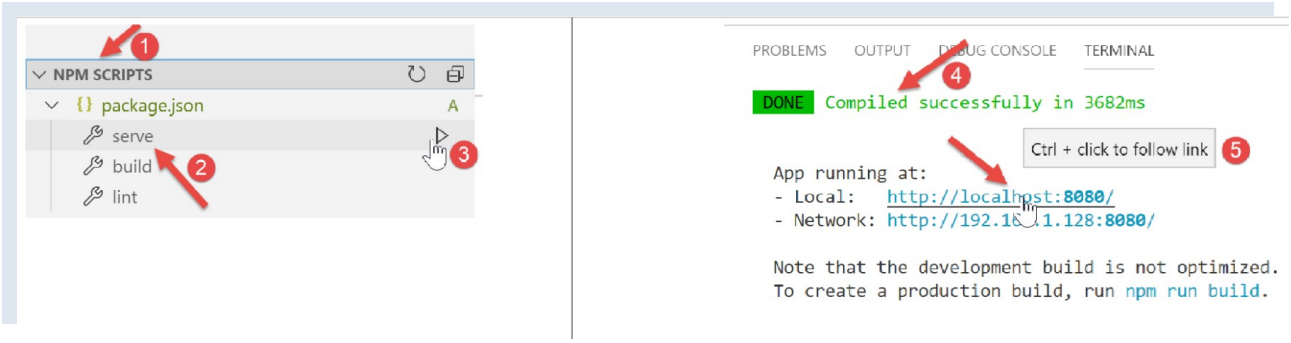
Un fichier **[README.md]** a été généré avec le contenu suivant :

```
1.  # vuejs
2.
3.  ## Project setup
4.  ```
5.  npm install
```

```
6.  ```
7.
8.  ### Compiles and hot-reloads for development
9.  ```
10. npm run serve
11. ```
12.
13. ### Compiles and minifies for production
14. ```
15. npm run build
16. ```
17.
18. ### Run your tests
19. ```
20. npm run test
21. ```
22.
23. ### Lints and fixes files
24. ```
25. npm run lint
26. ```
27.
28. ### Customize configuration
29. See [Configuration Reference](https://cli.vuejs.org/config/).
```

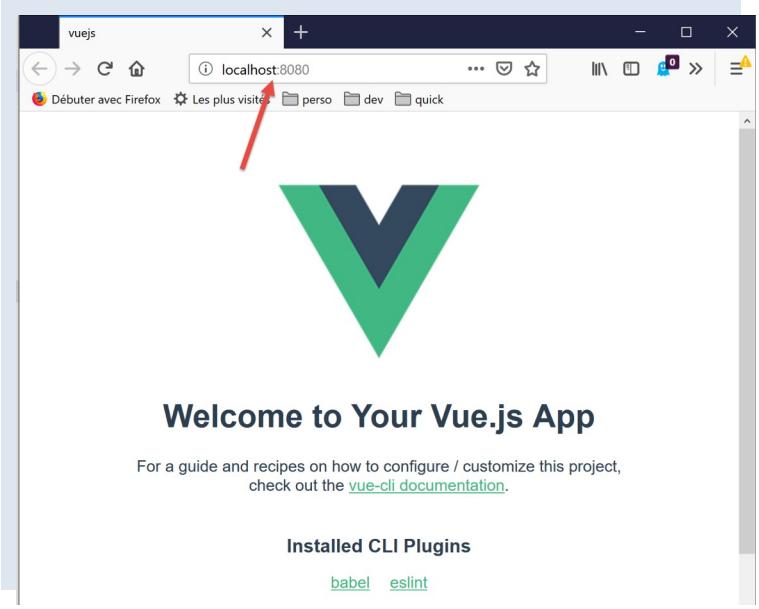
Ce fichier résume les commandes à utiliser pour gérer le projet.

Nous savons que dans **[VSCode]**, les tâches **[npm]** sont proposées à l'exécution :



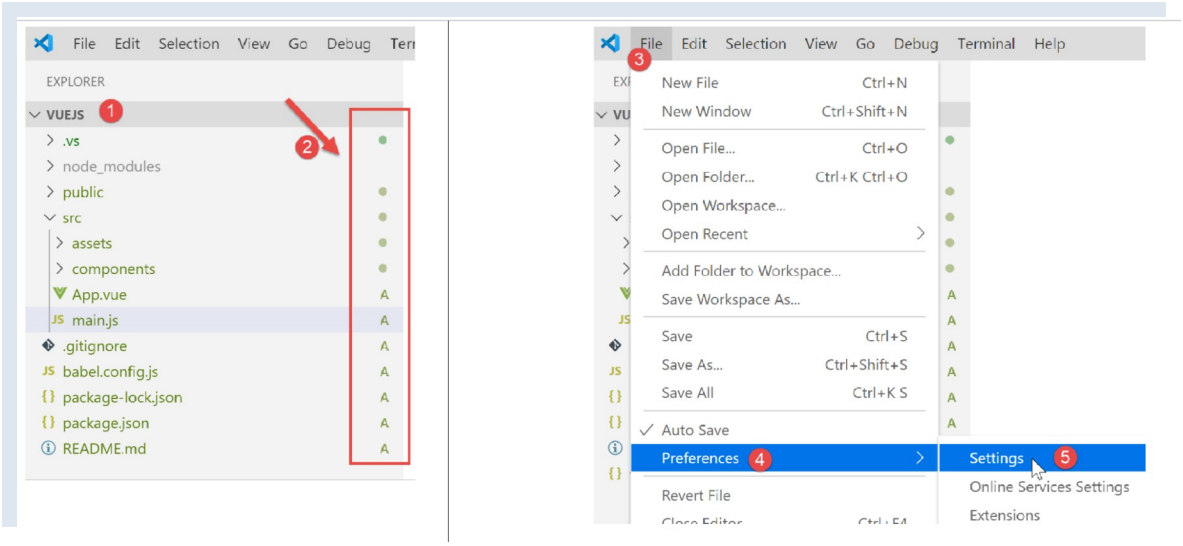
- en **[1-3]**, nous exécutons la commande **[serve]** qui va compiler, puis exécuter le projet **[4-5]** ;

A l'URL **[http://localhost:8080]**, nous obtenons la page suivante :

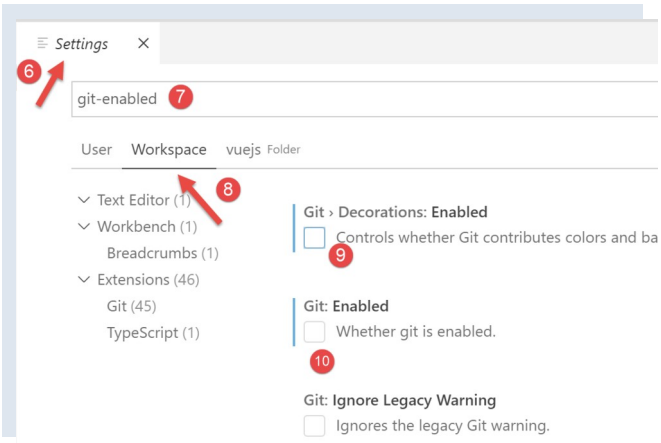


Nous expliquerons un peu plus loin ce qui a amené à cette page.

Continuons à configurer notre environnement de travail :



- en **[2]** ci-dessus, nous voyons des indicateurs **[git]**. **[git]** est un gestionnaire de code source permettant de gérer des versions successives de celui-ci et de les partager entre développeurs. Nous allons désactiver cet outil pour le projet ;
- en **[3-5]**, nous allons dans les propriétés du projet ;



- en **[9-10]**, on désactive l'utilisation de **[git]** dans le projet ;

Nous allons écrire divers tests pour montrer le fonctionnement de **[vue.js]**. Nous ne voulons cependant pas créer à chaque fois un nouveau projet car il faudrait alors à chaque fois générer un dossier **[node_modules]** alors que celui-ci fait plusieurs centaines de méga-octets. Revenons sur les tâches **[npm]** du fichier **[package.json]** :

```
1.    "scripts": {
2.      "serve": "vue-cli-service serve vuejs-00/main.js",
3.      "build": "vue-cli-service build vuejs-00/main.js",
4.      "lint": "vue-cli-service lint"
5.    },
```

- ligne 2 : la commande **[serve]** utilise par défaut :
  - le fichier **[public/index.html]** ;
  - associé au fichier **[src/main.js]** ;

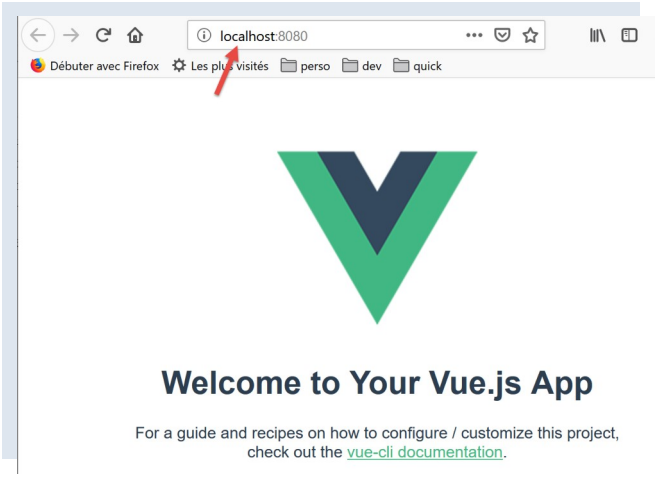
Ligne 2, il est possible de préciser à la commande **[serve]**, le point d'entrée du projet, par exemple :

```
"serve": "vue-cli-service serve vuejs-00/main.js",
```

Essayons :

- en **[1]**, le dossier **[src]** a été renommé en **[vuejs-00]** ;
- en **[2-3]**, on a modifié la commande **[serve]** ;
- en **[4-6]**, on exécute le projet ;

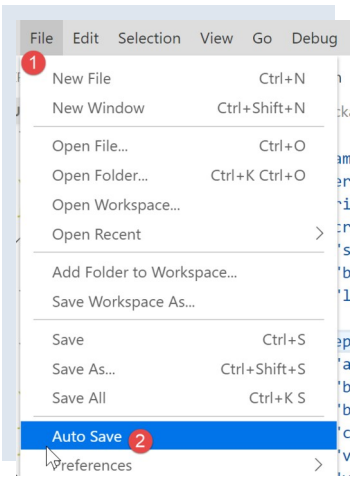
On obtient le même résultat que précédemment :



Pour nos tests, nous procéderons donc ainsi :

- écriture de code dans un dossier **[vuejs-xx]** du projet ;
- test de ce projet avec la commande **[vue-cli-service serve vuejs-xx/main.js]** dans le fichier **[package.json]** ;

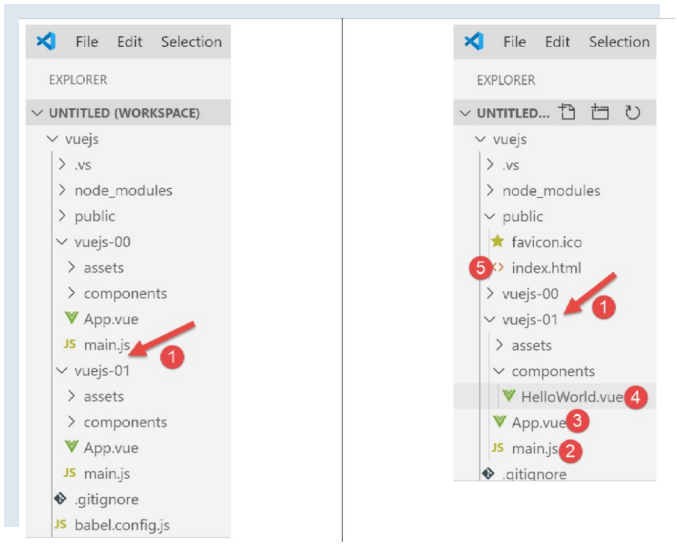
Lorsque le serveur de développement est lancé, toute modification d'un des fichiers du projet provoque une recompilation. Pour cette raison, nous inhibons le mode **[Auto Save]** de **[VSCode]**. En effet, nous ne voulons pas de recompilation dès qu'on tape des caractères dans un des fichiers du projet. Nous ne voulons de recompilation qu'à certains moments :



- en **[2]**, l'option **[Auto Save]** ne doit pas être cochée ;

3 projet [vuejs-01] : les bases

Pour expliquer le code exécuté dans **[vuejs-00]** nous allons le simplifier dans **[vuejs-01]**. Nous dupliquons le dossier **[vuejs-00]** dans **[vuejs-01]** :



Le projet **[vuejs-01]** comprend essentiellement quatre fichiers :

- **[main.js] [2]** est le point d'entrée du projet ;
- **[App.vue, HelloWorld.vue] [3-4]** sont des composants **[Vue.js]**, comprenant de façon facultative les éléments suivants :
  - **[<template>...</template>]** : du code HTML ;
  - **[<script>...</script>]** : le code Javascript associé au code HTML ;
  - **[<style>...</style>]** : le style CSS associé au code HTML ;
- **[public/index.html] [5]** : le document HTML visualisé par la commande **[npm run serve]** ;

Le fichier **[public/index.html]** affiché à l'exécution du projet est celui-ci :

```
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="<%= BASE_URL %>favicon.ico">
8.      <title>vuejs</title>
9.    </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.   </body>
17. </html>
```

Ce fichier HTML n'affiche donc rien **statiquement**. Il n'y a ici pas de code HTML. L'affichage est **dynamique** : le code jS du projet va générer du HTML qui va remplacer entièrement la balise **[ ]** de la ligne 14. Le code HTML généré par le code jS du projet et inséré à la place de la balise **[ ]** de la ligne 14 provient des balises **[template]** des composants **[vue.js]**, les fichiers ayant le suffixe **[.vue]**.

Le code HTML est inséré dynamiquement ligne 14 par le script **[vuejs-01/main.js]** suivant :

```
1.  // imports
2.  import Vue from 'vue'
3.  import App from './App.vue'
4.
5.  // configuration
6.  Vue.config.productionTip = false
7.
```

```
8.  // instanciation projet [App]
9.  new Vue({
10.   render: h => h(App),
11. }).$mount('#app')
```

Commentaires

- ligne 2 : l'objet **[Vue]** est fourni par le framework **[vue.js]** ;
- ligne 3 : l'objet **[App]** est fourni par le fichier **[vuejs-01/App.vue]** ;
- ligne 6 : configuration de l'objet **[Vue]** ;
- lignes 9-11 : ce sont les lignes qui :
  - génèrent le code HTML de l'application. Ligne 10, c'est le fichier **[App.vue]** qui le génère ;
  - chargent le code HTML généré ligne 10 dans la section **[ ]** du fichier **[public/index.html]** ;

Tout projet **[Vue.js]** peut conserver le fichier **[index.html]** tel quel.

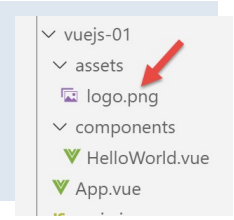
Le fichier **[App.vue]** du projet initial **[vuejs-00]** est simplifié de la façon suivante dans le projet **[vuejs-01]** :

```
1.  <template>
2.    <div id="myApp">
3.      <img alt="Vue logo" src="./assets/logo.png" />
4.      <HelloWorld msg="Notre première application Vue.js" />
5.    </div>
6.  </template>
7.
8.  <script>
9.  import HelloWorld from "./components/HelloWorld.vue";
10.
11. export default {
12.   name: "app",
13.   components: {
14.     HelloWorld
15.   }
16. };
17. </script>
```

Commentaires

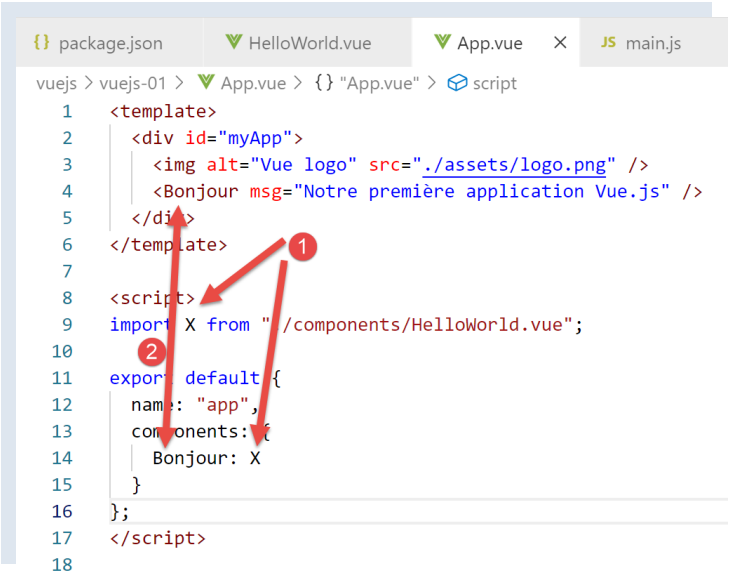
- un fragment **[.vue]** comprend au plus trois sections :
  - **[<template>...</template>]** : du code HTML ;
  - **[<script>...</script>]** : le code Javascript associé au code HTML ;
  - **[<style>...</style>]** : le style CSS associé au code HTML ;

  Ici, nous n'avons pas de section **[style]**.
- lignes 1-6 : le code HTML du fragment (page, composant, vue, ...) ;
- lignes 2-5 : la section **[template]** ne peut contenir qu'un élément. On met en général une section **[div]** qui englobe tout le HTML du fragment. On peut mettre également une balise <template> ;
- ligne 3 : une image ;



- ligne 4 : un composant nommé **[HelloWorld]**. Le principe de **[Vue.js]** est de construire des pages web à l'aide de fragments définis dans des fichiers **[.vue]** comme ici **[App.vue]**. Ce composant est défini par le fichier **[HelloWorld.vue]** défini ligne 9 du script jS associé ;
- ligne 4 : un composant peut accepter des paramètres. Le paramètre est ici l'attribut **[msg]** ;
- lignes 8-17 : le script jS du fragment (ou composant) ;
- ligne 9 : pour pouvoir utiliser le composant **[HelloWorld]** dans le composant **[App]**, il faut importer sa définition dans la partie **[script]** ;
- lignes 11-16 : le script définit un objet et l'exporte afin de le rendre disponible à l'extérieur ;
- ligne 12 : l'attribut **[name]** : définit le nom du composant exporté ;
- lignes 13-15 : l'attribut **[components]** liste les composants utilisés par le composant **[App]**. Ils sont exportés avec lui ;

Ligne 9, il n'y a pas obligation que le composant **[HelloWorld]** porte le même nom que le fichier qui le définit. On pourrait l'importer en tant que **[X]** et l'exporter en tant que composant **[Bonjour]** :

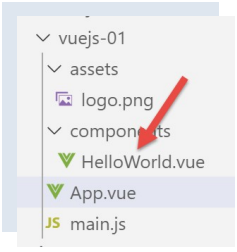


- ligne 14 : le composant **[X]** est exporté sous le nom **[Bonjour]**. Il est alors utilisé sous ce nom, ligne 4 ;

La première version est la version la plus courante, aussi définirons-nous nos composants de cette façon ;

```
1.  <template>
2.    <div id="myApp">
3.      <img alt="Vue logo" src="./assets/logo.png" />
4.      <HelloWorld msg="Notre première application Vue.js" />
5.    </div>
6.  </template>
7.
8.  <script>
9.  import HelloWorld from "./components/HelloWorld.vue";
10.
11. export default {
12.   name: "app",
13.   components: {
14.     HelloWorld
15.   }
16. };
17. </script>
```

La ligne 14 est un raccourci pour le code **[HelloWorld : HelloWorld]** : le composant **[HelloWorld]** (à droite, importé ligne 9) est exporté sous le nom **[HelloWorld]** (à gauche).



Nous simplifions le composant **[HelloWorld.vue]** de la façon suivante :

```
1.  <template>
2.    <div>
3.      <h1>{{ msg }}</h1>
4.    </div>
5.  </template>
6.
7.  <script>
```

```
8.  export default {
9.    name: "HelloWorld",
10.   props: {
11.     msg: String
12.   }
13. };
14. </script>
```

Commentaires

- le composant **[HelloWorld]** a la même structure de fichier que le composant principal **[App]** ;
- ligne 3 : on a ici une évaluation d'expression Javascript, ici l'expression **[msg]** ;
- lignes 10-12 : définissent les propriétés du composant, plus exactement ses paramètres. Lorsque le composant **[App]** a instancié un composant **[HelloWorld]**, il l'a fait avec la syntaxe suivante :

```
<HelloWorld msg="Notre première application Vue.js" />
```

Le composant **[HelloWorld]** est instancié en donnant une valeur au paramètre (attribut) **[msg]**. Si on suit le **[template]** du composant **[HelloWorld]**, celui-ci devient :

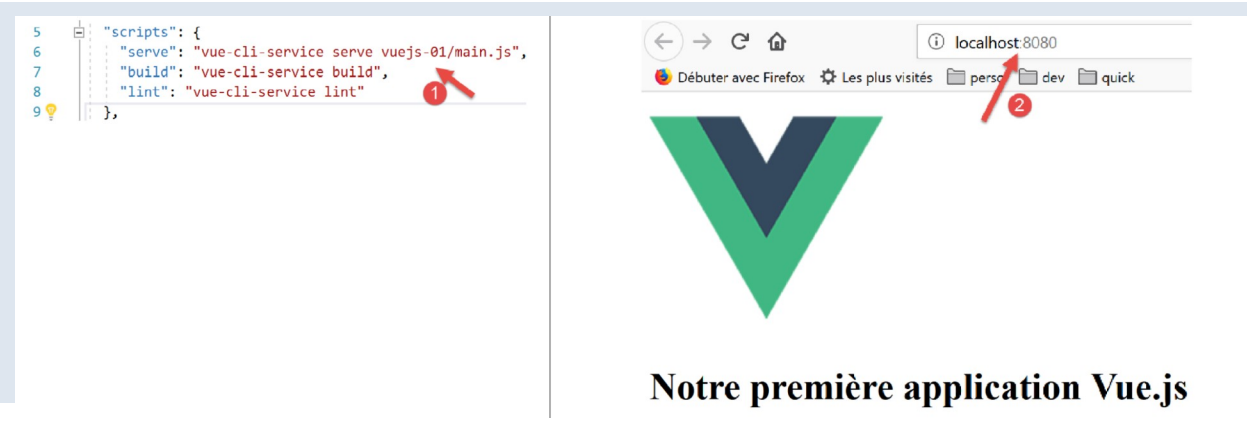
```
<div>
    <h1>Notre première application Vue.js</h1>
</div>
```

- lignes 7-14 : les propriétés du composant définies sous la forme d'un objet qui est exporté ;
  - ligne 9 : le composant est exporté sous le nom **[HelloWorld]** ;
  - lignes 10-12 : ses paramètres sont définis par la propriété **[props]** ;

Au final, si on rassemble les templates des deux composants **[App, HelloWorld]** utilisés, le fichier **[index.html]** affiché sera le suivant :

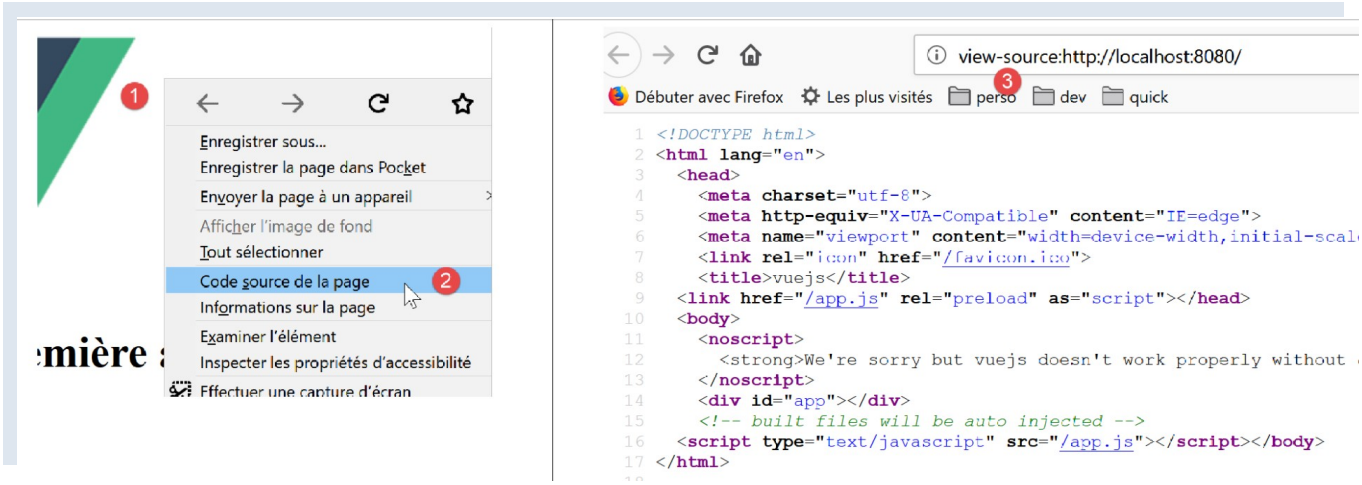
```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.  <head>
4.    <meta charset="utf-8">
5.    <meta http-equiv="X-UA-Compatible" content="IE=edge">
6.    <meta name="viewport" content="width=device-width,initial-scale=1.0">
7.    <link rel="icon" href="<%= BASE_URL %>favicon.ico">
8.    <title>vuejs</title>
9.  </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="myApp">
15.     <img alt="Vue logo" src="./assets/logo.png" />
16.     <div>
17.       <h1>Notre première application Vue.js</h1>
18.     </div>
19.   </div></body>
20. </html>
```

Nous lançons l'application en modifiant la commande **[serve] [1]** du fichier **[package.json]** :



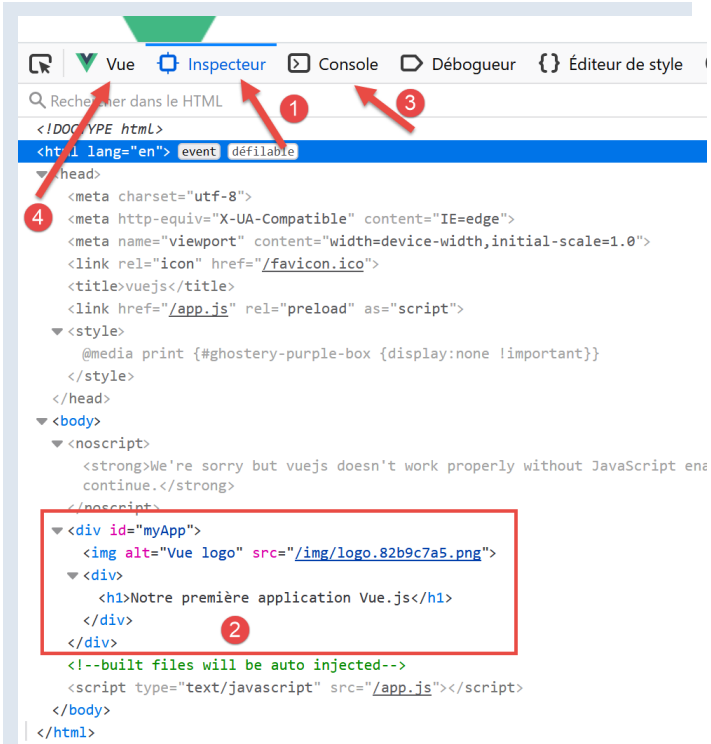
La page affichée est alors **[2]**.

Maintenant regardons le code de cette page :



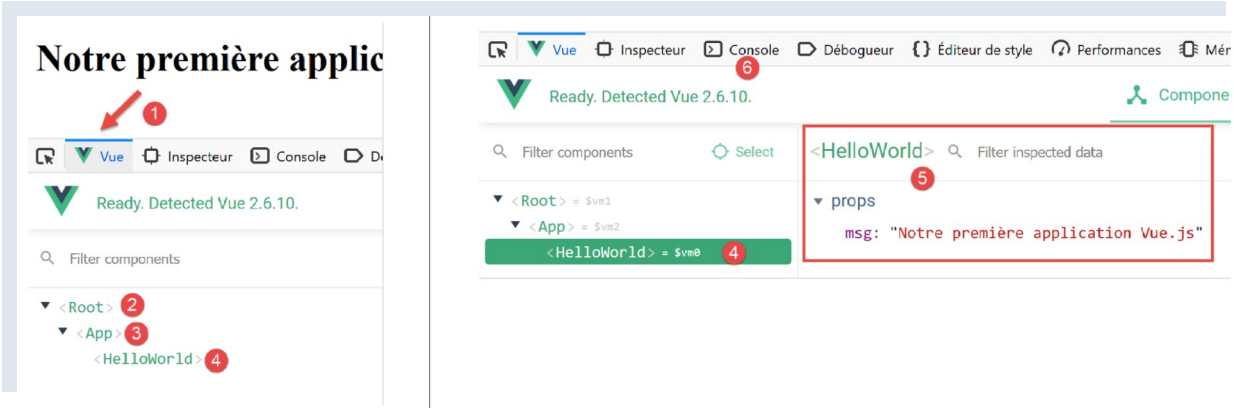
- en **[1]**, faire **[clic droit]** ;
- en **[2]**, le code source de la page. On voit que c'est le code du fichier initial **[index.html]** et ce n'est pas ça qui a été affiché. C'est bien la page **[index.html]** qui a été chargée initialement. Ensuite, dynamiquement, du code Javascript a modifié cette page, mais cela ne nous est pas montré ;

Lorsque les pages sont générées dynamiquement par du Javascript, l'option **[2]** ne sert à rien. Il faut aller dans les outils du navigateur (F12 sur Firefox) pour voir le code de la page actuellement affichée :



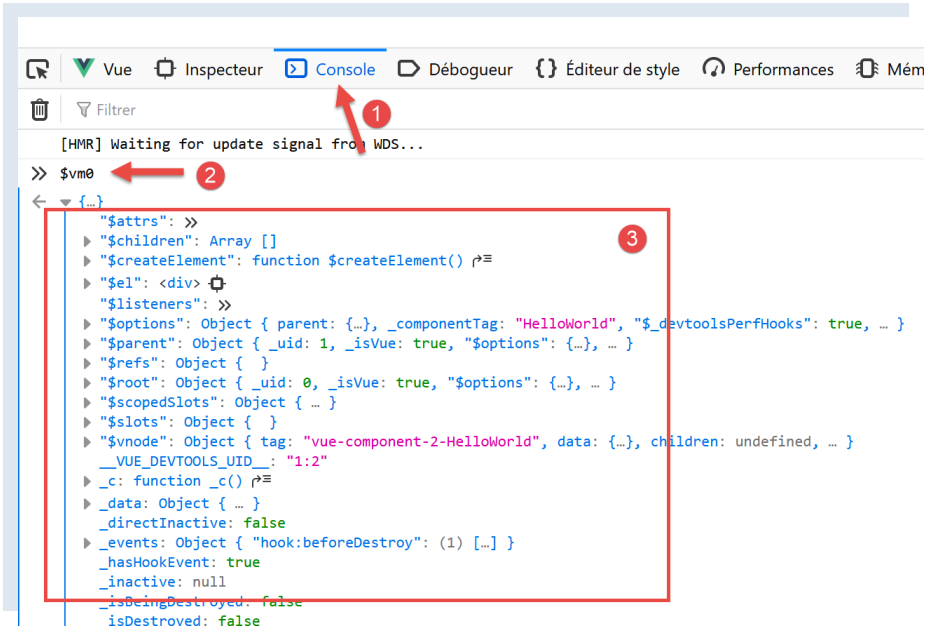
- en **[1]**, l'inspecteur du DOM (Document Object Model) du document affiché ;
- en **[2]**, ce que contient réellement ce DOM ;
- **[3-4]**, des outils que nous utiliserons pour afficher les objets Javascript utilisés par le framework **[Vue.js]** ;
- **[4]** est une extension (ici Firefox) pour déboguer des applications **[Vue.js]** :
  - pour Firefox : **[https://addons.mozilla.org/fr/firefox/addon/vue-js-devtools/]**;
  - pour Chrome : **[https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd]**;

Examinons l'onglet **[Vue] [4]** :



La vue **[1-4]** nous montre la structure **[Vue.js]** du document : la racine du document **[2]** (index.html) comprend le composant **[App]** (3) qui lui même comprend le composant **[HelloWorld]** (4). Cliquer sur **[4]** fait apparaître les propriétés du composant **[HelloWorld]** **[5]**.

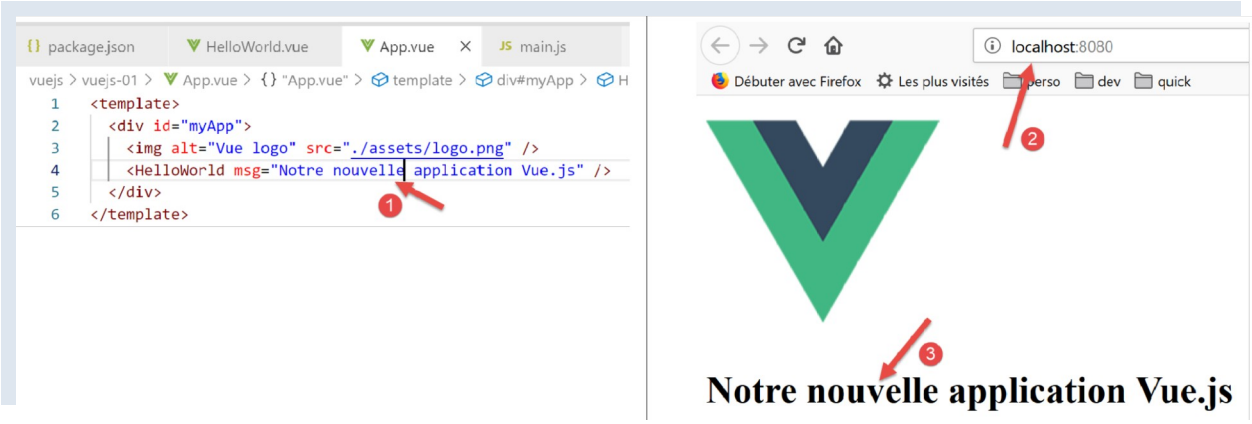
On voit en **[4]** (à droite), l'indicateur **[$vm0]**. C'est le nom de la variable qu'on peut utiliser dans la console JavaScript **[6]** pour désigner l'objet **[HelloWorld]**. Faisons-le :



- en **[2]**, on fait évaluer l'expression **[$vm0]**, ce qui a pour effet d'afficher sa structure. Normalement nous n'aurons pas à utiliser directement cette structure ;

Terminons en montrant la capacité de **[hot reload]** de la commande **[serve]** utilisée pour exécuter le projet :

- dans **[App.vue]**, modifiez le message affiché par **[HelloWorld]** :



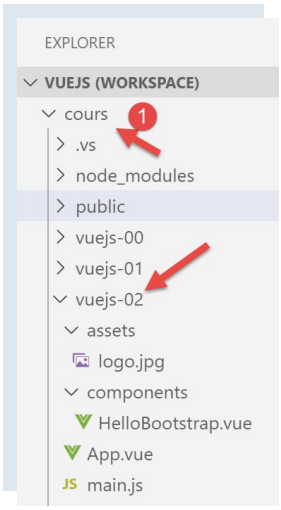
- en **[1]**, on modifie le message affiché ;
- en **[2-3]**, la page est automatiquement mise en jour sans intervention de notre part ;

Nous allons maintenant créer divers projets **[vuejs-xx]** pour illustrer les points importants de **[Vue.js]**. Par 'importants', il faut entendre 'que nous allons utiliser dans le client **[vue.js]** du serveur de calcul de l'impôt'. D'autres points 'importants' seront passés sous silence s'ils ne sont pas utilisés dans le client. Ce n'est donc pas une présentation exhaustive de **[vue.js]** qui sera faite.

4 projet [vuejs-02] : utilisation du framework CSS Bootstrap

Le projet **[vuejs-02]** présente l'utilisation de Bootstrap dans un projet **[vue.js]**. C'est le framework CSS qui sera utilisé dans tous nos projets. Nous utiliserons une variation de Bootstrap appelée **[BootstrapVue]** **[https://bootstrap-vue.js.org/]**.

L'arborescence du projet sera la suivante :



Note : ci-dessus le dossier **[vuejs]** a été renommée **[cours]** **[1]** dans la suite du document.

4.1 Installation du framework [BootstrapVue]

[BootstrapVue] est un framework qu'on ajoute au projet avec l'outil **[npm]** :

- en **[1]**, c'est donc deux frameworks qu'on installe : **[Bootstrap]** et sa variante **[BootstrapVue]** ;
- en **[2]**, les deux dépendances apparaissent dans le fichier **[package.json]** ;

4.2 Le script [main.js]

Le script principal **[main.js]** est 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. // configuration
14. Vue.config.productionTip = false
15.
16. // instanciation projet [App]
17. new Vue({
18.   render: h => h(App),
19. }).$mount('#app')
```

- ligne 2 : import du framework **[Vue]** ;
- ligne 3 : import de la vue principale ;
- ligne 6 : import du framework **[BootstrapVue]** ;
- ligne 7 : ce framework est conçu comme un plugin du framework **[Vue]**. La ligne 7 inclut ce plugin dans le framework **[Vue]** ;
- lignes 10-11 : import des fichiers CSS des frameworks **[Bootstrap]** et **[BootstrapVue]** ;
- les lignes 5-11 sont donc entièrement consacrées à l'utilisation de **[BootstrapVue]**. Le reste du code est identique à ce qu'on avait vu au paragraphe précédent ;

4.3 Le composant [App.vue]

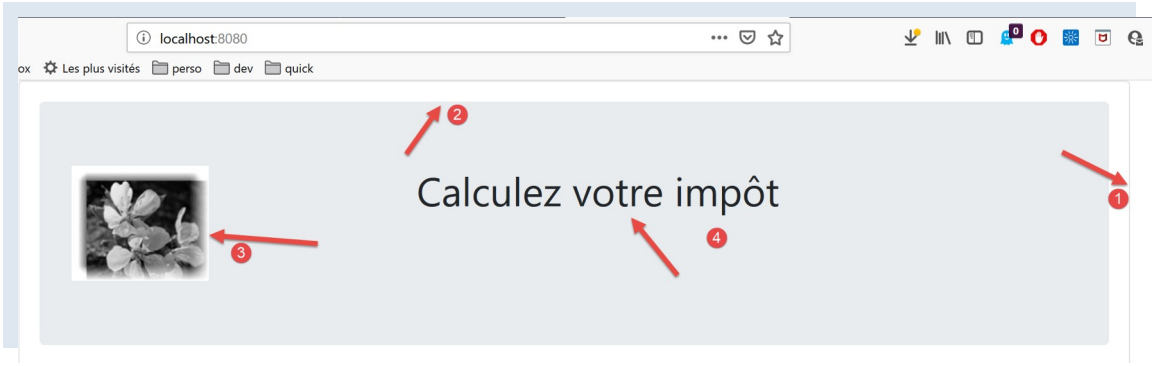
```
1.  <template>
2.    <b-container>
3.      <b-card>
4.        <!-- Bootstrap Jumbotron -->
5.        <b-jumbotron>
6.          <!-- ligne -->
7.          <b-row>
8.            <!-- colonne de largeur 4 -->
9.            <b-col cols="4">
10.             <img src="./assets/logo.jpg" alt="Cerisier en fleurs" />
11.           </b-col>
12.           <!-- colonne de largeur 8 -->
13.           <b-col cols="8">
14.             <h1>Calculez votre impôt</h1>
15.           </b-col>
16.         </b-row>
17.       </b-jumbotron>
18.       <HelloBootstrap msg="Hello Bootstrap !" />
19.     </b-card>
20.   </b-container>
21. </template>
22.
23.
24. <script>
25.   import HelloBootstrap from "./components/HelloBootstrap.vue";
26.
27.   export default {
28.     name: "app",
29.     components: {
30.       HelloBootstrap
31.     }
32.   };
33. </script>
```

Commentaires

- lignes 1-21 : toutes les balises <b-xx> sont des balises du framework **[BootstrapVue]** ;
- lignes 2, 20 : la balise <b-container> définit un conteneur Bootstrap. A l'intérieur de ce conteneur, on va pouvoir définir des lignes avec la balise <b-row> et des colonnes avec la balise <b-col> ;
- lignes 3, 19 : la balise <b-card> définit une 'carte' Bootstrap. Cela se matérialise visuellement par un rectangle avec une bordure ;
- lignes 5, 17 : la balise <b-jumbotron> permet de mettre en avant une partie de la page, ici une image et un texte. On l'utilisera dans nos divers projets comme identification visuelle du projet ;
- ligne 7 : la balise <b-row> définit une ligne ;
- lignes 9-11 : la balise <b-col> définit une colonne de la ligne précédente. Bootstrap attribue 12 colonnes à chaque ligne. L'attribut **[cols='4']** indique que la colonne <b-col> va occuper 4 de ces 12 colonnes ;
- ligne 10 : une image
- lignes 13-15 : une colonne qui va occuper 8 des 12 colonnes de la ligne. on y met un texte ;
- ligne 18 : utilisation d'un composant appelé **[HelloBootstrap]** avec une propriété nommée **[msg]** ;

- lignes 24-33 : la partie <script> du composant ;
- lignes 29-31 : le composant **[HelloBootstrap]** utilisée ligne 18 est exporté. Pour être connu, il doit être importé ligne 25 ;

Le résultat est le suivant :



- en **[1]**, la balise <b-card> ;
- en **[2]**, la balise <jumbotron> ;
- en **[3]**, l'image sur 4 colonnes ;
- en **[4]**, le texte sur 8 colonnes ;

4.4 Le composant [HelloBootstrap]

[HelloBootstrap] est le composant suivant :

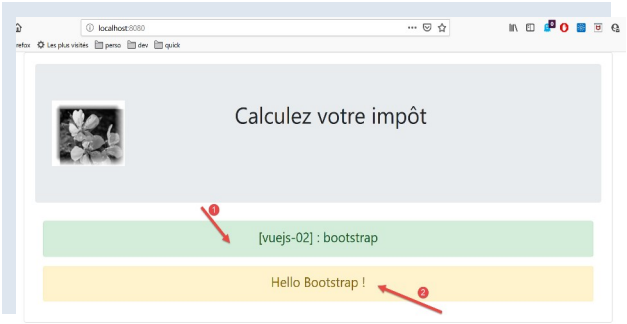
```
1.  <template>
2.    <div>
3.      <!-- message sur fond vert -->
4.      <b-col cols="12">
5.        <b-alert show variant="success" align="center">
6.          <h4>[vuejs-02] : bootstrap</h4>
7.        </b-alert>
8.      </b-col>
9.      <!-- message sur fond jaune -->
10.     <b-col cols="12">
11.       <b-alert show variant="warning" align="center">
12.         <h4>{{msg}}</h4>
13.       </b-alert>
14.     </b-col>
15.   </div>
16. </template>
17.
18. <script>
19.   export default {
20.     name: "HelloBootstrap",
21.     props: {
22.       msg: String
23.     }
24.   };
25. </script>
```

Commentaires

- ligne 3 : la balise <b-alert show> affiche un rectangle de couleur dans lequel on met en général un texte (ligne 6). L'attribut **[variant]** permet de sélectionner un type d'alerte. Chaque type d'alerte a une couleur de fond différente. La couleur de la variante **[success]** est le vert. L'attribut **[align]** permet d'aligner le texte de l'alerte (gauche, droite, centré). On notera que l'attribut **[show]** est obligatoire pour afficher l'alerte. Sans cet attribut, l'alerte n'est pas visible ;
- les valeurs possibles de **[variant]** :
  - **[primary]** : bleu ;
  - **[secondary]** : gris ;
  - **[success]**:vert ;
  - **[danger]** : rouge léger ;
  - **[warning]** : jaune ;
  - **[info]** : turquoise ;
  - **[light]** : pas de couleur de fond ;
  - **[dark]** : gris un peu plus foncé que **[secondary]** ;

- ligne 12 : `[msg]` est un paramètre du composant `[HelloBootstrap]` (lignes 21-23) ;

Le rendu visuel est le suivant :



- `[1]` : balise <b-alert show variant='success'> ;
- `[2]` : balise <b-alert show variant='warning'> ;

4.5 Exécution du projet

Pour exécuter le projet, on modifie d'abord le fichier `[package.json]` :

- en `[3]`, on modifie le script exécuté par la commande `[serve]` `[2]` du fichier package.json `[1]` ;
- en `[4]`, on exécute le projet ;

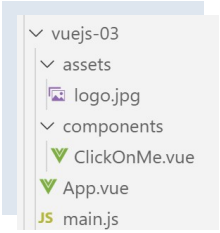
Note : dans tout ce qui suit on utilisera les balises du framework BootstrapVue, des balises de la forme <b-qqchose>. Ce n'est pas obligatoire. On peut utiliser les balises originelles du framework Bootstrap. Elles sont fonctionnelles dans les templates de `[Vue.js]`. Aussi le développeur habitué aux balises Bootstrap peut continuer à les utiliser.

5 projet [vuejs-03] : gestion des événements

Le projet **[vuejs-03]** introduit deux concepts :

- la gestion d'un événement **[clic]** sur un bouton ;
- la directive **[v-if]** qui permet d'afficher un bloc HTML de façon conditionnelle ;

L'arborescence du projet est la suivante :



5.1 Le script principal [main.js]

Le script **[main.js]** reste inchangé :

```
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. // configuration
14. Vue.config.productionTip = false
15.
16. // instanciation projet [App]
17. new Vue({
18.   render: h => h(App),
19. }).$mount('#app')
```

5.2 Le composant principal [App.vue]

Le composant principal **[App.vue]** utilise le composant **[ClickOnMe]** au lieu du composant **[HelloBootstrap]** :

```
1.  <template>
2.    <b-container>
3.      <b-card>
4.      <!-- Bootstrap Jumbotron -->
5.      <b-jumbotron>
6.        <!-- ligne -->
7.        <b-row>
8.          <!-- colonne de largeur 4 -->
9.          <b-col cols="4">
10.           <img src="./assets/logo.jpg" alt="Cerisier en fleurs" />
11.         </b-col>
12.         <!-- colonne de largeur 8 -->
13.         <b-col cols="8">
14.           <h1>Calculez votre impôt</h1>
15.         </b-col>
16.       </b-row>
17.     </b-jumbotron>
18.     <!-- composant -->
19.     <ClickOnMe msg="Information..." />
20.     </b-card>
21.   </b-container>
22. </template>
23.
24.
```

```
25. <script>
26. import ClickOnMe from "./components/ClickOnMe.vue";
27.
28. export default {
29.   name: "app",
30.   components: {
31.     ClickOnMe
32.   }
33. };
34. </script>
```

5.3 Le composant `[ClickOnMe]`

Le composant **`[ClickOnMe]`** introduit les nouveaux concepts :

```
1.  <template>
2.    <div>
3.      <!-- message sur fond vert -->
4.      <b-alert show variant="success" align="center">
5.        <h4>[vuejs-03] : événement @click, directive v-if, méthodes</h4>
6.      </b-alert>
7.      <!-- message sur fond jaune -->
8.      <b-alert show variant="warning" align="center" v-if="show">
9.        <h4>{{msg}}</h4>
10.     </b-alert>
11.     <!-- bouton bleu -->
12.     <b-button variant="primary" @click="changer">{{buttonTitle}}</b-button>
13.   </div>
14. </template>
15.
16. <script>
17.   export default {
18.     name: "ClickOnMe",
19.     // paramètres du composant
20.     props: {
21.       msg: String
22.     },
23.     // attributs du composant
24.     data() {
25.       return {
26.         // titre du bouton
27.         buttonTitle: "Cacher",
28.         // contrôle l'affichage du message
29.         show: true
30.       };
31.     },
32.     // méthodes
33.     methods: {
34.       // montre / cache le message
35.       changer() {
36.         if (this.show) {
37.           // on cache le message
38.           this.show = false;
39.           this.buttonTitle = "Montrer";
40.         } else {
41.           // on montre le message
42.           this.show = true;
43.           this.buttonTitle = "Cacher";
44.         }
45.       }
46.     }
47.   };
48. </script>
```

Commentaires

- lignes 4-6 : une alerte verte Bootstrap. Le nombre de colonnes occupées n'est pas indiqué. Ce sont alors les 12 colonnes de Bootstrap qui sont utilisées ;
- lignes 8-10 : une alerte jaune Bootstrap :
  - ligne 8 : la directive **`[v-if]`** de **`[Vue.js]`** contrôle la visibilité d'un bloc HTML. L'alerte est ici contrôlée par un booléen **`[show]`** (ligne 29). Si **`[show==true]`** alors l'alerte sera visible sinon elle ne le sera pas ;
  - ligne 9 : l'alerte affiche un message **`[msg]`** qui est une propriété (lignes 20-22) du composant ;
- ligne 12 : un bouton de couleur bleue sur laquelle on clique pour cacher / montrer l'alerte **`[warning]`** ;
- lignes 16-48 : le code jS du composant. Ce code règle le fonctionnement dynamique du composant :

- lignes 20-22 : les propriétés du composant ;
- lignes 24-31 : les attributs du composant ;

Quelle est la différence entre `[propriétés]` et `[attributs]` d'un composant, entre les champs `[props]` et `[data]` de l'objet exporté par le composant aux lignes 17-47 ?

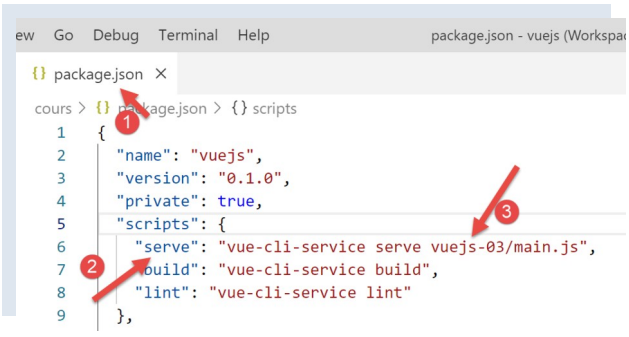
- comme nous l'avons déjà vu, les propriétés `[props]` d'un composant sont des paramètres du composant. Leurs valeurs sont fixées de l'**extérieur** du composant. Un composant A utilisant un composant B ayant les propriétés `[prop1, prop2, ..., propn]` l'utilisera de la façon suivante : **<B :prop1='val1' :prop2='val2' ...>** ;
- l'objet rendu par la fonction `[data]` des lignes 24-31 représente l'état du composant ou **attributs** du composant. Cet état est manipulé par les méthodes du composant (lignes 33-46). Le <template> des lignes 1-14 utilise aussi bien des éléments `[propriétés]` que `[attributs]` :
  - les valeurs des propriétés sont fixées par un composant parent ;
  - les valeurs des attributs sont fixées initialement par la fonction `[data]` puis peuvent être modifiées par les méthodes ;
  - dans les deux cas, le rendu visuel **réagit** immédiatement aux changements d'une propriété (composant parent) ou d'un attribut (méthode du composant). On parle alors d'interface **réactive** ;

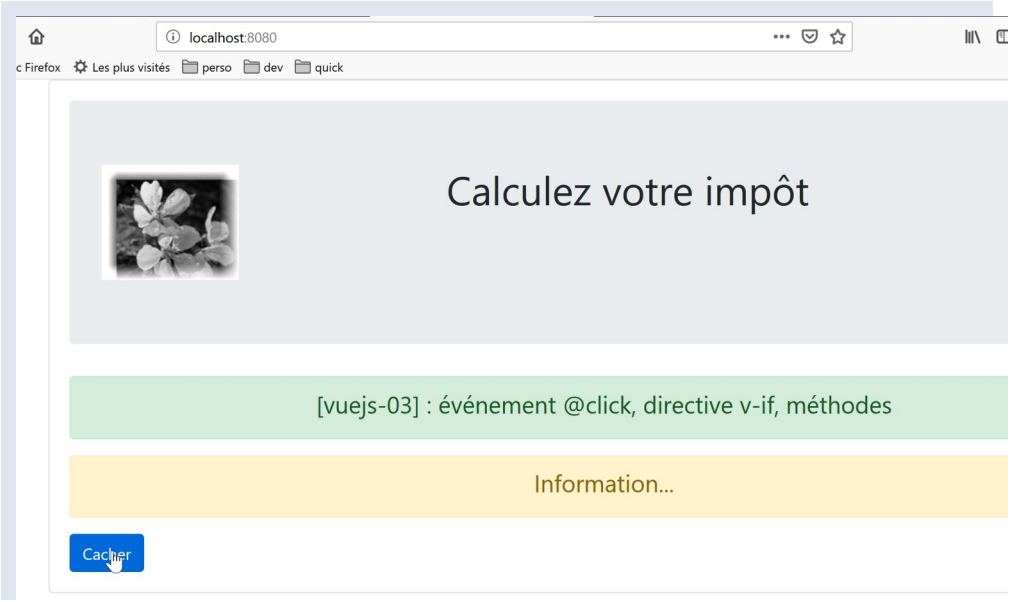
  Dans le `[template]` d'un composant, rien ne diffère une propriété `[prop]` d'un attribut `[data]`. Pour savoir si une donnée dynamique du `[template]` doit être mise dans l'attribut `[props]` ou dans l'objet rendu par la fonction `[data]`, il faut simplement se demander qui fixe la valeur de cette donnée :
  - si la réponse est le composant parent, alors on mettra la donnée dans l'attribut `[props]` ;
  - si la réponse est la méthode gérant tel événement du composant, alors on mettra la donnée dans l'objet rendu par la fonction `[data]` ;

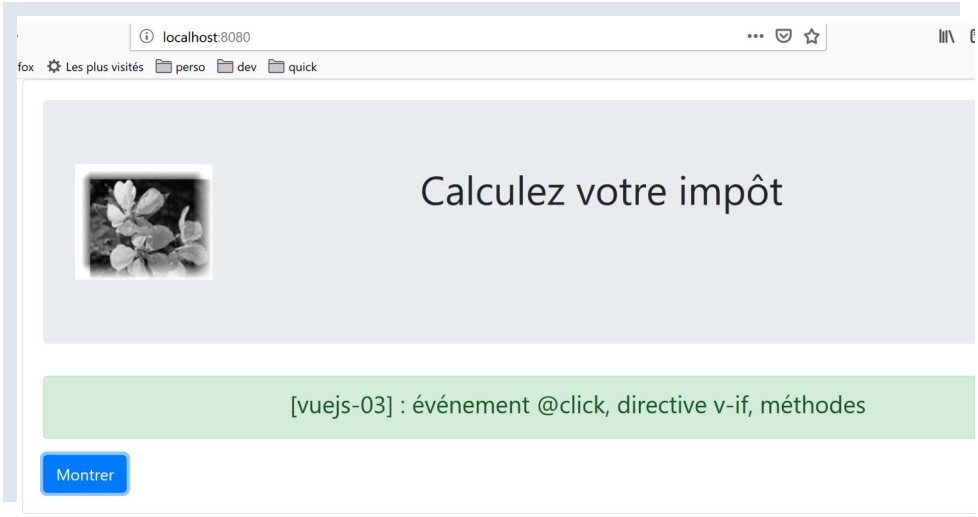
  Le `[template]` utilise ici les données dynamiques suivantes :

  - `[show]`, ligne 8. Cette donnée est manipulée en interne par la méthode `[changer]` qui gère l'événement `[click]` sur le bouton de la ligne 12. C'est donc un attribut construit par la fonction `[data]` (ligne 29) ;
  - `[msg]`, ligne 9. C'est un message fixé par le composant parent. On le met donc dans l'attribut `[props]` (ligne 21) ;
  - `[buttonTitle]` ligne 12. Cette donnée est manipulée en interne par la méthode `[changer]` qui gère l'événement `[click]` sur le bouton de la ligne 12. C'est donc un attribut construit par la fonction `[data]` (ligne 27) ;
- les noms des attributs `[name, props, data, methods]` de l'objet exporté par le composant sont prédéfinis. On ne peut pas utiliser d'autres noms ;
- ligne 12 : l'attribut `[@click]` du bouton sert à désigner la méthode qui doit réagir au clic sur le bouton. Cette méthode doit se trouver dans la propriété `[methods]` du composant ;
- ligne 33 : l'attribut `[methods]` du composant réunit toutes les méthodes de celui-ci. La plupart du temps ce sont des fonctions qui réagissent à un événement du composant ;
- lignes 35-46 : la méthode `[changer]` est appelée lorsque l'utilisateur clique sur le bouton :
  - si l'alerte `[warning]` est affichée alors elle est cachée et le texte du bouton devient `[Montrer]` (ligne 39) ;
  - si l'alerte `[warning]` est cachée alors elle est affichée et le texte du bouton devient `[Cacher]` (ligne 43) ;
  - pour afficher / cacher l'alerte `[warning]`, on modifie la valeur du booléen `[show]` (lignes 38 et 42) ;
  - lorsqu'une méthode doit référencer l'attribut `[attr]` rendu par la fonction `[data]`, on écrit `[this.attr]` (lignes 38 et 42). Cela signifie que les attributs de l'objet rendu par la fonction `[data]` sont des attributs directs du composant `[this]` ;

5.4 Exécution du projet

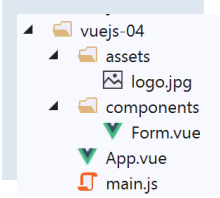



localhost:8080
Les plus visités
perso
dev
quick
Calculez votre impôt
[vuejs-03] : événement @click, directive v-if, méthodes
Information...
Cacher


localhost:8080
Les plus visités
perso
dev
quick
Calculez votre impôt
[vuejs-03] : événement @click, directive v-if, méthodes
Montrer

6 projet [vuejs-04] : directives [v-model, v-bind], attributs calculés, formulaire de saisie

L'arborescence du projet **[vuejs-04]** est la suivante :



6.1 Le script principal [main.js]

C'est le même que dans l'exemple précédent.

6.2 Le composant principal [App]

Le code de **[App.vue]** est le suivant :

```
1.  <template>
2.    <b-container>
3.      <b-card>
4.        <!-- message de présentation -->
5.        <b-row>
6.          <b-col cols="8">
7.            <b-alert show variant="success" align="center">
8.              <h4>[vuejs-04] : directives [v-model, v-bind], attributs calculés, formulaire de saisie</h4>
9.            </b-alert>
10.         </b-col>
11.       </b-row>
12.       <Form />
13.     </b-card>
14.   </b-container>
15. </template>
16.
17.
18. <script>
19.   import Form from "./components/Form.vue";
20.
21.   export default {
22.     name: "app",
23.     components: {
24.       Form
25.     }
26.   };
27. </script>
```

Le composant **[App.vue]** utilise le nouveau composant **[Form]** (lignes 12, 19, 24).

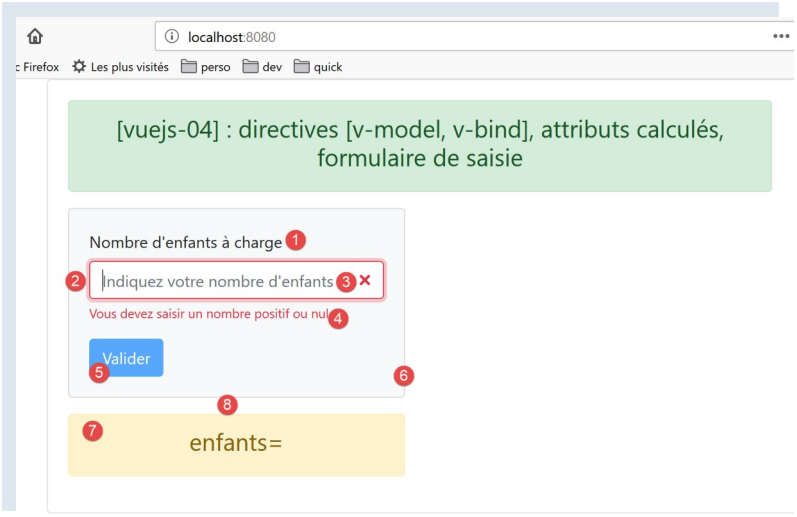
6.3 Le composant [Form]

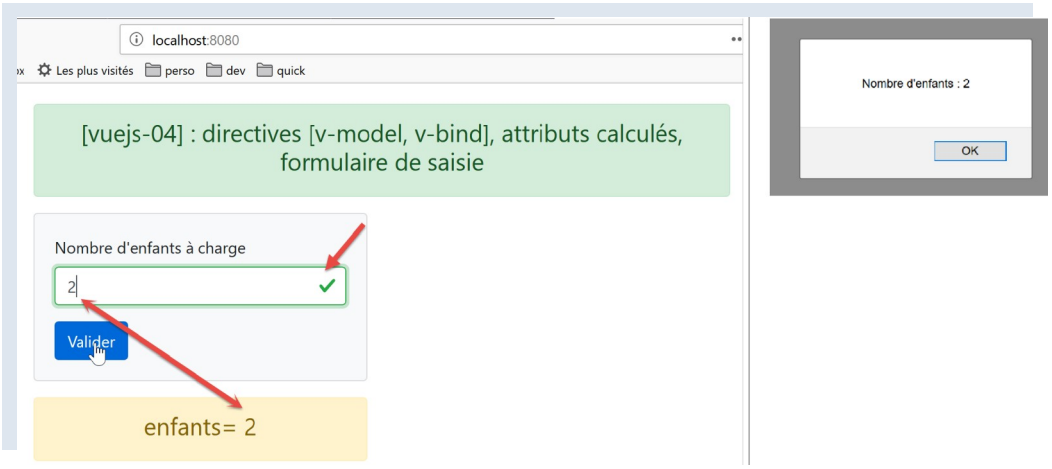
Le code du composant **[Form]** est le suivant :

```
1.  <template>
2.    <div>
3.      <!-- formulaire -->
4.      <b-form>
5.        <!-- éléments du formulaire -->
6.        <b-row>
7.          <b-col cols="4">
8.            <b-card bg-variant="light">
9.              <b-form-group label="Nombre d'enfants à charge" label-for="enfants">
10.               <b-input type="text"
11.                        id="enfants"
12.                        placeholder="Indiquez votre nombre d'enfants"
13.                        v-model="enfants"
14.                        v-bind:state="enfantsValide" />
```

```
15.               <b-form-invalid-feedback :state="enfantsValide">Vous devez saisir un nombre positif ou
    nul</b-form-invalid-feedback>
16.             </b-form-group>
17.             <!-- bouton -->
18.             <b-button variant="primary" :disabled="formInvalide" @click="doSomething">Valider</b-button>
19.           </b-card>
20.         </b-col>
21.       </b-row>
22.     </b-form>
23.     <b-row class="mt-3">
24.       <b-col cols="4">
25.         <b-alert show variant="warning" align="center">
26.           <h4>enfants= {{enfants}}</h4>
27.         </b-alert>
28.       </b-col>
29.     </b-row>
30.   </div>
31. </template>
32.
33. <!-- script -->
34. <script>
35.   export default {
36.     // nom
37.     name: "Form",
38.     // attributs statiques du composant
39.     data() {
40.       return {
41.         // nbre d'enfants
42.         enfants: ""
43.       };
44.     },
45.
46.     // attributs calculés
47.     computed: {
48.       // attribut [formInvalide]
49.       formInvalide() {
50.         return !this.enfantsValide;
51.       },
52.       // attribut [enfantsInvalide]
53.       enfantsValide() {
54.         return Boolean(this.enfants.match(/^\s*\d+\s*$/));
55.       }
56.     },
57.     // méthodes
58.     methods: {
59.       doSomething() {
60.         // le nbre d'enfants est connu lorsque la validation a lieu
61.         alert("Nombre d'enfants : " + this.enfants);
62.       }
63.     }
64.   };
65. </script>
```

Rendu visuel





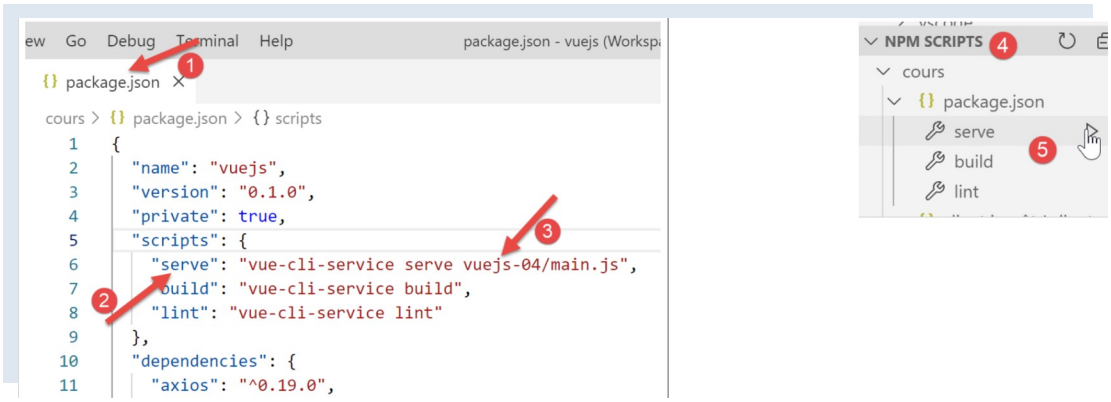
Commentaires

- lignes 4-32 : la balise <b-form> introduit un formulaire Bootstrap ;
- ligne 6 : la balise <b-row> introduit une ligne dans le formulaire ;
- ligne 7 : la balise <b-col cols='4'> introduit une colonne s'étalant sur 4 colonnes Bootstrap ;
- ligne 8 : la balise <b-card> `[6]` introduit une carte Bootstrap, une zone encadrée par une bordure ;
- ligne 9 : la balise <b-form-group> introduit un groupe d'éléments du formulaire liés entre-eux. Ici un texte (attribut `[label]`) `[1]` est lié à une zone de saisie (attribut `[label-for]`). La valeur de `[label-for]` est la valeur du champ `[id]`, ligne 12, de la zone de saisie ;
- lignes 10-14 : la balise <b-input> `[2]` introduit une zone de saisie :
  - ligne 10 : `[type='text']` indique qu'on peut taper du texte dans la zone de saisie. On aurait pu écrire `[type='number']` avec des contraintes `[min='val1' max='val2' step='val3']` puisqu'on attend un nombre d'enfants. On a mis `[type='text']` afin de montrer comment vérifier la validité d'une saisie ;
  - ligne 12 : l'attribut `[placeholder] [3]` fixe le message affiché dans la zone de saisie tant que l'utilisateur n'a rien saisi ;
  - ligne 13 : la directive `[v-model]` associe, de façon **bidirectionnelle** la valeur saisie avec l'attribut `[enfants]`, ligne 42, du composant :
    - lorsque la valeur saisie change, alors la valeur de l'attribut `[enfants]` change également ;
    - lorsque la valeur de l'attribut `[enfants]` change, alors la valeur saisie change également, ç-à-d que le contenu de `[2]` change ;
    - le point important à comprendre est que, grâce au mécanisme précédent, lorsque l'utilisateur clique sur le bouton `[Valider] [5]`, l'attribut `[enfants]` de la ligne 42 a pour valeur, la valeur saisie en `[2]` ;
  - ligne 14 : la directive `[v-bind]` introduit une liaison entre d'un côté un attribut de la balise <b-input>, ici l'attribut `[state]` avec un attribut du composant, ici `[enfantsValide]`, ligne 53. L'attribut `[enfantsValide]` est particulier en ce sens que c'est une **fonction** qui rend la valeur de l'attribut. On appelle, attribut **calculé**, ce type d'attribut. On trouve les attributs calculés dans la propriété `[computed]`, ligne 47, du composant. Les attributs calculés s'utilisent de la même façon que les attributs statiques de la fonction `[data]` : On n'écrit pas, ligne 14, `[v-bind:state='enfantsValide()']` mais `[v-bind:state='enfantsValide']`, **sans les parenthèses**. Aussi à la lecture du `[template]`, on ne sait pas distinguer un attribut **calculé** d'un attribut **statique**. Il faut pour cela regarder le code du script du composant ;
  - ligne 14 : l'attribut `[state]` va fixer l'état valide / invalide de la valeur saisie : si `[enfantsValide]` rend la valeur `[true]`, la valeur saisie est considérée comme valide, sinon comme invalide. La copie d'écran ci-dessus montre le composant `[b-input]` lorsque la fonction `[enfantsValide]` rend la valeur `[false]` ;
  - ligne 15 : la balise <b-form-invalid-feedback> `[4]` permet d'afficher un message lorsque la saisie en `[2]` est invalide. Son attribut `[:state='enfantsValide']` est identique à l'attribut `[v-bind:state='enfantsValide']` de la ligne 14. On peut omettre la directive `[v-bind]` mais il faut garder le signe `[:]`. Le message d'erreur s'affiche donc lorsque l'attribut `[enfantsValide]` vaut `[false]` ;
  - ligne 16 : fin du groupe d'éléments <b-group> ;
  - ligne 18 : le bouton `[5]` qui va permettre de valider la saisie :
    - il sera bleu `[variant='primary']` ;
    - `[:disabled="formInvalide"]` : l'attribut `[disabled]` permet de valider / invalider le bouton. Cet attribut est lié (v-bind) à l'attribut calculé `[formInvalide]` de la ligne 49 ;
    - `[@click="doSomething"]` : lorsque l'utilisateur cliquera sur le bouton, la méthode `[doSomething]`, ligne 59, sera exécutée ;
  - lignes 19-22 : fermeture des différentes balises ouvertes ;
  - lignes 23-29 : une nouvelle ligne dans le `[template]`. `[class='mt-3']` signifie `[margin (m) top (t) égale à 3 spacers]`. `[spacer]` est une mesure d'espacement de Bootstrap. Cette classe génère l'espacement `[8]` dans la copie d'écran ci-dessus. Sans cette classe, la zone `[7]` est collée à la zone `[1-6]` ;
  - ligne 24 : une colonne occupant 4 colonnes Bootstrap ;

- lignes 25-27 : une alerte **[warning]** affichant la valeur de l'attribut statique **[enfants]** (ligne 42). Comme cet attribut a une liaison bidirectionnelle avec la zone de saisie, dès que l'utilisateur modifie celle-ci, la valeur affichée dans l'alerte change également ;
- lignes 34-65 : le code jS du composant ;
- ligne 42 : l'unique attribut statique du composant ;
- lignes 47-56 : les attributs calculés du composant ;
- lignes 53-55 : la saisie est considérée valide si c'est un nombre entier positif éventuellement précédé / suivi par des 'espaces' ;
- lignes 49-51 : le formulaire est considéré comme valide si la saisie du nombre d'enfants est valide. En général, un formulaire a plusieurs saisies et est considéré valide si celles-ci sont toutes valides ;
- lignes 58-63 : les méthodes du composant qui réagissent aux événements de celui-ci. Ici, il n'y a qu'un événement : le **[click]** sur le bouton. On se contente d'afficher la valeur saisie pour montrer qu'on a bien accès à celle-ci ;

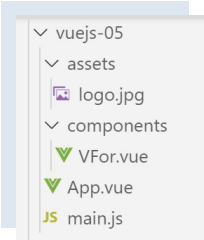
6.4 Exécution du projet

On modifie le fichier **[package.json]** et on exécute le projet :



7 projet [vuejs-05] : directive [v-for]

Le projet **[vuejs-05]** présente la directive **[v-for]** :



7.1 Le script principal [main.js]

Le code du script principal **[main.js]** est identique à celui du script **[main.js]** des projets précédents.

7.2 Le composant principal [App]

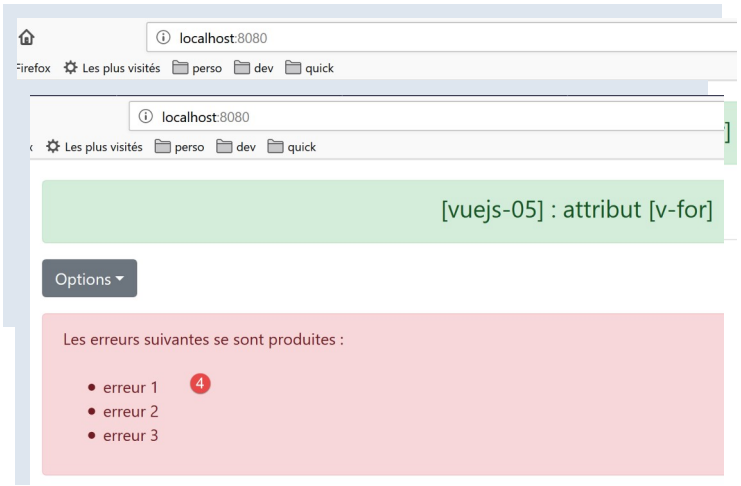
Le code du composant **[App]** est le suivant :

```
1.  <template>
2.    <b-container>
3.      <b-card>
4.        <b-alert show variant="success" align="center">
5.          <h4>[vuejs-05] : attribut [v-for]</h4>
6.        </b-alert>
7.        <VFor />
8.      </b-card>
9.    </b-container>
10. </template>
11.
12.
13. <script>
14.   import VFor from "./components/VFor.vue";
15.
16.   export default {
17.     name: "app",
18.     components: {
19.       VFor
20.     }
21.   };
22. </script>
```

- lignes 7, 14, 19 : le composant **[App]** utilise le composant **[VFor]** ;

7.3 Le composant [VFor]

Le rendu visuel sera le suivant :



Le code du composant **[VFor]** est le suivant :

```
1.  <template>
2.    <div>
3.      <!-- une liste déroulante -->
4.      <b-dropdown id="dropdown" text="Options">
5.        <b-dropdown-item v-for="(option,index) in options"
6.                         :key="option.id"
7.                         @click="select(index)">{{option.text}}</b-dropdown-item>
8.      </b-dropdown>
9.      <!-- bouton -->
10.     <b-button class="ml-3"
11.               variant="primary"
12.               @click="generateErrors"
13.               v-if="!error">Générer une liste d'erreurs</b-button>
14.     <!-- alerte -->
15.     <b-alert show variant="danger" v-if="error" class="mt-3">
16.       Les erreurs suivantes se sont produites :
17.       <br />
18.       <ul>
19.         <li v-for="(erreur,index) in erreurs" :key="index">{{erreur}}</li>
20.       </ul>
21.     </b-alert>
22.   </div>
23. </template>
24. <!-- script -->
25. <script>
26.   export default {
27.     name: "VFor",
28.
29.     // propriétés statiques du composant
30.     data() {
31.       return {
32.         // liste des erreurs
33.         erreurs: [],
34.         // erreur ou pas
35.         error: false,
36.         // liste des options du menu
37.         options: [
38.           { text: "option 1", id: 1 },
39.           { text: "option 2", id: 2 },
40.           { text: "option 3", id: 3 }
41.         ]
42.       };
43.     },
44.
45.     // méthodes
46.     methods: {
47.       // génération d'une liste d'erreurs
48.       generateErrors() {
49.         this.erreurs = ["erreur 1", "erreur 2", "erreur 3"];
50.         this.error = true;
51.       },
52.       // l'utilisateur a sélectionné une option
53.       select(index) {
54.         alert("Vous avez choisi : " + this.options[index].text);
55.       }
56.     }
57.   };
58. </script>
```

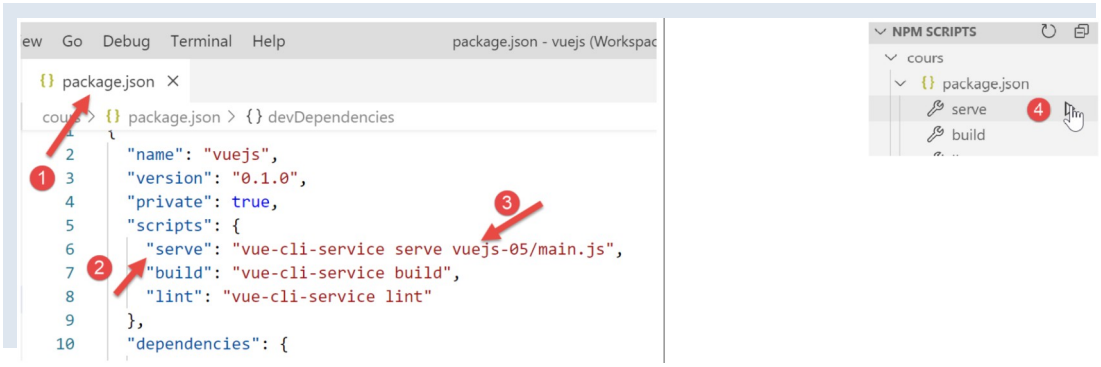
Commentaires

- ligne 4 : la balise <b-dropdown> sert à définir une liste déroulante **[1]** sous la forme d'un bouton qu'on clique pour voir les options de la liste **[2]**. **[text='Options']** définit le texte affiché sur le bouton **[1]** ;
- lignes 5-7 : la balise <dropdown-item> définit un élément de la liste déroulante ;
- ligne 5 : l'attribut **[v-for]** indique que la balise <dropdown-item> doit être répétée pour chaque élément **[option]** de l'attribut **[options]**, lignes 37-41, du composant. **[index]** représente le n° de l'élément dans la liste **[0, 1, ..., n]**. Le nom de l'élément **[option]** ainsi que celui de l'index sont libres. On aurait pu écrire **[<b-dropdown-item v-for="(o,i) in options" :key="o.id" @click="select(i)">{{o.text}}</b-dropdown-item>]** ;
- ligne 6 : si on oublie l'attribut **[key]**, ESLint émet un warning. La valeur de l'attribut **[key]** doit persister dans le temps. Aussi la valeur **[index]** de l'élément ne convient-elle pas. Car si cet élément est supprimé, les valeurs **[index]** de ceux qui sont derrière lui dans la liste vont être décrémentées de 1. Aussi ici, prend-on comme valeur de clé la valeur **[option.id]**, lignes 38-40, qui ne changera pas en cas de suppression d'un élément. L'attribut **[key]** est un élément d'optimisation de régénération

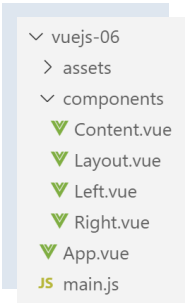
du DOM (Document Object Model) par `[Vue.js]` lorsque la liste doit être régénérée. Notez la notation `[:key]`, car `[key]` a une valeur dynamique ;

- ligne 7 : la méthode `[select(index)]`, lignes 49-51, sera appelée lorsque l'utilisateur cliquera sur un élément de la liste ;
- ligne 7 : le texte de l'option sera la valeur `[option.text]` définie aux lignes 37-41 ;
- ligne 10 : le bouton `[3]`. `[class='ml-3]` signifie margin (m) left (l) de trois spacers. `[@click="generateErrors"]` indique que la méthode `[generateErrors]`, lignes 45-48, sera exécutée lors d'un `[click]` sur le bouton. `[v-if="!error"]` indique que l'affichage du bouton est conditionné à la valeur de l'attribut statique `[error]` de la ligne 35 ;
- lignes 15-21 : une alerte de type `[danger] [4]` contrôlée elle-aussi par l'attribut statique `[error]` de la ligne 35. L'attribut `[class='mt-3']` (**m**argin **t**op **3** spacers) fixe l'espace entre cette alerte et l'élément qui est au-dessus de lui ;
- ligne 27 : la balise HTML
 fait un saut de ligne ;
- ligne 18 : début d'une liste non ordonnée `[ul=unordered list]` ;
- ligne 19 : la balise <li> définit un élément de la liste <ul>. Là encore, on utilise une directive `[v-for]` pour générer la balise plusieurs fois. Autant de fois ici que le tableau `[erreurs]` de la ligne 33 aura d'éléments. On utilise ici l'attribut `[:key=index]`. On a dit précédemment que l'index des éléments d'une liste ne faisait pas un bon discriminant entre éléments de la liste car en cas de suppression d'un élément, tous les éléments qui suivent voient leur index changer. Ici ça n'a pas d'importance car les éléments de la liste d'erreurs ne sont pas susceptibles d'être supprimés ;
- ligne 19 : l'élément sert à afficher l'élément `[erreur]` de la liste `[erreurs]` ;
- lignes 30-43 : tous les éléments dynamiques du `[template]` sont des attributs du composant. Il n'y a ici aucune propriété de type `[props]` dont la valeur serait fixée par le composant parent ;
- lignes 48-55 : la méthode `[generateErrors]` génère la liste d'erreurs à afficher par la balise <ul> des lignes 16-18. De plus, elle modifie l'attribut statique `[error]`, à la fois pour afficher cette liste d'erreurs (ligne 15) et cacher le bouton de génération (ligne 13) ;

7.4 Exécution du projet



8 projet [vuejs-06] : mise en page d'une vue avec des slots



8.1 Le script principal [main.js]

Le script principal **[main.js]** reste inchangé.

8.2 Le composant principal [App]

Le code du composant **[App]** est le suivant :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <b-alert show variant="success" align="center">
5.          <h4>[vuejs-06] : mise en page avec des slots</h4>
6.        </b-alert>
7.        <Content />
8.      </b-card>
9.    </div>
10. </template>
11.
12.
13. <script>
14.   import Content from './components/Content'
15.   export default {
16.     name: "app",
17.     // composants utilisés
18.     components: {
19.       Content
20.     }
21.   };
22. </script>
```

Le composant **[App]** introduit un nouveau composant appelé **[Content]** (lignes 7, 14, 19).

8.3 Le composant [Layout]

Le composant **[Layout]** sert à définir une mise en page des vues de l'application :

```
1.  <template>
2.    <!-- ligne -->
3.    <b-row>
4.      <!-- zone à trois colonnes -->
5.      <b-col cols="3" v-if="left">
6.        <slot name="slot-left" />
7.      </b-col>
8.      <!-- zone à neuf colonnes -->
9.      <b-col cols="9" v-if="right">
10.       <slot name="slot-right" />
11.     </b-col>
12.   </b-row>
13. </template>
14.
15. <script>
16.   export default {
17.     name: "patron",
18.     // paramètres
```

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

Commentaires

- le composant **[Layout]** définit une unique ligne (lignes 3-12). Celle-ci est divisée en deux colonnes :
  - une colonne composée de 3 colonnes Bootstrap (lignes 5-7) ;
  - une colonne composée de 9 colonnes Bootstrap (lignes 9-11) ;
- ligne 5 : la présence de la colonne de gauche (lignes 5-7) est conditionnée par la valeur du paramètre **[left]**, définie dans la propriété **[props]** du composant (lignes 20-22) ;
- ligne 9 : la présence de la colonne de droite (lignes 9-11) est conditionnée par la valeur du paramètre **[right]**, définie dans la propriété **[props]** du composant (lignes 23-25) ;
- les valeurs des paramètres **[left, right]** doivent être fixées par un composant parent du composant **[Layout]** ;
- ligne 6 : on ne fixe pas le contenu de la colonne de gauche. On donne simplement un nom à cette colonne avec la balise <slot>. Ici elle s'appelera **[slot-left]**. Un composant utilisant le composant **[Content]** devra indiquer ce qu'il veut mettre dans la zone appelée **[slot-left]** ;
- ligne 10 : la colonne de droite s'appellera **[slot-right]** ;

8.4 Le composant [Right]

Le composant **[Right]** est le suivant :

```
1.  <template>
2.    <!-- un message dans une alerte de type warning -->
3.    <b-alert show variant="warning" align="center">
4.      <h4>{{msg}}</h4>
5.    </b-alert>
6.  </template>
7.
8.  <!-- script -->
9.  <script>
10.   export default {
11.     name: "droite",
12.     // paramètres
13.     props: {
14.       msg: String
15.     }
16.   };
17. </script>
```

- lignes 3-5 : le composant **[Right]** affiche un message dans une alerte de type **[warning]**. Ce message **[msg]** est défini comme étant un paramètre du composant, ligne 14. Le composant parent devra donc lui donner une valeur ;

8.5 Le composant [Left]

Le composant **[Left]** est le suivant :

```
1.  <template>
2.    <!-- un message dans une alerte de type primary -->
3.    <b-alert show variant="primary" align="center">
4.      <h4>{{msg}}</h4>
5.    </b-alert>
6.  </template>
7.
8.  <!-- script -->
9.  <script>
10.   export default {
11.     name: "gauche",
12.     // paramètres
13.     props: {
14.       msg: String
15.     }
16.   };
```

```
17. </script>
```

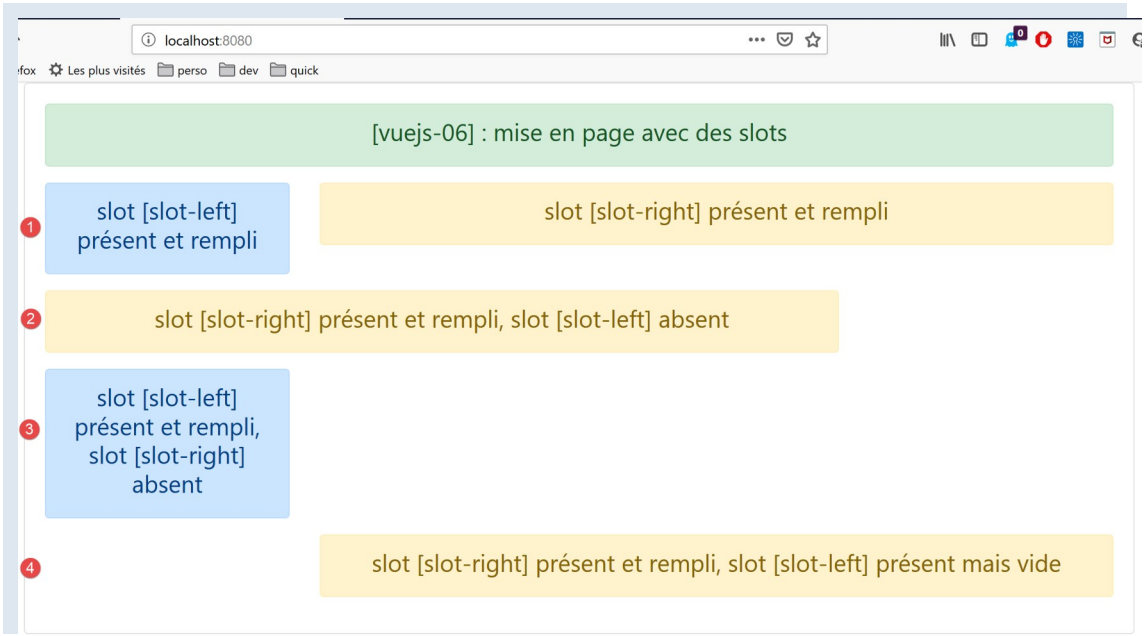
- lignes 3-5 : le composant `[Left]` affiche un message dans une alerte de type `[primary]`. Ce message `[msg]` est défini comme étant un paramètre du composant, ligne 14. Le composant parent devra donc lui donner une valeur ;

8.6 Le composant `[Content]`

On se rappelle que le composant `[Content]` est le composant affiché par la vue principale `[App.vue]` :

```
1.  <template>
2.    <div>
3.      <!-- colonnes gauche et droite remplies -->
4.      <Layout :left="true" :right="true">
5.        <Right slot="slot-right" msg="slot [slot-right] présent et rempli" />
6.        <Left slot="slot-left" msg="slot [slot-left] présent et rempli" />
7.      </Layout>
8.      <!-- colonnes gauche basente. colonne droite remplie -->
9.      <Layout :left="false" :right="true">
10.       <Right slot="slot-right" msg="slot [slot-right] présent et rempli, slot [slot-left] absent" />
11.     </Layout>
12.     <!-- colonnes gauche remplie, colonne droite absente -->
13.     <Layout :left="true" :right="false">
14.       <Left slot="slot-left" msg="slot [slot-left] présent et rempli, slot [slot-right] absent" />
15.     </Layout>
16.     <!-- colonnes gauche présente mais pas remplie, colonne droite remplie -->
17.     <Layout :left="true" :right="true">
18.       <Right slot="slot-right" msg="slot [slot-right] présent et rempli, slot [slot-left] présent mais
    vide" />
19.     </Layout>
20.   </div>
21. </template>
22.
23. <!-- script -->
24. <script>
25.   import Layout from "./Layout";
26.   import Left from "./Left";
27.   import Right from "./Right";
28.   export default {
29.     name: "contenu",
30.     // composants
31.     components: {
32.       Layout,
33.       Left,
34.       Right
35.     }
36.   };
37. </script>
```

Rendu visuel



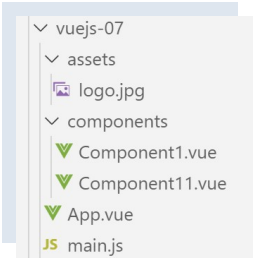
Commentaires

- le composant **[Layout]** est utilisé 4 fois (lignes 4-7, 9-11, 13-15, 17-19). Si on se rappelle que le composant **[Layout]** définit 1 ligne, le **[template]** ci-dessus définit quatre lignes. On se rappelle également que la ligne du **[Layout]** définit deux colonnes :
  - une colonne appelée **[slot-left]** qui occupe les 3 colonnes Bootstrap de gauche ;
  - une colonne appelée **[slot-Right]** qui occupe les 9 colonnes Bootstrap de droite ;
- lignes 4-7 : définit une ligne **[1]** où le composant **[Left]** occupe la colonne **[slot-left]** et le composant **[Right]** la colonne **[slot-right]** ;
- lignes 9-11 : définit une ligne **[2]** où la colonne **[slot-left]** n'est pas affichée et le composant **[Right]** occupe la colonne **[slot-right]** ;
- lignes 13-15 : définit une ligne **[3]** où la colonne **[slot-right]** n'est pas affichée et le composant **[Left]** occupe la colonne **[slot-left]** ;
- lignes 17-19 : définit une ligne **[4]** où la colonne **[slot-left]** est affichée **[:left='true']** mais pas remplie et le composant **[Right]** occupe la colonne **[slot-right]** ;

Au final, le composant **[Layout]** a servi de mise en page au composant **[Content]**.

9 projet [vuejs-07] : remontée d'événements dans la hiérarchie des composants

L'arborescence du projet est la suivante :



9.1 Le script principal [main.js]

Le script principal **[main.js]** reste inchangé.

9.2 Le composant principal [App]

Le code du composant **[App]** est le suivant :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <b-alert show variant="success" align="center">
5.          <h4>[vuejs-07] : remontée d'événements dans la hiérarchie des composants</h4>
6.        </b-alert>
7.        <Component1 />
8.      </b-card>
9.    </div>
10. </template>
11.
12. <script>
13.   import Component1 from './components/Component1'
14.   export default {
15.     name: "app",
16.     components: {
17.       Component1
18.     }
19.   };
20. </script>
```

Le composant **[App]** utilise un nouveau composant **[Component1]** (lignes 7, 13, 17).

9.3 Le composant [Component11]

Le code du composant **[Component11]** est le suivant :

```
1.  <template>
2.    <b-button @click="createEvent">Créer un événement</b-button>
3.  </template>
4.  <!-- script -->
5.  <script>
6.    export default {
7.      name: "component11",
8.      // méthodes du composant
9.      methods: {
10.       // gestion de l'évt [click] sur le bouton
11.       createEvent() {
12.         // on émet un événement couplé à une donnée
13.         this.$emit("someEvent", { x: 2, y: 4 })
14.       }
15.     }
16.   };
17. </script>
```

Commentaires

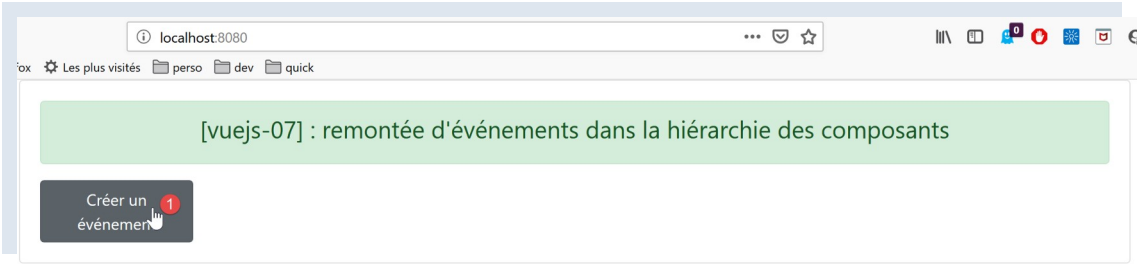
- lignes 1-3 : le composant **[Component11]** ne comporte qu'un bouton qu'on peut cliquer. Lorsqu'on le clique, la méthode **[createEvent]** des lignes 11-14 est exécutée ;
- ligne 13 : chaque instance **[Vue]** dispose d'une méthode **[$emit]** qui permet d'émettre un événement :
  - le 1er paramètre est le nom de l'événement émis ;
  - le second paramètre est la donnée que l'on veut associer à cet événement ;
  - l'événement émis remonte la hiérarchie des composants qui chapeautent le composant **[Component11]**. Il remonte la hiérarchie jusqu'à ce qu'il trouve un composant disposé à le traiter. Cette remontée commence par le composant parent de **[Component11]** ;

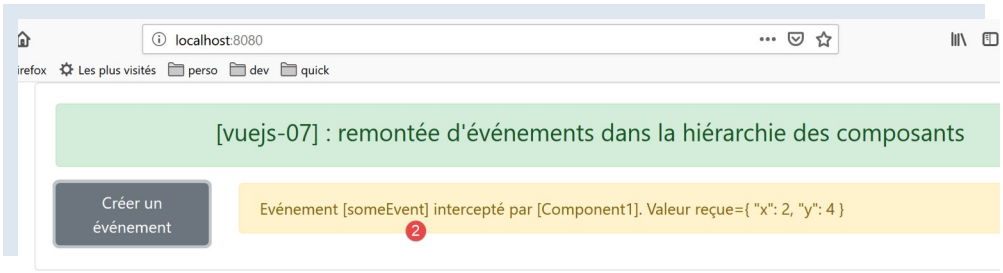
9.4 Le composant [Component1]

Le code du composant **[Component1]** est le suivant :

```
1.  <template>
2.    <b-row>
3.      <!-- le composant qui lance l'événement -->
4.      <b-col cols="2">
5.        <Component11 @someEvent="doSomething" />
6.      </b-col>
7.      <!-- message affiché par la méthode de gestion de l'évt-->
8.      <b-col>
9.        <b-alert show
10.                variant="warning"
11.                v-if="showMsg">Evénement [someEvent] intercepté par [Component1]. Valeur reçue={{data}}</b-
    alert>
12.     </b-col>
13.   </b-row>
14. </template>
15.
16. <script>
17.   import Component11 from "./Component11";
18.   export default {
19.     name: "component1",
20.     // composants
21.     components: {
22.       Component11
23.     },
24.     // état du composant
25.     data() {
26.       return {
27.         data: "",
28.         showMsg: false
29.       };
30.     },
31.     // méthodes de gestion des évts
32.     methods: {
33.       doSomething(data) {
34.         // data est l'objet qui a été associé à l'évt [someEvent]
35.         this.data = data;
36.         // affiche l'alerte
37.         this.showMsg = true;
38.       }
39.     }
40.   };
41. </script>
```

Rendu visuel

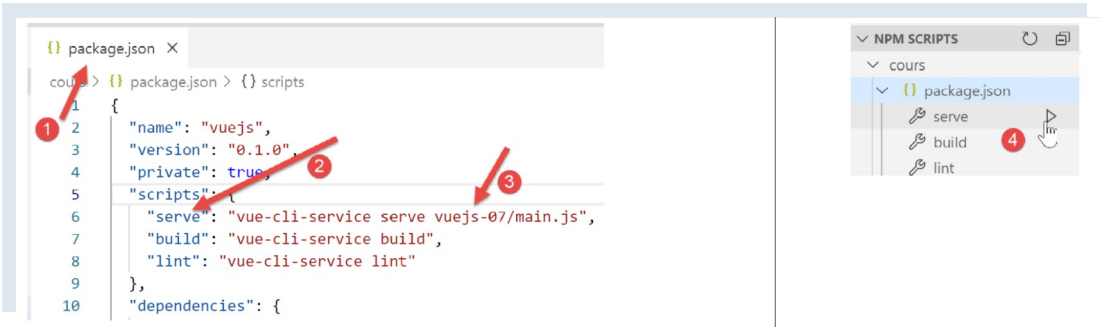




Commentaires

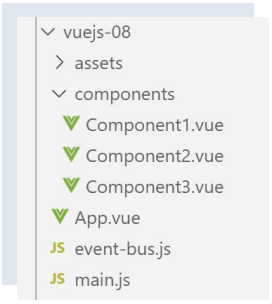
- ligne 5 : **`[Component1]`** est parent de **`[Component11]`** et peut donc 'écouter' (c'est le terme) les événements émis par ce composant. On sait que celui-ci peut émettre l'événement **`[someEvent]`**. L'attribut **`[@someEvent="doSomething"]`** indique que si l'événement **`[someEvent]`** émis par **`[Component11]`** se produit, alors la méthode **`[doSomething]`** de la ligne 33 doit être exécutée ;
- lignes 9-11 : un message affiché par la méthode **`[doSomething]`**. Son affichage est contrôlé par le booléen **`[showMsg]`** de la ligne 28. L'alerte affiche l'attribut **`[data]`** de la ligne 27 ;
- ligne 33 : la méthode **`[doSomething]`** exécutée lorsque l'événement **`[someEvent]`** se produit ligne 5, reçoit comme paramètre la donnée **`[data]`** associée à cet événement. Ce paramètre est affecté à l'attribut **`[data]`** de la ligne 27 et qui est affiché par la ligne 11 ;
- ligne 37 : on met l'attribut **`[showMsg]`** à **`[true]`** pour afficher l'alerte des lignes 9-11 ;

9.5 Exécution du projet



10 projet [vuejs-08] : événements indépendants de la hiérarchie des composants, cycle de vie des composants

Le projet **[vuejs-08]** lontre comment des composants non reliés directement dans la hiérarchie des composants peuvent néanmoins communiquer via des événements. L'arborescence du projet **[vuejs-08]** est la suivante :



10.1 Le script principal [main.js]

Le script **[main.js]** reste ce qu'il était dans les exemples précédents.

10.2 Le script [even-bus.js]

Le script **[event-bus.js]** est l'outil que nous allons utiliser pour émettre et écouter des événements :

```
1.  import Vue from 'vue';
2.  const eventBus = new Vue();
3.  export default eventBus;
```

Le script **[event-bus]** se contente de créer une instance de la classe **[Vue]** (lignes 1-2) et de l'exporter (ligne 3). La classe **[Vue]** possède deux méthodes pour gérer les événements :

- **[Vue].$emit(nom, donnée)** : permet d'émettre un événement nommé **[nom]** et associé à la donnée **[donnée]** ;
- **[Vue].$on(nom, fn)** : permet d'intercepter l'événement nommé **[nom]** et de le faire traiter par la fonction **[fn]**. La fonction **[fn]** recevra en paramètre la donnée **[donnée]** associée à l'événement par son émetteur ;

Cette gestion d'événements n'est pas liée à la hiérarchie des composants. Il faut simplement que les composants qui veulent communiquer ainsi utilisent **la même instance** de la classe **[Vue]** pour émettre / écouter les événements. Dans notre exemple, nous utiliserons l'instance **[eventBus]** définie par le script **[event-bus.js]** précédent.

10.3 La vue principale [App.vue]

Le code de la vue principale **[App]** est le suivant :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <b-alert show variant="success" align="center">
5.          <h4>[vuejs-08] : événements indépendants de la hiérarchie des composants, cycle de vie des
    composants</h4>
6.        </b-alert>
7.        <!-- les trois composants qui communiquent par évt -->
8.        <Component1 />
9.        <Component3 />
10.       <Component2 />
11.     </b-card>
12.   </div>
13. </template>
14.
15. <script>
16.   import Component1 from "./components/Component1";
17.   import Component2 from "./components/Component2";
18.   import Component3 from "./components/Component3";
19.   export default {
20.     name: "app",
21.     // composants
```

```
22.     components: {
23.       Component1,
24.       Component2,
25.       Component3
26.     }
27.   };
28. </script>
```

- lignes 8-10 : la vue principale **[App]** utilise trois composants **[Component1, Component2, Component3]** qui vont communiquer par événements. Les trois composants sont au même niveau dans la vue **[App]**. De plus, il n'y aura pas de relation de hiérarchie entre eux ;

10.4 Le composant [Component1]

Le code du composant **[Component1]** est le suivant :

```
1.  <template>
2.    <b-row>
3.      <b-col>
4.        <b-alert show
5.                 variant="warning"
6.                 v-if="showMsg">Evénement [someEvent] intercepté par [Component1]. Valeur reçue={{data}}</b-
    alert>
7.      </b-col>
8.    </b-row>
9.  </template>
10.
11. <script>
12.   import eventBus from "../event-bus.js"
13.   export default {
14.     name: "component1",
15.     // état du composant
16.     data() {
17.       return {
18.         data: "",
19.         showMsg: false
20.       };
21.     },
22.     // méthodes de gestion des évts
23.     methods: {
24.       // gestion de l'evt [someEvent]
25.       doSomething(data) {
26.         this.data = data;
27.         this.showMsg = true;
28.       }
29.     },
30.     // gestion du cycle de vie du composant
31.     // évt [created] - le composant a été créé
32.     created() {
33.       // écoute de l'évt [someEvent]
34.       eventBus.$on("someEvent", this.doSomething);
35.     }
36.   };
37. </script>
```

Commentaires

- lignes 4-6 : une alerte qui affiche la donnée **[data]** de la ligne 18. Cette donnée sera initialisée par le gestionnaire d'événement **[doSomething]** de la ligne 25. Ce gestionnaire est activé à réception de l'événement **[someEvent]** (ligne 34). L'alerte est affichée conditionnellement à la valeur de l'attribut **[showMsg]** de la ligne 19. Cet attribut est lui également positionné par le gestionnaire d'événement **[doSomething]** de la ligne 25 ;
- ligne 32 : la fonction **[created]** est un gestionnaire d'événement. Elle gère l'événement **[created]**, un événement émis au cours du cycle de vie du composant. Il en existe d'autres **[beforeCreate, created, beforeMount, mounted, beforeUpdate, updated, beforeDestroy, destroyed]**. L'événement **[created]** est émis lorsque le composant a été créé ;
- ligne 12 : l'instance **[eventBus]** de la classe **[Vue]** est importée ;
- ligne 34 : elle est utilisée pour écouter l'événement **[someEvent]**. Lorsque celui-ci se produit, la méthode **[doSomething]** de la ligne 25 est appelée ;
- ligne 25 : la méthode **[doSomething]** reçoit comme paramètre **[data]** la donnée que l'émetteur de l'événement **[someEvent]** a associée à l'événement ;
- lignes 26-27 : l'état du composant est modifié pour que l'alerte des lignes 4-6 affiche la donnée reçue ;

10.5 Le composant [Component2]

Le composant **[Component2]** est le suivant :

```
1.  <template>
2.    <div>
3.      <b-button @click="createEvent">Créer un événement</b-button>
4.    </div>
5.  </template>
6.  <!-- script -->
7.  <script>
8.    import eventBus from '../event-bus.js'
9.    export default {
10.     name: "component2",
11.     // méthodes de gestion des évts
12.     methods: {
13.       createEvent() {
14.         eventBus.$emit("someEvent", { x: 2, y: 4 })
15.       }
16.     }
17.   };
18. </script>
```

Commentaires

- ligne 3 : un bouton pour créer un événement. Lorsque l'utilisateur clique sur ce bouton, la méthode **[createEvent]** de la ligne 13 est appelée ;
- ligne 8 : l'instance **[eventBus]** définie par le script **[event-bus.js]** est importée ;
- ligne 14 : cette instance est utilisée pour émettre un événement nommé **[someEvent]** associé à la donnée **[{ x: 2, y: 4 }]**. Au final, lorsque l'utilisateur clique sur le bouton de la ligne 3, l'événement **[someEvent]** est émis. Si on se rappelle la définition de **[Component1]**, celui-ci interceptera cet événement ;

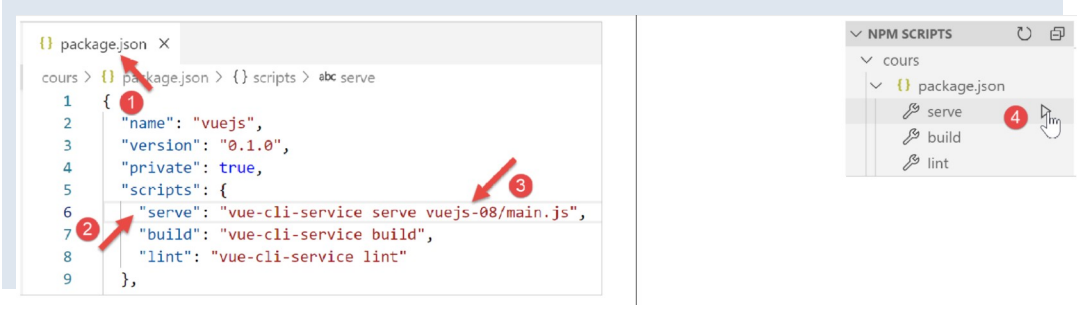
10.6 Le composant [Component3]

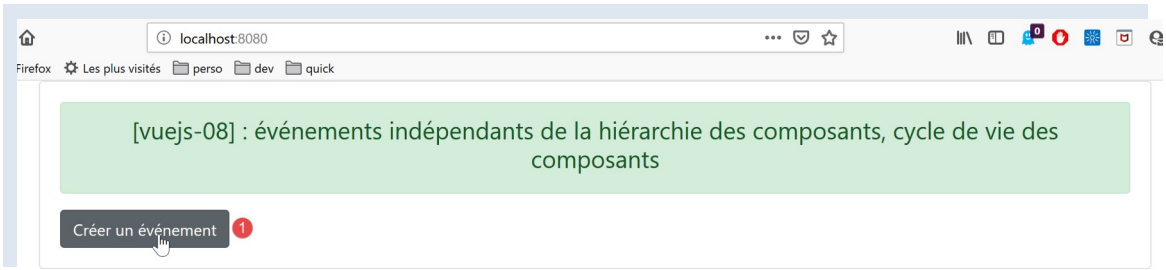
Le code de ce composant est le suivant :

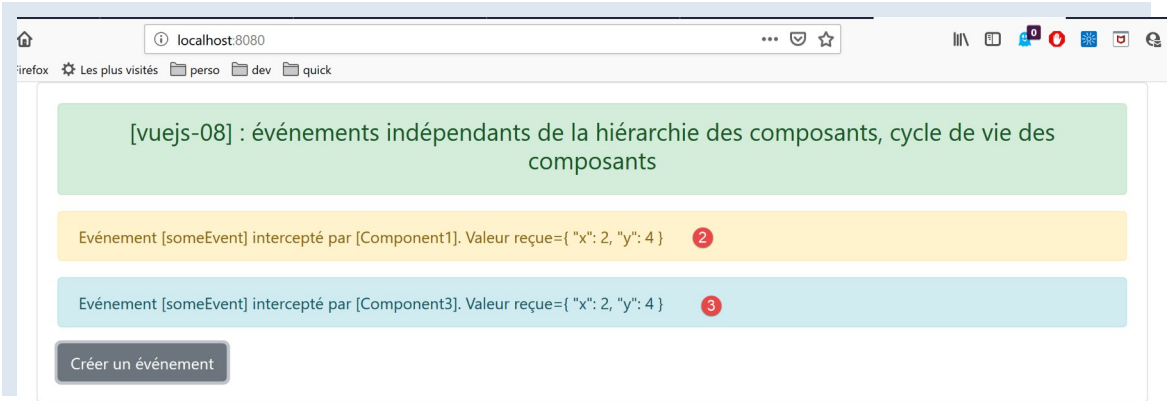
```
1.  <template>
2.    <b-row>
3.      <b-col>
4.        <b-alert show
5.                 v-if="showMsg">Evénement [someEvent] intercepté par [Component3]. Valeur reçue={{data}}</b-
    alert>
6.      </b-col>
7.    </b-row>
8.  </template>
9.
10. <script>
11.   import eventBus from "../event-bus.js"
12.   export default {
13.     name: "component3",
14.     // état du composant
15.     data() {
16.       return {
17.         data: "",
18.         showMsg: false
19.       };
20.     },
21.     methods: {
22.       // gestion de l'evt [someEvent]
23.       doSomething(data) {
24.         this.data = data;
25.         this.showMsg = true;
26.       }
27.     },
28.     // gestion du cycle de vie du composant
29.     // évt [created] - le composant a été créé
30.     created() {
31.       // écoute de l'évt [someEvent]
32.       eventBus.$on("someEvent", this.doSomething);
33.     }
34.   };
35. </script>
```

[Component3] est un clone de **[Component1]**. Il n'est là que pour montrer qu'un événement peut être intercepté par plusieurs composants.

10.7 Exécution du projet [vuejs-08]

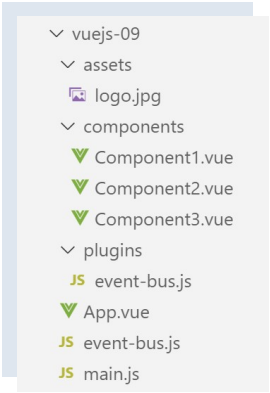






11 projet [vuejs-09] : utilisation d'un plugin [eventBus]

Le projet **[vuejs-09]** est identique au projet **[vuejs-08]** si ce n'est qu'il introduit la notion de plugin. L'arborescence du projet est la suivante :



11.1 Le plugin [./plugins/event-bus]

Le script **[./event-bus.js]** reste ce qu'il était dans l'exemple précédent :

```
1.  import Vue from 'vue';
2.  const eventBus = new Vue();
3.  export default eventBus;
```

Le plugin **[./plugins/event-bus.js]** est le suivant :

```
1.  export default {
2.    install(Vue, eventBus) {
3.      // ajoute une propriété [$eventBus] à la classe Vue
4.      Object.defineProperty(Vue.prototype, '$eventBus', {
5.        // lorsque Vue.$eventBus est référencé, on rend le 2ième paramètre [eventBus]
6.        get: () => eventBus,
7.      })
8.    }
9.  }
```

Commentaires

- un plugin **[Vue]** est un objet ayant une fonction **[install]** (ligne 2). Celle-ci sera automatiquement appelée lorsqu'un code déclare l'utilisation du plugin ;
- lignes 1-9 : l'objet exporté par le script ;
- ligne 2 : la fonction **[install]** accepte ici deux paramètres :
  - **[Vue]** : la fonction obtenue par l'instruction **[import Vue from 'Vue']**. Peut-être assimilée à une classe ;
  - **[eventBus]** : l'objet exporté par le script **[./event-bus]** ;
- lignes 4-7 : modifient la définition (on dit le prototype) de la fonction **[Vue]** en lui ajoutant la propriété **[$eventBus]**. Si on raisonne avec le terme **[classe]**, la propriété **[$eventBus]** est ajoutée à la clase **[Vue]**. Les composants qui sont des instances de **[Vue]** auront donc accès à cette nouvelle propriété ;
- la ligne 6 indique que lorsqu'on référencera la propriété **[Vue].$eventBus**, on obtiendra le paramètre **[eventBus]** de la ligne 2. Nous allons voir un peu plus loin que ce second paramètre sera l'objet **[eventBus]** exporté par le script **[./event-bus.js]**. Donc au final, lorsqu'un composant C utilisera l'expression **[C.$eventBus]** il référencera l'objet **[eventBus]** exporté par le script **[./event-bus.js]**. Cela lui évitera d'importer le script **[./event-bus.js]**. L'intérêt du plugin est là : simplifier l'accès à l'objet **[eventBus]** exporté par le script **[./event-bus.js]** ;
- on notera que le plugin n'a pas à s'appeler lui-même **[event-bus.js]**. On aurait pu l'appeler **[plugin-event-bus]** par exemple ;
- on notera également que le terme $ dans **[$eventBus]** est une convention pour désigner les propriétés de **[Vue]** qui ont été ajoutées via des plugins. On n'est pas obligés d'observer cette convention. Dans ce texte, nous l'observerons ;

11.2 Le script principal [main.js]

Le script **[./plugins/event-bus.js]** définit un plugin pour le framework **[Vue.js]**. Ce plugin n'est pas encore utilisé, juste défini. C'est le script **[main.js]** qui l'active :

```
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. // eventbus
14. import eventBus from './event-bus';
15. import PluginEventBus from './plugins/event-bus';
16. Vue.use(PluginEventBus, eventBus);
17.
18. // configuration
19. Vue.config.productionTip = false
20.
21. // instanciation projet [App]
22. new Vue({
23.   render: h => h(App),
24. }).$mount('#app')
```

Commentaires

- les lignes 14-16 activent le plugin **[PluginEventBus]**. Après la ligne 16, toutes les instances de la classe (fonction) **[Vue]** possèdent la propriété **[$eventBus]** qui pointe pour chacune d'elles sur le même object exporté par le script **[./event-bus.js]**. Ce sera le cas pour chacun des composants du projet ;

11.3 La vue principale [App]

La vue principale **[App]** reste ce qu'elle était dans le projet précédent.

11.4 Le composant [Component1]

Le composant **[Component1]** utilise désormais sa propriété **[$eventBus]** pour écouter l'événement **[someEvent]** :

```
1.  <template>
2.    <b-row>
3.      <b-col>
4.        <b-alert show
5.                 variant="warning"
6.                 v-if="showMsg">Evénement [someEvent] intercepté par [Component1]. Valeur reçue={{data}}</b-
    alert>
7.      </b-col>
8.    </b-row>
9.  </template>
10.
11. <script>
12.   export default {
13.     name: "component1",
14.     // état du composant
15.     data() {
16.       return {
17.         data: "",
18.         showMsg: false
19.       };
20.     },
21.     // méthodes de gestion des évts
22.     methods: {
23.       // gestion de l'evt [someEvent]
24.       doSomething(data) {
25.         this.data = data;
26.         this.showMsg = true;
27.       }
28.     },
29.     // gestion du cycle de vie du composant
30.     // évt [created] - le composant a été créé
31.     created() {
32.       // écoute de l'évt [someEvent]
33.       this.$eventBus.$on("someEvent", this.doSomething);
34.     }
```

```
35.   };
36. </script>
```

Commentaires

- ligne 33, utilisation de la propriété `[this.$eventBus]` du composant. On remarquera de plus que le script ligne 11 n'importe plus le script `[./event-bus.js]` ;

11.5 Le composant `[Component2]`

Le composant `[Component2]` utilise désormais sa propriété `[$eventBus]` pour émettre l'événement `[someEvent]` :

```
1.  <template>
2.    <div>
3.      <b-button @click="createEvent">Créer un événement</b-button>
4.    </div>
5.  </template>
6.  <!-- script -->
7.  <script>
8.    export default {
9.      name: "component2",
10.     // méthodes de gestion des évts
11.     methods: {
12.       createEvent() {
13.         this.$eventBus.$emit("someEvent", { x: 2, y: 4 })
14.       }
15.     }
16.   };
17. </script>
```

Commentaires

- ligne 13, utilisation de la propriété `[this.$eventBus]` du composant. On remarquera de plus que le script ligne 7 n'importe plus le script `[./event-bus.js]` ;

11.6 Composant [Component3]

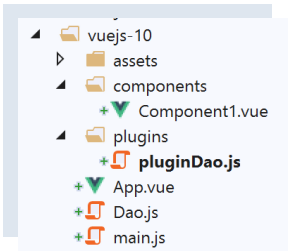
Le composant `[Component3]` a le même code que `[Component1]`. Lui également écoute l'événement `[someEvent]`.

11.7 Exécution du projet

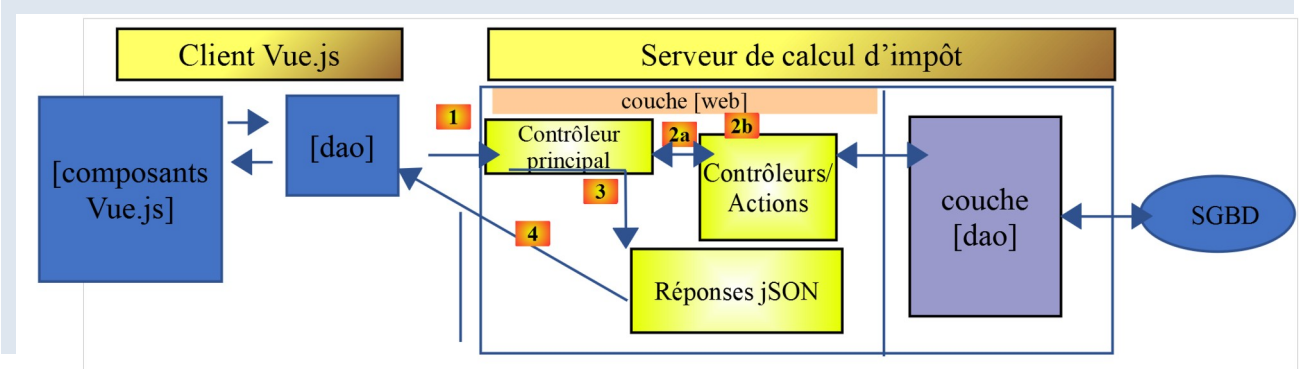
On obtient les mêmes résultats que dans le projet précédent.

12 projet [vuejs-10] : plugin [dao], requêtes HTTP asynchrones

L'arborescence du projet **[vuejs-10]** est la suivante :



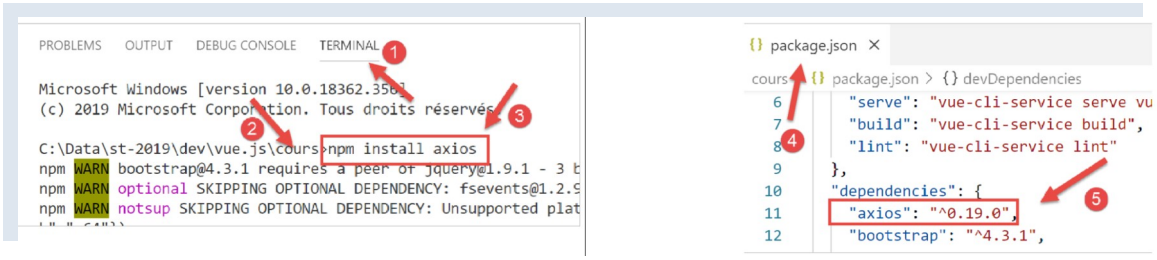
Le projet **[vuejs-10]** montre un composant faisant une requête HTTP à un serveur distant. L'architecture utilisée est la suivante :



Un composant **[Vue.js]** utilise la couche **[dao]** pour dialoguer avec le serveur de calcul de l'impôt.

12.1 Installation des dépendances

L'application **[vuejs-10]** utilise la bibliothèque **[axios]** pour faire les requêtes asynchrones vers le serveur de calcul d'impôt. Il nous faut installer cette dépendance :



- en **[4-5]**, la ligne ajoutée au fichier **[package.json]** après l'installation de la bibliothèque **[axios] [1-3]** ;

12.2 La classe [Dao]

La classe **[Dao]** est celle qui a été développée au paragraphe [La classe Dao]. Nous la redonnons ici pour mémoire :

```
1.  'use strict';
2.
3.  // imports
4.  import qs from 'qs'
5.
6.  // classe [Dao]
7.  class Dao {
8.
9.    // constructeur
10.   constructor(axios) {
11.     this.axios = axios;
12.     // cookie de session
13.     this.sessionCookieName = "PHPSESSID";
14.     this.sessionCookie = '';
15.   }
```

```
16.
17.    // init session
18.    async  initSession() {
19.      // options de la requête HHTP [get /main.php?action=init-session&type=json]
20.      const options = {
21.        method: "GET",
22.        // paramètres de l'URL
23.        params: {
24.          action: 'init-session',
25.          type: 'json'
26.        }
27.      };
28.      // exécution de la requête HTTP
29.      return await this.getRemoteData(options);
30.    }
31.
32.    async  authentifierUtilisateur(user, password) {
33.      // options de la requête HHTP [post /main.php?action=authentifier-utilisateur]
34.      const options = {
35.        method: "POST",
36.        headers: {
37.          'Content-type': 'application/x-www-form-urlencoded',
38.        },
39.        // corps du POST
40.        data: qs.stringify({
41.          user: user,
42.          password: password
43.        }),
44.        // paramètres de l'URL
45.        params: {
46.          action: 'authentifier-utilisateur'
47.        }
48.      };
49.      // exécution de la requête HTTP
50.      return await this.getRemoteData(options);
51.    }
52.
53.    async getAdminData() {
54.      // options de la requête HHTP  [get /main.php?action=get-admindata]
55.      const options = {
56.        method: "GET",
57.        // paramètres de l'URL
58.        params: {
59.          action: 'get-admindata'
60.        }
61.      };
62.      // exécution de la requête HTTP
63.      const data = await this.getRemoteData(options);
64.      // résultat
65.      return data;
66.    }
67.
68.    async  getRemoteData(options) {
69.      // pour le cookie de session
70.      if (!options.headers) {
71.        options.headers = {};
72.      }
73.      options.headers.Cookie = this.sessionCookie;
74.      // exécution de la requête HTTP
75.      let response;
76.      try {
77.        // requête asynchrone
78.        response = await this.axios.request('main.php', options);
79.      } catch (error) {
80.        // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
81.        if (error.response) {
82.          // la réponse du serveur est dans [error.response]
83.          response = error.response;
84.        } else {
85.          // on relance l'erreur
86.          throw error;
87.        }
88.      }
89.      // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-même)
90.      // on récupère le cookie de session s'il existe
91.      const setCookie = response.headers['set-cookie'];
92.      if (setCookie) {
```

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

Le projet **[vuejs-10]** n'utilise que la méthode asynchrone **[initSession]** des lignes 18-30. On rappelle que la classe **[Dao]** est instanciée avec un paramètre **[axios]**, ligne 10, paramètre initialisé par le code appelant. Ce code appelant sera ici le script **[./main.js]**.

12.3 Le plugin [pluginDao]

Le plugin **[pluginDao]** est le suivant :

```
1.  export default {
2.    install(Vue, dao) {
3.      // ajoute une propriété [$dao] à la classe Vue
4.      Object.defineProperty(Vue.prototype, '$dao', {
5.        // lorsque Vue.$dao est référencé, on rend le 2ième paramètre [dao]
6.        get: () => dao,
7.      })
8.    }
9.  }
```

Si on se souvient de l'explication donnée pour le plugin **[event-bus]**, on voit que le plugin **[pluginDao]** crée dans la classe / fonction **[Vue]**, une nouvelle propriété appelée **[$dao]**. Cette propriété aura (ça reste à montrer) pour valeur, l'objet exporté par le script **[./Dao]**, ç-à-d la classe **[Dao]** précédente.

12.4 Le script principal [main.js]

Le code du script principal **[main.js]** est le suivant :

```
1.  // imports
2.  import Vue from 'vue'
3.  import App from './App.vue'
4.  import axios from 'axios';
5.
6.  // plugins
7.  import BootstrapVue from 'bootstrap-vue'
8.  Vue.use(BootstrapVue);
9.
10. // bootstrap
11. import 'bootstrap/dist/css/bootstrap.css'
12. import 'bootstrap-vue/dist/bootstrap-vue.css'
13.
14. // couche [dao]
15. import Dao from './Dao';
16. // configuration axios
17. axios.defaults.timeout = 2000;
18. axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';
19. axios.defaults.withCredentials = true;
20. // instanciation couche [dao]
21. const dao = new Dao(axios);
22.
```

```
23. // plugin [dao]
24. import pluginDao from './plugins/dao'
25. Vue.use(pluginDao, dao)
26.
27. // configuration
28. Vue.config.productionTip = false
29.
30. // instanciation projet [App]
31. new Vue({
32.   render: h => h(App),
33. }).$mount('#app')
```

Le script **[main.js]** :

- instancie la couche **[dao]** aux lignes 14-21 ;
- intègre le plugin **[pluginDao]** aux lignes 24-25 ;
- ligne 15 : la classe **[Dao]** est importée ;
- lignes 17-18 : on configure l'objet **[axios]** qui réalise les requêtes HTTP. Cet objet est importé à la ligne 4 ;
  - ligne 17 : définition d'un **[timeout]** de 2 secondes ;
  - ligne 18 : l'URL du serveur de calcul de l'impôt ;
  - ligne 19 : pour pouvoir échanger des cookies avec le serveur ;
- lignes 24-25 : utilisation du plugin **[pluginDao]**
  - ligne 24 : import du plugin ;
  - ligne 25 : intégration du plugin. On voit que le second paramètre de la méthode **[Vue.use]** est la référence de la couche **[dao]** définie ligne 21. C'est pour cette raison que la propriété **[Vue.$dao]** désignera la couche **[dao]** dans toutes les instances de la classe / fonction **[Vue]**, ç-à-d dans tous les composants **[Vue.js]** ;

12.5 La vue principale [App.vue]

Le code de la vue principale **[App]** est le suivant :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <!-- message -->
5.        <b-alert show variant="success" align="center">
6.          <h4>[vuejs-10] : plugin [dao], requêtes HTTP asynchrones</h4>
7.        </b-alert>
8.        <!-- composant faisant une requête asynchrone au serveur de calcul d'impôt-->
9.        <Component1 @error="doSomethingWithError" @endWaiting="endWaiting" @beginWaiting="beginWaiting" />
10.       <!-- affichage d'une éventuelle erreur -->
11.       <b-alert show
12.                variant="danger"
13.                v-if="showError">Evénement [error] intercepté par [App]. Valeur reçue = {{error}}</b-alert>
14.       <!-- message d'attente avec un spinner -->
15.       <b-alert show v-if="showWaiting" variant="light">
16.         <strong>Requête au serveur de calcul d'impôt en cours...</strong>
17.         <b-spinner variant="primary" label="Spinning"></b-spinner>
18.       </b-alert>
19.     </b-card>
20.   </div>
21. </template>
22.
23. <script>
24.   import Component1 from "./components/Component1";
25.   export default {
26.     name: "app",
27.     // état du composant
28.     data() {
29.       return {
30.         // contrôle le spinner d'attente
31.         showWaiting: false,
32.         // contrôle l'affichage de l'erreur
33.         showError: false,
34.         // l'erreur interceptée
35.         error: {}
36.       };
37.     },
38.     // composants utilisés
39.     components: {
40.       Component1
41.     },
42.     // méthodes de gestion des évts
```

```
43.     methods: {
44.       // début attente
45.       beginWaiting() {
46.         // on affiche l'attente
47.         this.showWaiting = true;
48.         // on cache le msg d'erreur
49.         this.showError = false;
50.       },
51.       // fin attente
52.       endWaiting() {
53.         // on cache l'attente
54.         this.showWaiting = false;
55.       },
56.       // gestion d'erreur
57.       doSomethingWithError(error) {
58.         // on note qu'il y a eu erreur
59.         this.error = error;
60.         // on affiche le msg d'erreur
61.         this.showError = true;
62.       }
63.     }
64.   };
65. </script>
```

Commentaires

- ligne 9 : **`[Component1]`** est le composant qui fait la requête HTTP asynchrone. Il peut émettre trois événements :
  - **`[beginWaiting]`** : la requête va être faite. Il faut afficher un message d'attente à destination de l'utilisateur ;
  - **`[endWaiting]`** : la requête est terminée. Il faut arrêter l'attente ;
  - **`[error]`** : la requête s'est mal passée. Il faut afficher un message d'erreur ;
- lignes 10-13 : l'alerte qui affiche l'éventuel message d'erreur. Elle est contrôlée par le booléen **`[showError]`** de la ligne 33. Elle affiche l'erreur de la ligne 35 ;
- lignes 14-18 : l'alerte qui affiche le message d'attente avec un spinner. Elle est contrôlée par le booléen **`[showWaiting]`** de la ligne 47 ;
- lignes 45-50 : **`[beginWaiting]`** est la méthode exécutée à réception de l'événement **`[beginWaiting]`**. Elle affiche le message d'attente (ligne 47) et cache le message d'erreur (ligne 49) au cas où celui-ci serait visible suite à une opération précédente ;
- lignes 52-55 : **`[endWaiting]`** est la méthode exécutée à réception de l'événement **`[endWaiting]`**. Elle cache le message d'attente (ligne 54) ;
- lignes 57-62 : **`[doSomethingWithError]`** est la méthode exécutée à réception de l'événement **`[error]`**. Elle enregistre l'erreur reçue (ligne 59) et affiche le message d'erreur (ligne 61) ;

12.6 Le composant `[Component1]`

Le code du composant **`[Component1]`** est le suivant :

```
1.  <template>
2.    <b-row>
3.      <b-col>
4.        <b-alert show
5.                 variant="warning"
6.                 v-if="showMsg">Valeur reçue du serveur = {{data}}</b-alert>
7.      </b-col>
8.    </b-row>
9.  </template>
10.
11. <script>
12.   export default {
13.     name: "component1",
14.     // état du composant
15.     data() {
16.       return {
17.         showMsg: false
18.       };
19.     },
20.     // méthodes de gestion des évts
21.     methods: {
22.       // traitement de la donnée reçue du serveur
23.       doSomethingWithData(data) {
24.         // on enregistre la donnée reçue
25.         this.data = data;
26.         // on l'affiche
27.         this.showMsg = true;
28.       }
```

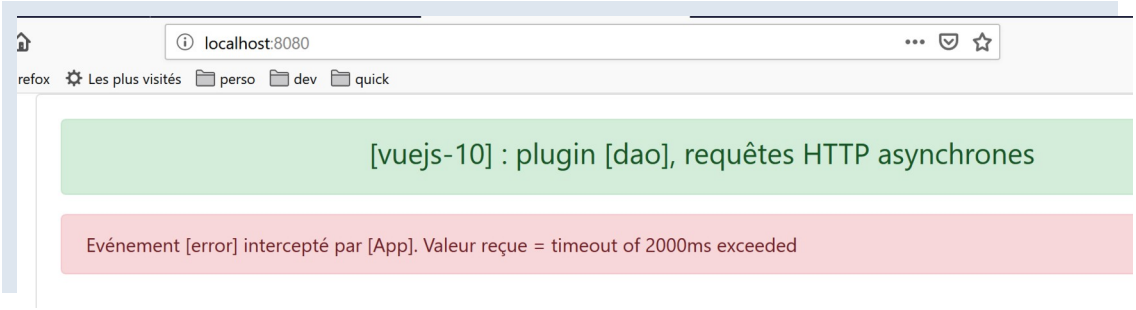
```
29.      },
30.      // le composant vient d'être créé
31.      created() {
32.        // on initialise la session avec le serveur - requête asynchrone
33.        // on utilise la promesse rendue par les méthodes de la couche [dao]
34.        // on signale le début de l'opération
35.        this.$emit("beginWaiting");
36.        // on lance l'opération asynchrone
37.        this.$dao
38.          // il s'agit d'initialiser une session jSON avec le serveur de calcul de l'impôt
39.          .initSession()
40.          // méthode qui traite la donnée reçue en cas de succès
41.          .then(data => {
42.            // on traite la donnée reçue
43.            this.doSomethingWithData(data);
44.          })
45.          // méthode qui traite l'erreur en cas d'erreur
46.          .catch(error => {
47.            // on remonte l'erreur au composant parent
48.            this.$emit("error", error.message);
49.          }).finally(() => {
50.            // fin de l'attente
51.            this.$emit("endWaiting");
52.          })
53.      }
54.    };
55.  </script>
```

Commentaires

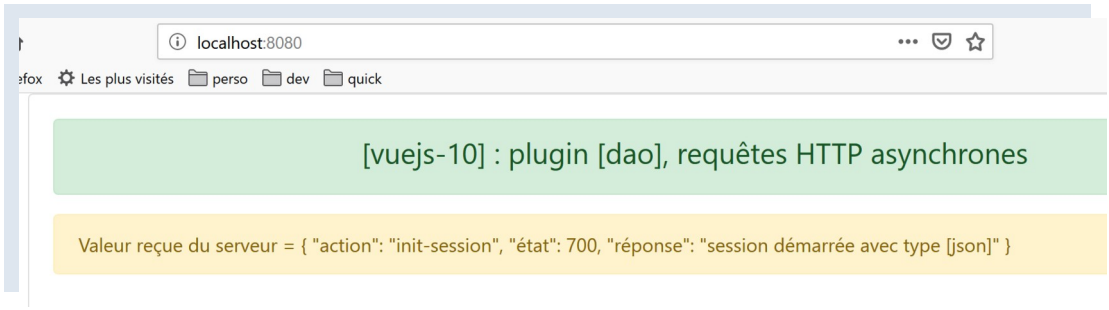
- lignes 4-6 : le composant est constitué d'une unique alerte qui affiche la valeur renvoyée par le serveur de calcul de l'impôt, ceci uniquement en cas de succès de la requête HTTP. Cette alerte est contrôlée par le booléen **[showMsg]** de la ligne 17 ;
- lignes 31-53 : la requête HTTP est faite dès que le composant a été créé. On met donc son code dans la méthode **[created]** de la ligne 31 ;
- ligne 35 : on indique au composant parent que la requête asynchrone va démarrer ;
- lignes 37-39 : la méthode **[this.$dao.initSession]** est exécutée. Elle initialise une session jSON avec le serveur de calcul d'impôt. Le résultat immédiat de cette méthode est une **[Promise]** ;
- lignes 41-44 : ce code s'exécute lorsque le serveur a rendu son résultat sans erreur. Le résultat du serveur est dans **[data]**. Ligne 43, on demande à la méthode **[doSomethingWithData]** de traiter ce résultat ;
- lignes 46-49 : ce code s'exécute en cas d'erreur lors de l'exécution de la requête. Ligne 48, on indique au composant parent qu'une erreur est survenue et on lui passe le message de l'erreur **[error.message]** ;
- lignes 49-52 : ce code s'exécute dans tous les cas. On indique au composant parent que la requête HTTP est terminée ;
- lignes 23-28 : la méthode **[doSomethingWithData]** est la méthode chagée d'exploiter la donnée **[data]** envoyée par le serveur. Ligne 25, on enregistre cette donnée et ligne 27 on l'affiche ;

12.7 Exécution du projet

Si lorsqu'on lance le projet, le serveur de calcul d'impôt n'est pas lancé alors on obtient le résultat suivant :



Lançons le serveur **[Laragon]** (cf https://tahe.developpez.com/tutoriels-cours/php7) et rechargeons la page ci-dessus. Le résultat est alors le suivant :

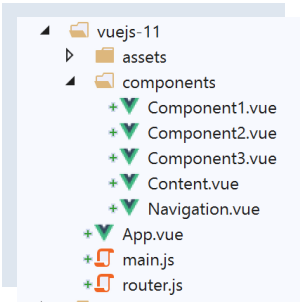


Note : nous utilisons ici la version 14 du serveur de calcul d'impôt définie au https://tahe.developpez.com/tutoriels-cours/php7.

13 projet [vuejs-11] : routage et navigation

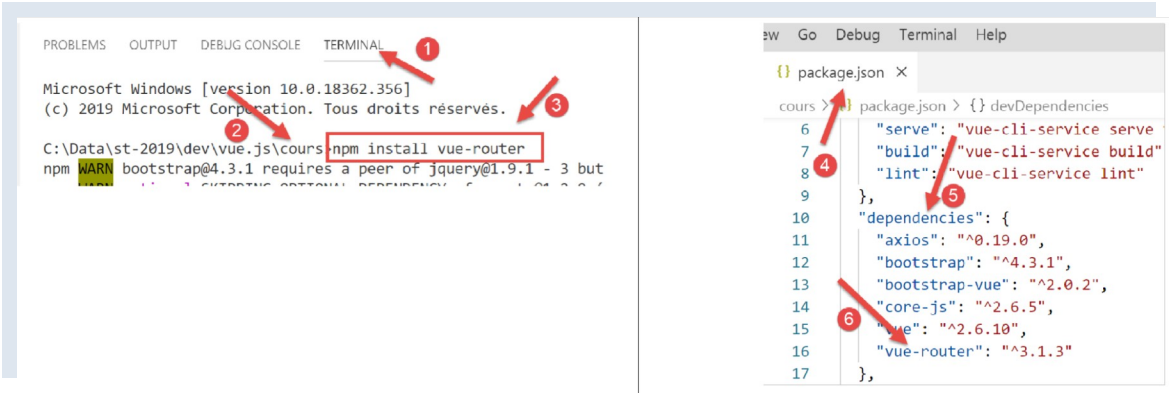
Le routage est ce qui va permettre à l'utilisateur de naviguer entre les différentes pages de l'application.

L'arborescence du projet est la suivante :



13.1 Installation des dépendances

Le routage dans **[Vue.js]** nécessite la dépendance **[vue-router]** :



- en **[1-3]**, on installe la dépendance **[vue-router]** ;
- en **[4-6]**, après installation de la dépendance, le fichier **[package.json]** a été modifié ;

13.2 Le script de routage [router.js]

Les règles de navigation de l'application sont inscrites dans le fichier **[router.js]** (le nom du script peut être quelconque) :

```
1.  // imports nécessaires au routage
2.  import Vue from 'vue'
3.  import VueRouter from 'vue-router'
4.
5.  // plugin de routage
6.  Vue.use(VueRouter)
7.
8.  // les composants cibles du routage
9.  import Component1 from './components/Component1.vue'
10. import Component2 from './components/Component2.vue'
11. import Component3 from './components/Component3.vue'
12.
13.
14. // les routes de l'application
15. const routes = [
16.   // home
17.   { path: '/', name: 'home', component: Component1 },
18.   // Component1
19.   {
20.     path: '/vue1', name: 'vue1', component: Component1
21.   },
22.   {
23.     // Component2
24.     path: '/vue2', name: 'vue2', component: Component2
```

```
25.    },
26.    // Component3
27.    {
28.      path: '/vue3', name: 'vue3', component: Component3
29.    },
30.  ]
31.
32.  // le routeur
33.  const router = new VueRouter({
34.    routes,
35.    mode: 'history',
36.  })
37.
38.  // export du routeur
39.  export default router
```

Commentaires

- le script **[router.js]** va fixer les règles de routage de notre application ;
- ligne 1-6 : activation du plugin **[vue-router]** nécessaire au routage. Cette activation nécessite l'import de la classe / fonction **[Vue]** (ligne 2) et celui du plugin de routage (ligne 3). Ce plugin a été installé avec la dépendance **[router]** que nous venons d'installer ;
- lignes 8-11 : import des vues cibles du routage ;
- lignes 15-21 : définition du tableau des routes. Chaque élément de ce tableau est un objet avec les propriétés suivantes :
  - **[path]** : l'URL de la vue, celle que nous voulons voir affichée dans le champ **[URL]** du navigateur. On est libre de mettre ce qu'on veut ;
  - **[name]** : le nom de la route. Là également on peut mettre ce qu'on veut ;
  - **[component]** : le composant qui affiche la vue. Là c'est forcément un composant existant ;

  On aura donc ici quatre routes **[/, /vue1, /vue2, /vue3]**.
- lignes 33-36 : le routeur est une instance de la classe **[VueRouter]** importée ligne 3. Le constructeur de **[VueRouter]** est ici utilisé avec deux paramètres :
  - les routes de l'application ;
  - un mode d'écriture des URL dans le navigateur : le mode par défaut **[hash]** écrit les URL sous la forme **[localhost:8080/#/vue1]** (# est le hash). Le mode **[history]** enlève le # **[localhost:8080/vue1]** ;
- ligne 39 : on exporte le routeur ;

13.3 Le script principal [main.js]

Le script principal **[main.js]** est 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')
```

Commentaires

- ligne 14 : on importe le routeur exporté par le script **[router.js]** ;

- ligne 25 : ce routeur est passé en paramètre du constructeur de la classe `[Vue]` qui va afficher la vue principale `[App]`, associé à la propriété `[router]` de la vue ;

13.4 La vue principale `[App]`

Le code de la vue principale est le suivant :

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

Commentaires

- ligne 9 : affiche la vue courante du routage. La balise <**router-view**> n'est reconnue que si la propriété `[router]` de la vue a été initialisée ;

13.5 La mise en page des vues

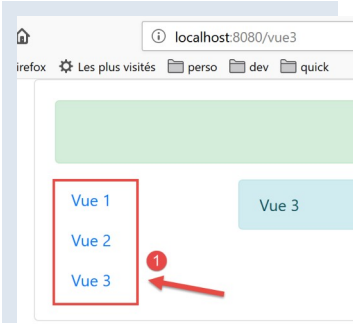
La mise en page des vues est assurée par le composant `[Layout]` suivant :

```
1.  <template>
2.    <!-- ligne -->
3.    <div>
4.      <b-row>
5.        <!-- zone à trois colonnes -->
6.        <b-col cols="2" v-if="left">
7.          <slot name="left" />
8.        </b-col>
9.        <!-- zone à neuf colonnes -->
10.       <b-col cols="10" v-if="right">
11.         <slot name="right" />
12.       </b-col>
13.     </b-row>
14.   </div>
15. </template>
16.
17. <script>
18.   export default {
19.     // paramètres
20.     props: {
21.       left: {
22.         type: Boolean
23.       },
24.       right: {
25.         type: Boolean
26.       }
27.     }
28.   };
29. </script>
```

Nous avons déjà utilisé et expliqué cette mise en page dans le projet `[vuejs-06]` du paragraphe [vuejs-06].

13.6 Le composant de navigation

Le composant `[Navigation]` offre un menu de navigation à l'utilisateur :



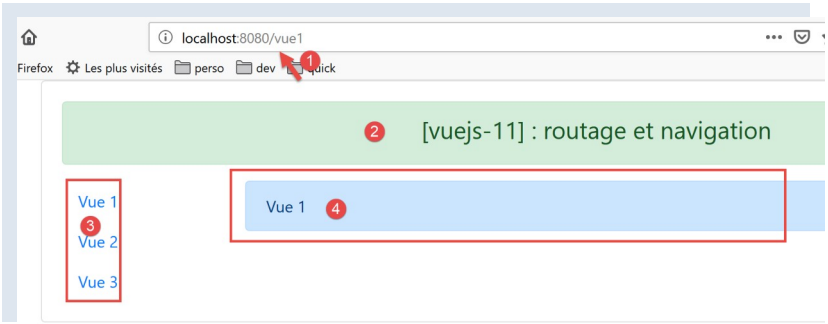
Le composant qui génère le bloc **[1]** est le suivant :

```
1.  <template>
2.    <!-- menu Bootstrap à trois options -->
3.    <b-nav vertical>
4.      <b-nav-item to="/vue1" exact exact-active-class="active">Vue 1</b-nav-item>
5.      <b-nav-item to="/vue2" exact exact-active-class="active">Vue 2</b-nav-item>
6.      <b-nav-item to="/vue3" exact exact-active-class="active">Vue 3</b-nav-item>
7.    </b-nav>
8.  </template>
```

- ce code génère le bloc 1 de trois liens permettant la navigation ;
- l'attribut **[to]** des balises <b-nav-item> doit correspondre à l'une des propriétés **[path]** des routes du routeur de l'application ;

13.7 Les vues

La vue n° 1 est la suivante :



Les zone **[3-4]** sont générées par le composant **[Component1]** suivant :

```
1.  <template>
2.    <Layout :left="true" :right="true">
3.      <!-- navigation -->
4.      <Navigation slot="left" />
5.      <!-- message-->
6.      <b-alert show variant="primary" slot="right">Vue 1</b-alert>
7.    </Layout>
8.  </template>
9.
10. <script>
11.   import Navigation from './Navigation';
12.   import Layout from './Layout';
13.
14.   export default {
15.     name: "component1",
16.     // composants utilisés
17.     components: {
18.       Layout, Navigation
19.     }
20.   };
21. </script>
```

Commentaires

- ligne 2 : la vue n° 1 utilise la mise en page du composant `[Layout]` composée de deux slots appelés `[left, right]` ;
- ligne 4 : le menu de navigation est placé dans le slot de gauche. C'est la zone `[3]` qu'on voit ci-dessus ;
- ligne 6 : un message est placé dans le slot de droite. C'est la zone `[4]` qu'on voit ci-dessus ;

Les vues 2 et 3 sont analogues.

Vue n° 2 affichée par le composant `[Component2]` :

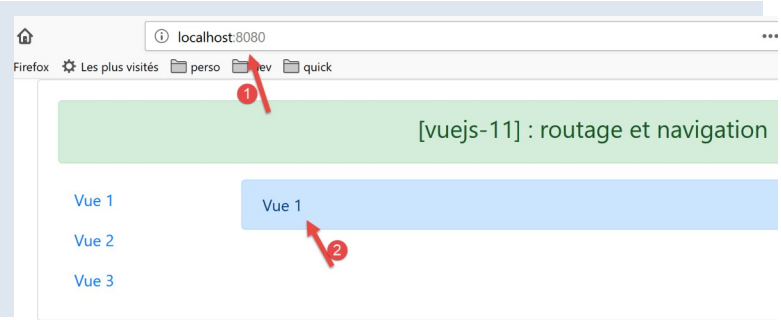
```
1.  <!-- vue n° 2 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-alert show variant="secondary" slot="right">Vue 2</b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12.   import Navigation from './Navigation';
13.   import Layout from './Layout';
14.
15.   export default {
16.     name: "component2",
17.     // composants de la vue
18.     components: {
19.       Layout, Navigation
20.     }
21.   };
22. </script>
```

Vue n° 3 affichée par le composant `[Component3]` :

```
1.  <!-- vue n° 3 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-alert show variant="info" slot="right">Vue 3</b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12.   import Navigation from "./Navigation";
13.   import Layout from "./Layout";
14.
15.   export default {
16.     name: "component3",
17.     // composants de la vue
18.     components: {
19.       Layout,
20.       Navigation
21.     }
22.   };
23. </script>
```

13.8 Exécution du projet

A l'exécution la vue suivante s'affiche :



- en **[1]**, l'URL est **[http://localhost:8080]**. C'est alors la règle de routage suivante qui s'est exécutée :

```
{ path: '/', name: 'home', component: Component1 }
```

C'est donc le composant **[Component1]** qui a été affiché. Il affiche la vue n° 1 **[2]**. Maintenant cliquons sur le lien **[Vue 1]** dont le code est le suivant :

```
<b-nav-item to="/vue1" exact exact-active-class="active">Vue 1</b-nav-item>
```

L'affichage devient le suivant :

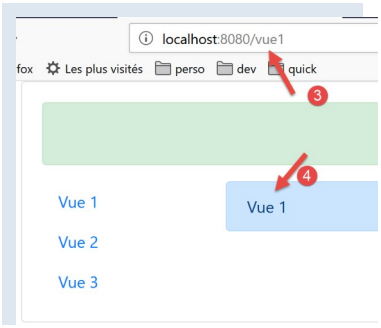
- en **[3]**, la règle suivante de routage s'est exécutée :

```
path: '/vue1', name: 'vue1', component: Component1
```

C'est donc de nouveau le composant **[Component1]** qui a été affiché et donc la vue n° 1 **[4]**. Maintenant cliquons sur le lien **[Vue 2]** dont le code est le suivant :

```
<b-nav-item to="/vue2" exact exact-active-class="active">Vue 2</b-nav-item>
```

La nouvelle vue est alors la suivante :



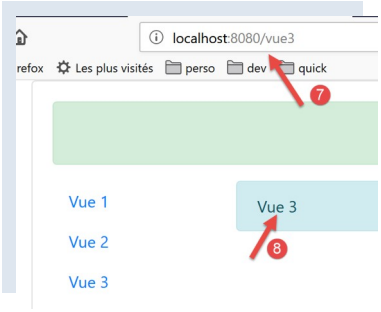
- en **[5]**, la règle suivante de routage s'est exécutée :

```
path: '/vue2', name: 'vue2', component: Component2
```

C'est donc le composant **[Component2]** qui a été affiché et donc la vue n° 2. Si maintenant nous cliquons sur le lien **[Vue 3]**, dont le code est le suivant :

```
<b-nav-item to="/vue3" exact exact-active-class="active">Vue 3</b-nav-item>
```

Nous obtenons la nouvelle vue suivante :



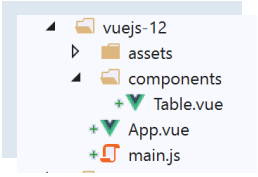
- en **[6]**, la règle de routage suivante s'est exécutée :

```
path: '/vue3', name: 'vue3', component: Component3
```

C'est donc le composant **[Component3]** qui s'est affiché, ç-à-d la vue n° 3 **[8]**.

14 projet [vuejs-12] : gestion des tables HTML

L'arborescence du projet **[vuejs-12]** est la suivante :



14.1 Le script principal [main.js]

Le script principal redevient ce qu'il était dans les premiers projets :

```
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. // configuration
14. Vue.config.productionTip = false
15.
16. // instanciation projet [App]
17. new Vue({
18.   name: "app",
19.   render: h => h(App),
20. }).$mount('#app')
```

14.2 La vue principale [App]

Le code de la vue principale **[App]** est le suivant :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <!-- message -->
5.        <b-alert show variant="success" align="center">
6.          <h4>[vuejs-12] : gestion des tables</h4>
7.        </b-alert>
8.        <!-- composant Table -->
9.        <Table @supprimerLigne="supprimerLigne" :lignes="lignes" @rechargerListe="rechargerListe" />
10.     </b-card>
11.   </div>
12. </template>
13.
14. <script>
15.   import Table from "./components/Table";
16.   export default {
17.     // nom
18.     name: "app",
19.     // composants utilisés
20.     components: {
21.       Table
22.     },
23.     // état interne
24.     data() {
25.       return {
26.         // liste des simulations
27.         lignes: [
28.           { id: 3, marié: "oui", enfants: 2, salaire: 35000, impôt: 1200 },
29.           { id: 5, marié: "non", enfants: 2, salaire: 35000, impôt: 1900 },
30.           { id: 7, marié: "non", enfants: 0, salaire: 30000, impôt: 2000 }
31.         ]
```

```
32.       };
33.     },
34.
35.     // méthodes
36.     methods: {
37.       // suppression d'une ligne
38.       supprimerLigne(index) {
39.         // eslint-disable-next-line
40.         console.log("App supprimerLigne", index);
41.         // suppression ligne à la position [index]
42.         this.lignes.splice(index, 1);
43.       },
44.       // rechargement de la table des lignes
45.       rechargerListe() {
46.         // eslint-disable-next-line
47.         console.log("App rechargerListe");
48.         // on régénère la liste des lignes
49.         this.lignes = [
50.           { id: 3, marié: "oui", enfants: 2, salaire: 35000, impôt: 1200 },
51.           { id: 5, marié: "non", enfants: 2, salaire: 35000, impôt: 1900 },
52.           { id: 7, marié: "non", enfants: 0, salaire: 30000, impôt: 2000 }
53.         ];
54.       }
55.     }
56.   };
57. </script>
```

Commentaires

- la vue principale affiche un composant **[Table]** (lignes 9, 15, 21) ;
- ligne 9 : le composant **[Table]** admet le paramètre **[:lignes]** qui représente les lignes à afficher dans une table HTML. Ces lignes sont définies par les lignes 27-31 du code ;
- ligne 9 : le composant **[Table]**est susceptible d'émettre deux événements :
  - **[supprimerLigne]** : pour supprimer une ligne dont on donne l'index (ligne 38) ;
  - **[rechargerListe]** : pour régénérer la liste des lignes 27-31. En effet, nous allons voir que l'utilisateur peut supprimer certaines des lignes affichées ;
- lignes 38-43 : la méthode chargée de gérer l'événement **[supprimerLigne]**. Elle reçoit en paramètre l'index de la ligne à supprimer ;
- lignes 45-54 : la méthode chargée de gérer l'événement **[rechargerListe]**. En modifiant l'attribut **[lignes]** de la ligne 27, on provoque la mise à jour du composant **[Table]** de la ligne 9, puisqu'il a un paramètre **[:lignes="lignes"]** ;

14.3 Le composant [Table]

Le composant **[Table]** est le suivant :

```
1.  <template>
2.    <div>
3.      <!-- liste vide -->
4.      <template v-if="lignes.length==0">
5.        <b-alert show variant="warning">
6.          <h4>Votre liste de simulations est vide</h4>
7.        </b-alert>
8.        <!-- bouton de rechargement-->
9.        <b-button variant="primary" @click="rechargerListe">Recharger la liste</b-button>
10.     </template>
11.     <!-- liste non vide-->
12.     <template v-if="lignes.length!=0">
13.       <b-alert show variant="primary" v-if="lignes.length==0">
14.         <h4>Liste de vos simulations</h4>
15.       </b-alert>
16.       <!-- tableau des simulations -->
17.       <b-table striped hover responsive :items="lignes" :fields="fields">
18.         <template v-slot:cell(action)="row">
19.           <b-button variant="link" @click="supprimerLigne(row.index)">Supprimer</b-button>
20.         </template>
21.       </b-table>
22.     </template>
23.   </div>
24. </template>
25.
26. <script>
27.   export default {
28.     // propriétés
```

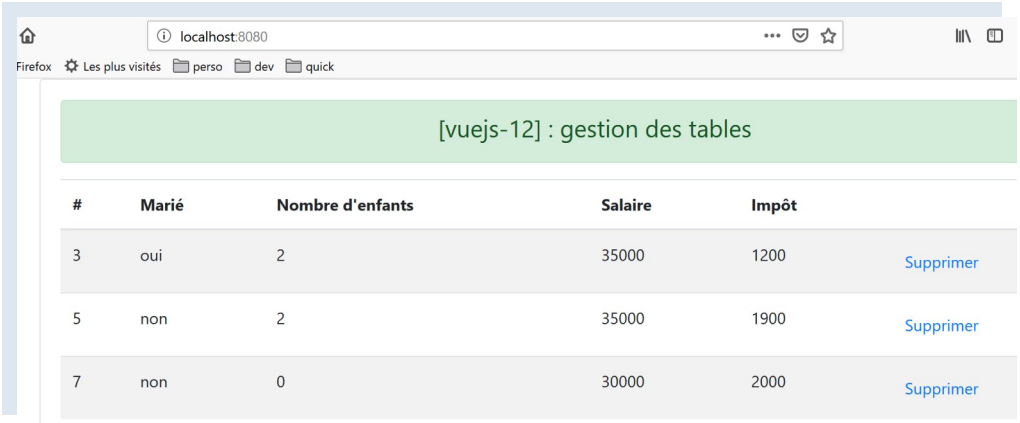
```
29.     props: {
30.       lignes: {
31.         type: Array
32.       }
33.     },
34.     // état interne
35.     data() {
36.       return {
37.         fields: [
38.           { label: "#", key: "id" },
39.           { label: "Marié", key: "marié" },
40.           { label: "Nombre d'enfants", key: "enfants" },
41.           { label: "Salaire", key: "salaire" },
42.           { label: "Impôt", key: "impôt" },
43.           { label: "", key: "action" }
44.         ]
45.       };
46.     },
47.     // méthodes
48.     methods: {
49.       // suppression d'une ligne
50.       supprimerLigne(index) {
51.         // eslint-disable-next-line
52.         console.log("Table supprimerLigne", index);
53.         // on passe l'information au composant parent
54.         this.$emit("supprimerLigne", index);
55.       },
56.       // rechargement de la liste affichée
57.       rechargerListe() {
58.         // eslint-disable-next-line
59.         console.log("Table rechargerListe");
60.         // on émet un événement vers le composant parent
61.         this.$emit("rechargerListe");
62.       }
63.     }
64.   };
65. </script>
```

Commentaires

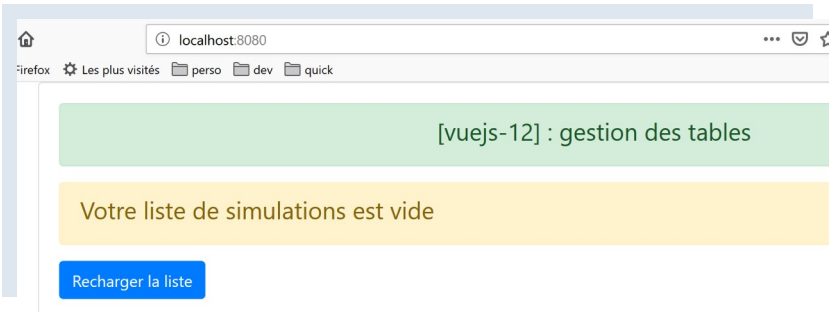
Ce composant a deux états :

1. il affiche une liste dans un tableau HTML ;
2. ou bien un message indiquant que la liste à afficher est vide ;

Le 1er état est affiché si la condition **`[lignes.length!=0]`** est vérifiée (ligne 12) :



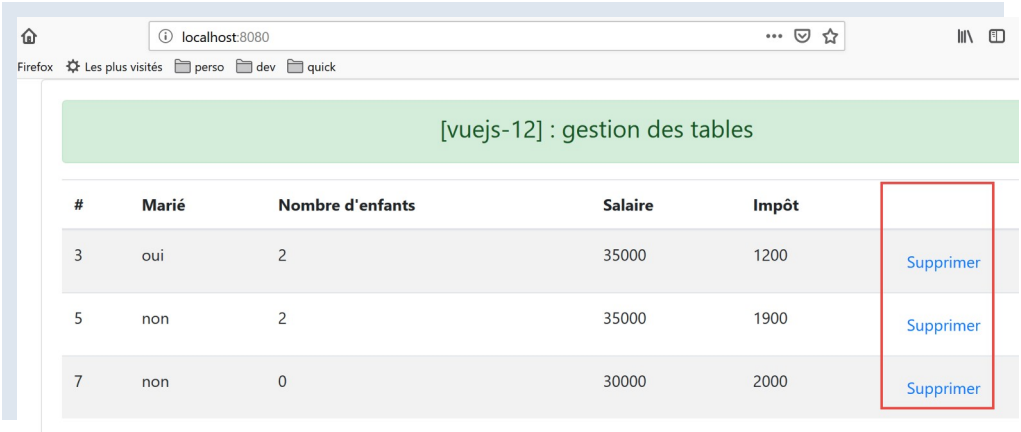
Le second est affiché si la condition **`[lignes.length==0]`** est vérifiée (ligne 4).



- **[lignes]** est un paramètre d'entrée du composant (lignes 29-33) ;
- lignes 4-10 : au lieu d'introduire un bloc de code avec une balise , on a ici utilisé une balise <template>. la différence entre les deux est que la balise <template> n'est pas insérée dans le code HTML généré ;
- ligne 9 : lorsque la liste affichée par la table HTML est vide, un bouton propose de la régénérer. Lorsqu'on clique sur ce bouton, la méthode **[rechargerListe]** des lignes 57-62 est exécutée. Celle-ci se contente d'émettre vers le composant parent l'événement **[rechargerListe]** qui demande au parent de régénérer la liste affichée par la table HTML ;
- lignes 12-22 : le code affiché lorsque la liste à afficher n'est pas vide ;
- ligne 17 : la balise <b-table> est la balise qui génère une table HTML. Les attributs utilisés ici sont les suivants :
  - **[striped]** : la couleur de fond des lignes alterne. Il y a une couleur pour les lignes paires et une autre pour les lignes impaires. Cela améliore la visibilité ;
  - **[hover]** : la ligne sur laquelle on passe la souris change de couleur ;
  - **[responsive]** : la taille de la table s'adapte à l'écran qui l'affiche ;
  - **[:items]** : le tableau des éléments à afficher. Ici le tableau **[lignes]** passé en paramètre au composant (lignes 30-32) ;
  - **[:fields]** : un tableau définissant la mise en page de la table HTML (lignes 37-44) ;
    - chaque élément du tableau **[fields]** définit une colonne de la table HTML ;
    - **[label]** : désigne le titre de la colonne ;
    - **[key]** : désigne le contenu de la colonne ;
- ligne 38 : définit la colonne 0 de la table HTML :
  - **[#]** : est le titre de la colonne ;
  - **[id]** : est son contenu. C'est le champ **[id]** de la ligne affichée qui remplira la colonne 0 ;
- ligne 39 : définit la colonne 1 de la table HTML :
  - **[Marié]** : est le titre de la colonne ;
  - **[marié]** : est son contenu. C'est le champ **[marié]** de la ligne affichée qui remplira la colonne 0 ;
- ligne 43 : définit la dernière colonne de la table HTML :
  - la colonne n'a pas de titre ;
  - son contenu est défini par un champ **[action]**, un champ qui n'existe pas dans les lignes affichées. Cette clé est référencée ligne 18 du **[template]**. La clé est donc utilisée ici uniquement pour identifier une colonne ;
- lignes 18-20 : ce code sert à définir la dernière colonne de la table HTML, celle de clé **[action]** :

```
<template v-slot:cell(action)="row">
```

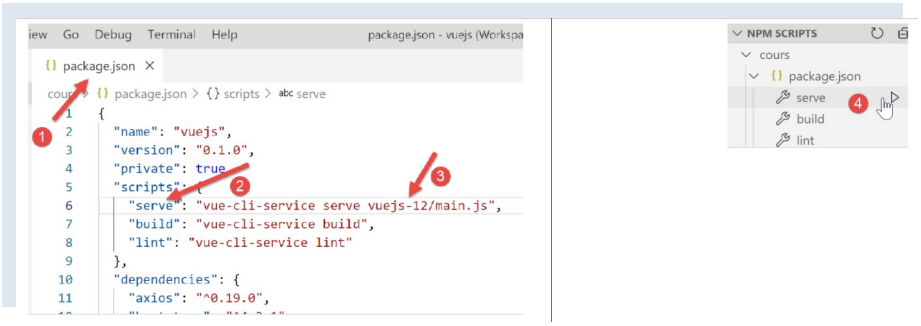
La syntaxe **[v-slot:cell(action)]** désigne la colonne de clé **[action]**. Cette syntaxe permet de définir une colonne lorsque la syntaxe du tableau **[fields]** n'est pas suffisante pour la décrire. Ici nous voulons que la dernière colonne contienne un lien permettant de supprimer une ligne de la table HTML :



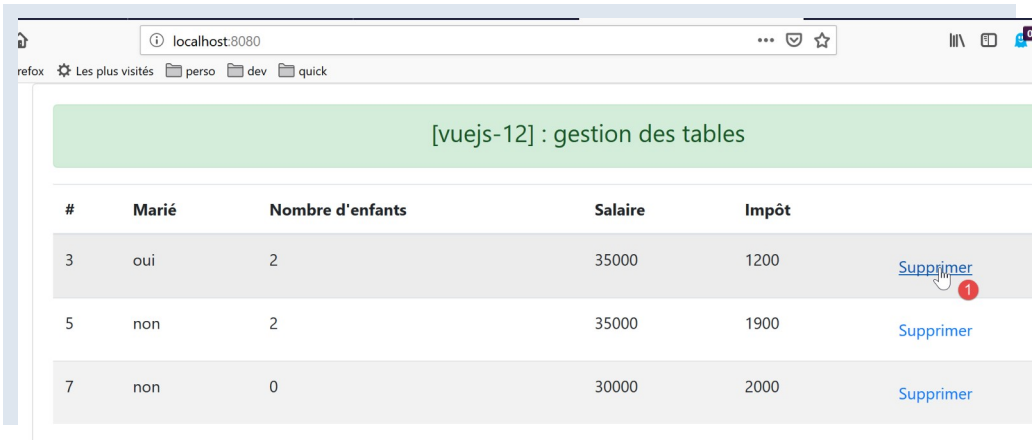
Dans la syntaxe `[<template v-slot:cell(action)="row">]`, le nom `[row]` désigne la ligne de la table. On peut utiliser le nom que l'on veut. On aurait pu écrire `[<template v-slot:cell(action)="ligne">]` ;

- ligne 19 : un bouton <b-button> affiché comme un lien `[variant='link']`. Un clic sur ce lien provoque l'exécution de la méthode `[supprimerLigne(row.index)]`. Ici `[row]` est le nom donné à la ligne de la table HTML, ligne 18 du code ;
- lignes 50-55 : la méthode `[supprimerLigne]` ;
- ligne 54 : on émet l'événement `[supprimerLigne]` vers le composant parent accompagné du n° de la ligne à supprimer ;
- lignes 57-62 : la méthode `[rechargerListe]` ;
- ligne 61 : on émet l'événement `[rechargerListe]` vers le composant parent ;

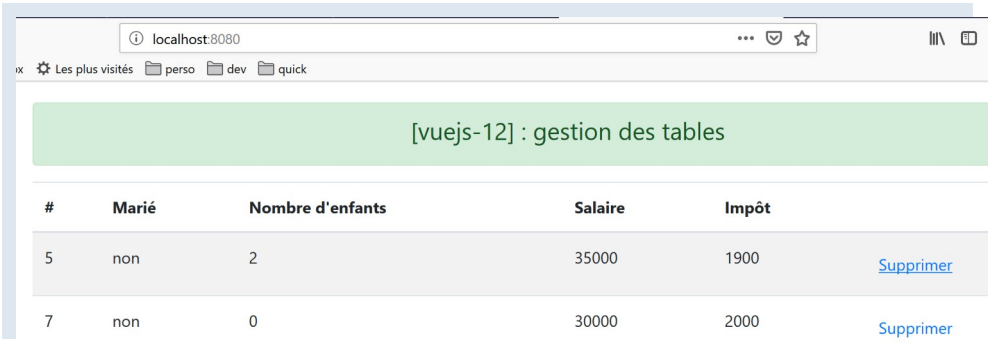
14.4 Exécution du projet



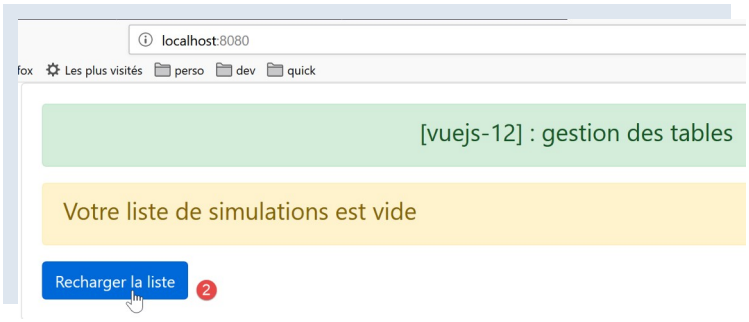
La 1ère vue affichée est la suivante :



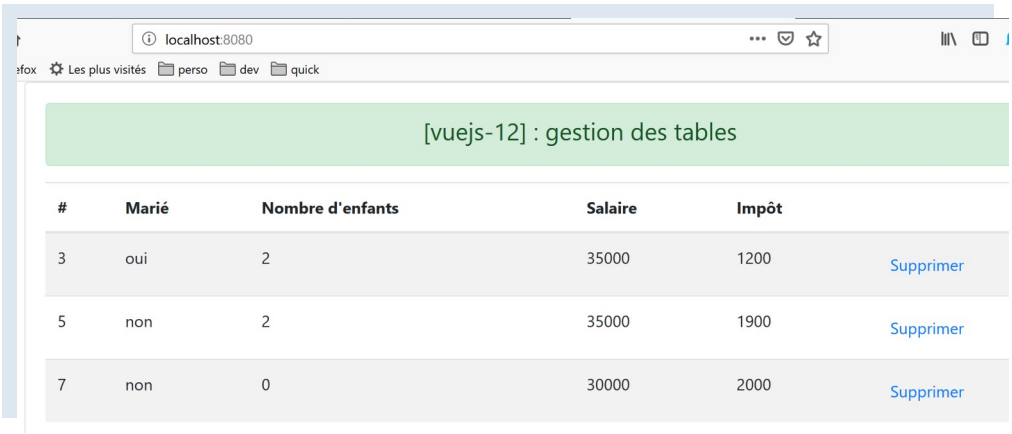
Après avoir supprimé la ligne 1 `[1]`, la vue devient la suivante :



Après avoir supprimé toutes les lignes :



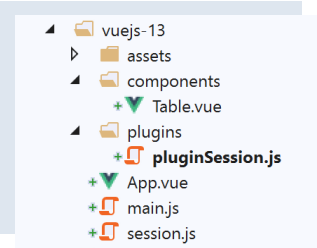
Après avoir cliqué sur le bouton **[2]**, on obtient de nouveau la liste :



15 projet [vuejs-13] : mise à jour d'un composant, utilisation d'une session

Le projet **[vuejs-13]** reprend le projet **[vuejs-12]** en amenant la modification suivante : le tableau affiché par la table HTML est défini dans un objet **[session]** connu de tous les composants. C'est donc une façon de partager de l'information entre composants. Ce concept est directement inspiré de la session web. Nous utilisons la méthode du plugin pour rendre disponible cet objet partagé dans un attribut **[Vue.$session]**.

L'arborescence du projet est la suivante :



15.1 L'objet [session]

L'objet **[session]** partagé par tous les composants est défini dans le script **[./session.js]** :

```
1.  const session = {
2.    // liste des simulations
3.    get lignes() {
4.      return this._lignes;
5.    },
6.    // génération de la liste des simulations
7.    generateLignes() {
8.      this._lignes =
9.        [
10.         { id: 3, marié: "oui", enfants: 2, salaire: 35000, impôt: 1200 },
11.         { id: 5, marié: "non", enfants: 2, salaire: 35000, impôt: 1900 },
12.         { id: 7, marié: "non", enfants: 0, salaire: 30000, impôt: 2000 }
13.       ]
14.   },
15.   // suppression ligne n° index
16.   deleteLigne(index) {
17.     this._lignes.splice(index, 1);
18.   }
19. }
20. // export de l'objet [session]
21. export default session;
```

15.2 Le plugin [./plugins/pluginSession]

Le script **[pluginSession]** est le suivant :

```
1.  export default {
2.    install(Vue, session) {
3.      // ajoute une propriété [$session] à la classe vue
4.      Object.defineProperty(Vue.prototype, '$session', {
5.        // lorsque Vue.$session est référencé, on rend le 2ième paramètre [session] de la méthode [install]
6.        get: () => session,
7.      })
8.    }
9.  }
```

- ligne 4 : l'objet partagé **[session]** sera disponible dans la propriété **[$session]** de tous les composants ;

15.3 Le script principal [main.js]

Le script principal **[main.js]** est 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. // session
14. import session from './session';
15. import pluginSession from './plugins/pluginSession'
16. Vue.use(pluginSession, session)
17.
18. // configuration
19. Vue.config.productionTip = false
20.
21. // instanciation projet [App]
22. new Vue({
23.   name: "app",
24.   render: h => h(App),
25. }).$mount('#app')
```

- lignes 14-16 : le plugin **[pluginSession]** est intégré au framework **[Vue.js]** ;
- après la ligne 16, l'attribut **[$session]** est disponible pour tous les composants ;

15.4 La vue principale [App]

La vue **[App]** est désormais la suivante :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <!-- message -->
5.        <b-alert show variant="success" align="center">
6.          <h4>[vuejs-13] : mise à jour d'un composant, partage des données avec une session</h4>
7.        </b-alert>
8.        <!-- table HTML -->
9.        <Table @updateTable="updateTable" :key="versionTable"/>
10.     </b-card>
11.   </div>
12. </template>
13.
14. <script>
15. import Table from "./components/Table";
16. export default {
17.   // nom
18.   name: "app",
19.   // composants
20.   components: {
21.     Table
22.   },
23.   // état interne
24.   data() {
25.     return {
26.       // version table
27.       versionTable: 1
28.     };
29.   },
30.
31.   // méthodes
32.   methods: {
33.     updateTable() {
34.       // eslint-disable-next-line
35.       console.log("App updateTable");
36.       // incrément version table
37.       this.versionTable++;
38.     }
39.   }
40. };
41. </script>
```

Commentaires

- la vue **[App]** ne gère plus désormais le tableau affiché par le composant **[Table]** de la ligne 9 ;

- ligne 9 : le composant `[Table]` émet l'événement `[updateTable]` qui demande à ce que le composant `[Table]` soit régénéré. Une façon de faire cela est d'utiliser l'attribut `[:key]`. On donne à cet attribut une valeur modifiable. A chaque fois qu'elle est modifiée, le composant `[Table]` est régénéré ;
- ligne 9 : la valeur de l'attribut `[:key]` est l'attribut `[versionTable]` de la ligne 27. La méthode `[updateTable]` (lignes 33-38) est chargée de régénérer le composant `[Table]` de la ligne 9. Pour cela, la méthode incrémente la valeur de l'attribut `[:key]` du composant `[Table]`, ligne 37. Le composant `[Table]` est alors automatiquement régénéré ;

15.5 Le composant `[Table]`

Le composant `[Table]` évolue de la façon suivante :

```
1.  <template>
2.    <div>
3.      <!-- liste vide -->
4.      <template v-if="lignes.length==0">
5.        <b-alert show variant="warning">
6.          <h4>Votre liste de simulations est vide</h4>
7.        </b-alert>
8.        <!-- bouton de rechargement-->
9.        <b-button variant="primary" @click="rechargerListe">Recharger la liste</b-button>
10.     </template>
11.     <!-- liste non vide-->
12.     <template v-if="lignes.length!=0">
13.       <b-alert show variant="primary" v-if="lignes.length==0">
14.         <h4>Liste de vos simulations</h4>
15.       </b-alert>
16.       <!-- tableau des simulations -->
17.       <b-table striped hover responsive :items="lignes" :fields="fields">
18.         <template v-slot:cell(action)="row">
19.           <b-button variant="link" @click="supprimerLigne(row.index)">Supprimer</b-button>
20.         </template>
21.       </b-table>
22.     </template>
23.   </div>
24. </template>
25.
26. <script>
27. export default {
28.   // état calculé
29.   computed: {
30.     lignes() {
31.       return this.$session.lignes;
32.     }
33.   },
34.   // état interne
35.   data() {
36.     return {
37.       fields: [
38.         { label: "#", key: "id" },
39.         { label: "Marié", key: "marié" },
40.         { label: "Nombre d'enfants", key: "enfants" },
41.         { label: "Salaire", key: "salaire" },
42.         { label: "Impôt", key: "impôt" },
43.         { label: "", key: "action" }
44.       ]
45.     };
46.   },
47.   // méthodes
48.   methods: {
49.     supprimerLigne(index) {
50.       // eslint-disable-next-line
51.       console.log("Table supprimerLigne", index);
52.       // on supprime la ligne
53.       this.$session.deleteLigne(index);
54.       // on demande au composant parent de mettre à jour la vue
55.       this.$emit("updateTable");
56.     },
57.     // rechargement de la liste affichée
58.     rechargerListe() {
59.       // eslint-disable-next-line
60.       console.log("Table rechargerListe");
61.       // on régènère la liste des simulations
62.       this.$session.generateLignes();
63.       // on demande au composant parent de mettre à jour la vue
64.       this.$emit("updateTable");
```

```
65.       }
66.     }
67.  };
68.  </script>
```

Commentaires :

- l'attribut `[lignes]` (lignes 4, 12, 17) n'est plus un paramètre fixé par le composant parent mais un attribut calculé du composant `[Table]` (lignes 30-32). `[lignes]` est alors le tableau `[$session.lignes]` (ligne 31) ;
- lignes 49-56 : la méthode `[supprimerLigne]` fait supprimer une ligne du tableau `[$session.lignes]`. Cette suppression ne change pas, par défaut, l'affichage de la table HTML. En effet, les éléments de `[$session]` **ne sont pas réactifs** : leur modification n'est pas répercutée sur les composants qui les utilisent. Pour cette raison, le composant `[Table]` demande à son parent de le régénérer au moyen de l'événement `[updateTable]` (ligne 55). On a vu que le composant parent allait alors incrémenter l'attribut `[:key]` du composant `[Table]` pour forcer sa régénération ;
- lignes 58-65 : la méthode `[rechargerListe]` demande à l'objet `[$session]` de régénérer le tableau `[$session.lignes]`. Pour la même raison que précédemment, cette modification de `[$session.liste]` ne change pas, par défaut, l'affichage de la table HTML. Pour cette raison, le composant `[Table]` demande à son parent de le régénérer au moyen de l'événement `[updateTable]` (ligne 64).

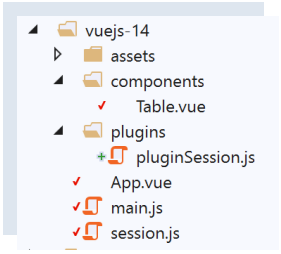
15.6 Exécution du projet

On obtient les mêmes résultats que dans le projet `[vuejs-12]`.

16 projet [vuejs-14] : rendre la session réactive

On a vu que l'objet **[session]** utilisé dans le projet précédent avait des propriétés **non réactives** : si on les modifie, les vues utilisant ces propriétés ne sont pas mise à jour. Il est possible d'avoir un objet [session] réactif si on le stocke dans les données réactives des vues. C'est ce que montre le projet [vuejs-14].

L'arborescence du projet est la suivante :



16.1 L'objet [session]

L'objet **[session]** partagé par tous les composants ne change pas.

16.2 Le plugin [./plugins/pluginSession]

Le script **[pluginSession]** ne change pas. L'objet partagé **[session]** est disponible dans la propriété **[$session]** de tous les composants.

16.3 Le script principal [main.js]

Le script principal **[main.js]** ne change pas.

16.4 La vue principale [App]

La vue **[App]** est désormais la suivante :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <!-- message -->
5.        <b-alert show variant="success" align="center">
6.          <h4>[vuejs-14] : utilisation d'un objet partagé entre composants</h4>
7.        </b-alert>
8.        <!-- table HTML -->
9.        <Table/>
10.     </b-card>
11.   </div>
12. </template>
13.
14. <script>
15. import Table from "./components/Table";
16. export default {
17.   // nom
18.   name: "app",
19.   // composants
20.   components: {
21.     Table
22.   },
23.   // cycle de vie
24.   created() {
25.     // génération du tableau des simulations
26.     this.$session.generateLignes();
27.   }
28. };
29. </script>
```

Commentaires

- ligne 9 : le composant **[Table]** n'émet plus l'événement **[updateTable]** qui demande à ce que le composant **[Table]** soit régénéré. Du coup, la méthode [updateTable] a disparu ;

16.5 Le composant [Table]

Le composant **[Table]** évolue de la façon suivante :

```
1.  <template>
2.    <div>
3.      <!-- liste vide -->
4.      <template v-if="lignes.length==0">
5.        <b-alert show variant="warning">
6.          <h4>Votre liste de simulations est vide</h4>
7.        </b-alert>
8.        <!-- bouton de rechargement-->
9.        <b-button variant="primary" @click="rechargerListe">Recharger la liste</b-button>
10.     </template>
11.     <!-- liste non vide-->
12.     <template v-if="lignes.length!=0">
13.       <b-alert show variant="primary" v-if="lignes.length==0">
14.         <h4>Liste de vos simulations</h4>
15.       </b-alert>
16.       <!-- tableau des simulations -->
17.       <b-table striped hover responsive :items="lignes" :fields="fields">
18.         <template v-slot:cell(action)="row">
19.           <b-button variant="link" @click="supprimerLigne(row.index)">Supprimer</b-button>
20.         </template>
21.       </b-table>
22.     </template>
23.   </div>
24. </template>
25.
26. <script>
27. export default {
28.   // état calculé
29.   computed: {
30.     lignes() {
31.       return this.session.lignes;
32.     }
33.   },
34.   // état interne
35.   data() {
36.     return {
37.       fields: [
38.         { label: "#", key: "id" },
39.         { label: "Marié", key: "marié" },
40.         { label: "Nombre d'enfants", key: "enfants" },
41.         { label: "Salaire", key: "salaire" },
42.         { label: "Impôt", key: "impôt" },
43.         { label: "", key: "action" }
44.       ],
45.       session : {}
46.     };
47.   },
48.   // cycle de vie
49.   created(){
50.     this.session=this.$session
51.   },
52.   // méthodes
53.   methods: {
54.     supprimerLigne(index) {
55.       // eslint-disable-next-line
56.       console.log("Table supprimerLigne", index);
57.       // on supprime la ligne
58.       this.session.deleteLigne(index);
59.     },
60.     // rechargement de la liste affichée
61.     rechargerListe() {
62.       // eslint-disable-next-line
63.       console.log("Table rechargerListe");
64.       // on régènère la liste des simulations
65.       this.session.generateLignes();
66.     }
67.   }
68. };
69. </script>
```

Commentaires :

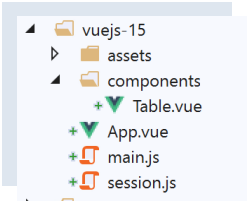
- la nouveauté est lignes 49-51 : lorsque la vue est créée, la session [this.$session] est stockée dans la propriété [session] de la ligne 45. Placée ici, la propriété [session] est réactive ;
- lignes 58 et 65 : au lieu d'utiliser [this.$session] pour ajouter / supprimer une ligne de la table, on utilise la propriété réactive [this.session] ;

16.6 Exécution du projet

On obtient les mêmes résultats que dans le projet **[vuejs-12]**.

17 projet [vuejs-15] : utilisation du plugin [Vuex]

Le projet **[vuejs-15]** reprend le projet **[vuejs-14]** en utilisant un objet **[session]** réactif généré par **[Vuex]**. L'aborescence du projet est la suivante :



17.1 Installation de la dépendance [vuex]

17.2 Le script [./session.js]

L'objet **[session]** devient le suivant :

```
1.  // plugin Vuex
2.  import Vue from 'vue'
3.  import Vuex from 'vuex'
4.  Vue.use(Vuex);
5.
6.  // la session est un store Vuex
7.  const session = new Vuex.Store({
8.    state: {
9.      // le tableau des simulations
10.     lignes: []
11.   },
12.   mutations: {
13.     // génération de la liste des simulations
14.     generateLignes(state) {
15.       // eslint-disable-next-line no-console
16.       console.log("mutation generateLignes");
17.       // on initialise [state.lignes]
18.       state.lignes =
19.         [
20.           { id: 3, marié: "oui", enfants: 2, salaire: 35000, impôt: 1200 },
21.           { id: 5, marié: "non", enfants: 2, salaire: 35000, impôt: 1900 },
22.           { id: 7, marié: "non", enfants: 0, salaire: 30000, impôt: 2000 }
23.         ];
24.       // eslint-disable-next-line no-console
25.       console.log(state.lignes);
26.     },
27.     // suppression ligne n° index
28.     deleteLigne(state, index) {
29.       // eslint-disable-next-line no-console
```

```
30.       console.log("mutation deleteLigne");
31.       // on supprime la ligne n° [index]
32.       state.lignes.splice(index, 1);
33.     }
34.   }
35. });
36. // export de l'objet [session]
1.  export default session;
```

Commentaires

- lignes 2-4 : on intègre le plugin **[Vuex]** au framework **[Vue]** ;
- ligne 7 : la session devient un objet de type **[Vuex.Store]** ;
- lignes 8-11 : la propriété **[state]** contient l'état partagé de l'application **[Vue]**. Cette propriété sera accessible à tous les composants de l'application. Ici nous partageons le tableau des simulations **[lignes]** (ligne 10) ;
- lignes 12-35 : la propriété **[mutations]** rassemble les méthodes qui modifie le contenu de l'objet **[state]** ;
- lignes 14-26 : la propriété **[generateLignes]** est une fonction générant une valeur initiale pour la propriété **[state.lignes]**. Elle admet ici **[state]** comme paramètre. Lignes 18-23 : la propriété **[state.lignes]** est initialisée ;
- lignes 28-35 : la propriété **[deleteLigne]** est une fonction supprimant une ligne du tableau **[state.lignes]**. Elle a pour paramètres :
  - **[state]** qui représente l'objet des lignes 8-11 ;
  - **[index]** qui est le n° de la ligne à supprimer ;
- les fonctions de la propriété **[mutations]** admettent toujours comme 1er paramètre, un objet représentant la propriété **[state]** de la ligne 8. Les paramètres suivants sont fournis par le code appelant la mutation ;
- ligne 37 : l'objet **[session]** est exporté.

Contrairement au projet précédent **[vuejs-13]** nous n'aurons pas ici de plugin pour rendre la session accessible aux composants dans un attribut **[Vue.$session]**.

17.3 Le script principal [./main.js]

Le script principal évolue de la façon suivante :

```
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. // session
14. import session from './session';
15.
16. // configuration
17. Vue.config.productionTip = false
18.
19. // instanciation projet [App]
20. new Vue({
21.   name: "app",
22.   // utilisation store de Vuex
23.   store: session,
24.   render: h => h(App),
25. }).$mount('#app')
```

Commentaires

- ligne 14 : la session est importée ;
- ligne 23 : elle est passée à la vue principale dans un attribut nommé **[store]** (c'est imposé). Grâce au plugin **[Vuex]**, cet attribut devient alors disponible à tous les composants dans un attribut **[Vue.$store]**. On est donc dans une configuration très proche de celle du projet précédent : là où dans un composant on accédait à la session via la notation **[this.$session]**, on y accèdera maintenant via la notation **[this.$store]** ;

17.4 La vue principale [App]

La vue principale **[App]** évolue comme suit :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <!-- message -->
5.        <b-alert show variant="success" align="center">
6.          <h4>[vuejs-14] : utilisation du plugin [Vuex]</h4>
7.        </b-alert>
8.        <!-- table HTML -->
9.        <Table />
10.     </b-card>
11.   </div>
12. </template>
13.
14. <script>
15. import Table from "./components/Table";
16. export default {
17.   // nom
18.   name: "app",
19.   // composants
20.   components: {
21.     Table
22.   },
23.   // cycle de vie
24.   created() {
25.     // génération du tableau des simulations
26.     this.$store.commit("generateLignes");
27.   }
28. };
29. </script>
```

Commentaires

- ligne 9 : la vue `[App]` utilise le composant `[Table]` mais ne reçoit plus d'événements de sa part, ceci grâce au fait que le store `[Vuex]` est **réactif** ;
- lignes 24-27 : la méthode `[created]` est exécutée juste après la création du composant `[App]`. Dans celle-ci, on exécute la mutation nommée `[generateLignes]` qui génère une valeur initiale pour le tableau des simulations. On notera la syntaxe particulière de l'instruction. On rappelle que la notation `[this.$store]` fait référence à la propriété `[store]` de la vue instanciée dans `[main.js]` :

```
1.  // instanciation vue [App]
2.  new Vue({
3.    name: "app",
4.    // utilisation store de Vuex
5.    store: session,
6.    render: h => h(App),
7.  }).$mount('#app')
```

La notation `[this.$store]` désigne donc l'objet `[session]`. On écrit ensuite `[this.$store.commit("generateLignes")]` pour exécuter la mutation s'appelant `[generateLignes]`. Cette mutation est une fonction ;

17.5 Le composant `[Table]`

Le composant `[Table]` évolue de la façon suivante :

```
1.  <template>
2.    <div>
3.      <!-- liste vide -->
4.      <template v-if="lignes.length==0">
5.        <b-alert show variant="warning">
6.          <h4>Votre liste de simulations est vide</h4>
7.        </b-alert>
8.        <!-- bouton de rechargement-->
9.        <b-button variant="primary" @click="rechargerListe">Recharger la liste</b-button>
10.     </template>
11.     <!-- liste non vide-->
12.     <template v-if="lignes.length!=0">
13.       <b-alert show variant="primary" v-if="lignes.length==0">
14.         <h4>Liste de vos simulations</h4>
15.       </b-alert>
16.       <!-- tableau des simulations -->
17.       <b-table striped hover responsive :items="lignes" :fields="fields">
18.         <template v-slot:cell(action)="row">
```

```
19.           <b-button variant="link" @click="supprimerLigne(row.index)">Supprimer</b-button>
20.         </template>
21.       </b-table>
22.     </template>
23.   </div>
24. </template>
25.
26. <script>
27. export default {
28.   // état calculé
29.   computed: {
30.     lignes() {
31.       return this.$store.state.lignes;
32.     }
33.   },
34.   // état interne
35.   data() {
36.     return {
37.       fields: [
38.         { label: "#", key: "id" },
39.         { label: "Marié", key: "marié" },
40.         { label: "Nombre d'enfants", key: "enfants" },
41.         { label: "Salaire", key: "salaire" },
42.         { label: "Impôt", key: "impôt" },
43.         { label: "", key: "action" }
44.       ]
45.     };
46.   },
47.   // méthodes
48.   methods: {
49.     supprimerLigne(index) {
50.       // eslint-disable-next-line
51.       console.log("Table supprimerLigne", index);
52.       // on supprime la ligne
53.       this.$store.commit("deleteLigne", index);
54.     },
55.     // rechargement de la liste affichée
56.     rechargerListe() {
57.       // eslint-disable-next-line
58.       console.log("Table rechargerListe");
59.       // on régènère la liste des simulations
60.       this.$store.commit("generateLignes");
61.     }
62.   }
63. };
64. </script>
```

Commentaires

- le **[template]** des lignes 1-24 ne change pas ;
- lignes 30-32 : la propriété calculée **[lignes]** utilise désormais le **[store]** de **[Vuex]** ;
- lignes 49-54 : pour supprimer une ligne de la table HTML, on utilise la mutation **[deleteLigne]** du **[store]** de **[Vuex]**. On passe en paramètre le n° **[index]** de la ligne à supprimer (ligne 53) ;
- lignes 56-61 : pour recharger la table HTML avec une nouvelle liste, on utilise la mutation **[generateLignes]** du **[store]** de **[Vuex]** ;

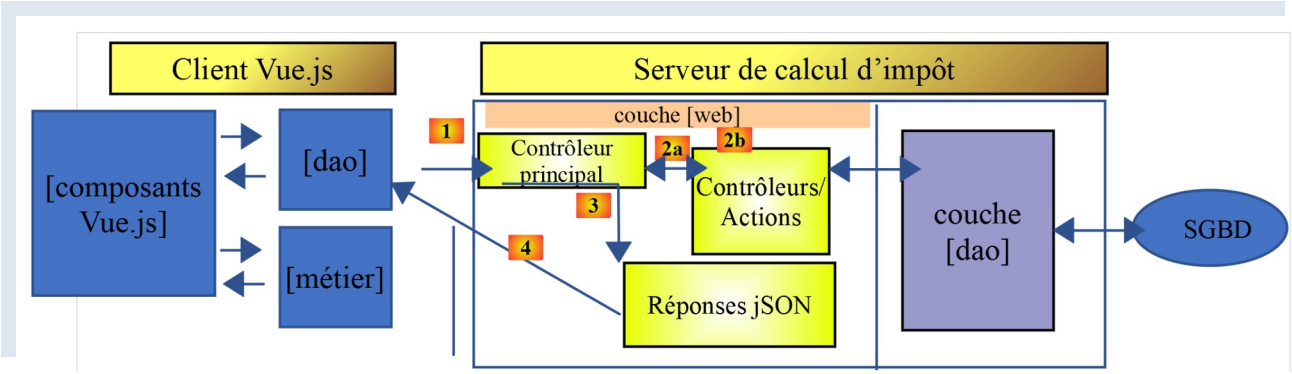
17.6 Conclusion

Les attributs **[Vue.$session]** du projet **[vuejs-13]** et **[Vue.$store]** du projet **[vuejs-15]** sont très proches l'un de l'autre. Ils visent le même objectif : partager de l'information entre vues. L'avantage de l'objet **[store]** est d'être réactif alors que l'objet **[session]** ne l'est pas. Mais le projet [vuejs-14] a montré qu'il était aisé de rendre réactif l'objet [session] en le dupliquant dans les propriétés réactives des vues.

18 Client Vue.js du serveur de calcul de l'impôt

18.1 Architecture

Nous allons implémenter une application client / serveur avec l'architecture suivante :

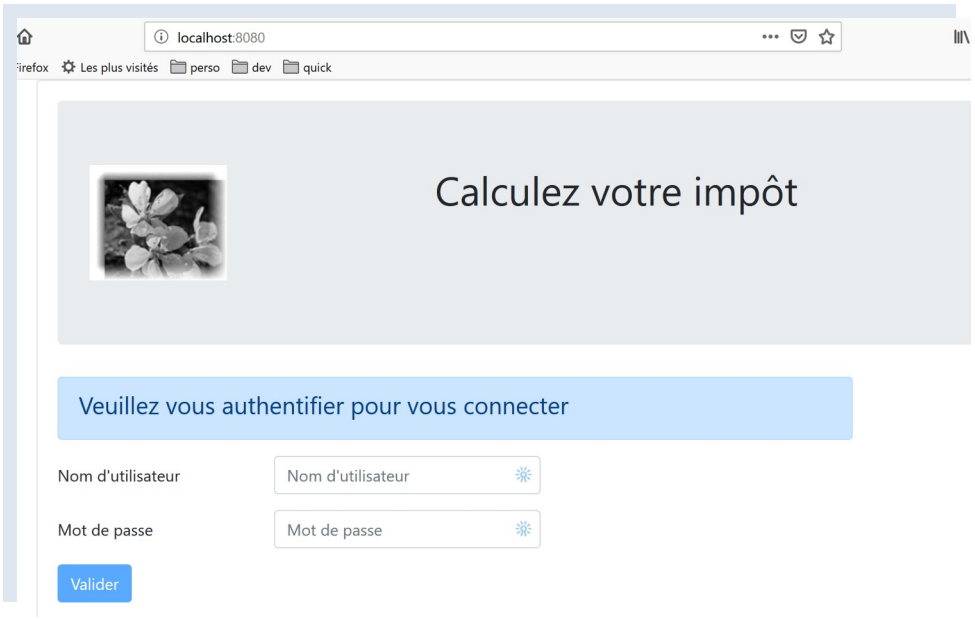


Le serveur de calcul de l'impôt sera la version 14 développée dans le document |https://tahe.developpez.com/tutoriels-cours/php7|

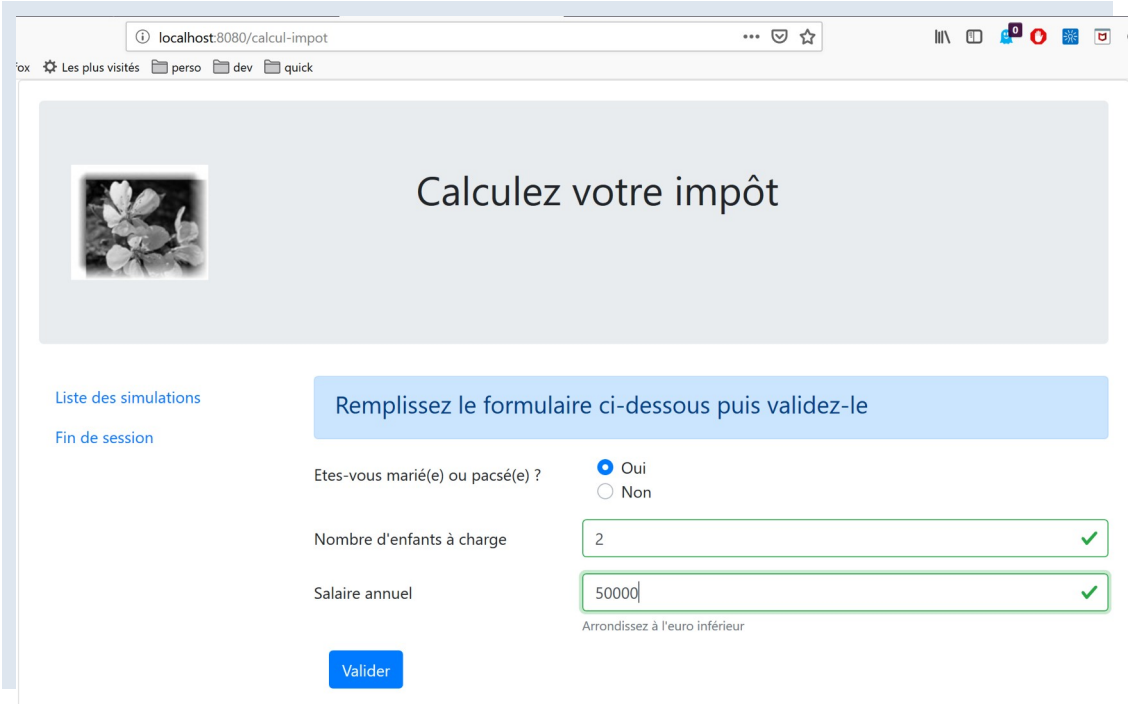
18.2 Les vues de l'application

Les vues de l'application **`[vuejs-10]`** sont celles de la version 13 du document |https://tahe.developpez.com/tutoriels-cours/php7| du serveur de calcul de l'impôt lorsqu'il est utilisé en mode HTML. Mais dans l'application présente, ces vues seront générées par le client Javascript et non par le serveur PHP.

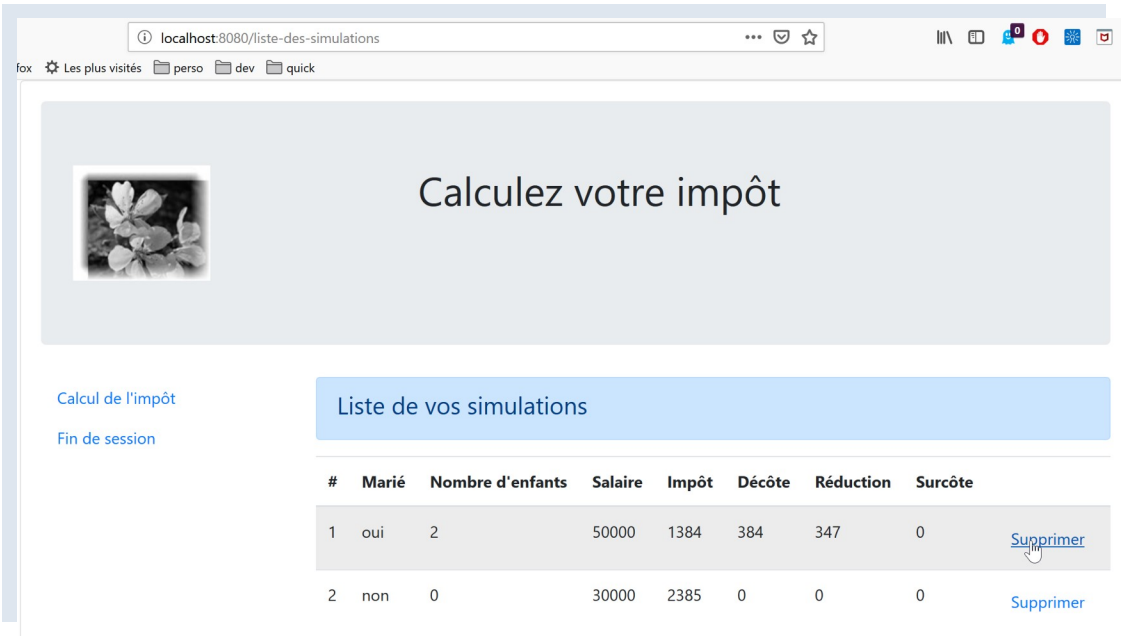
La 1ère vue est la vue d'authentification :



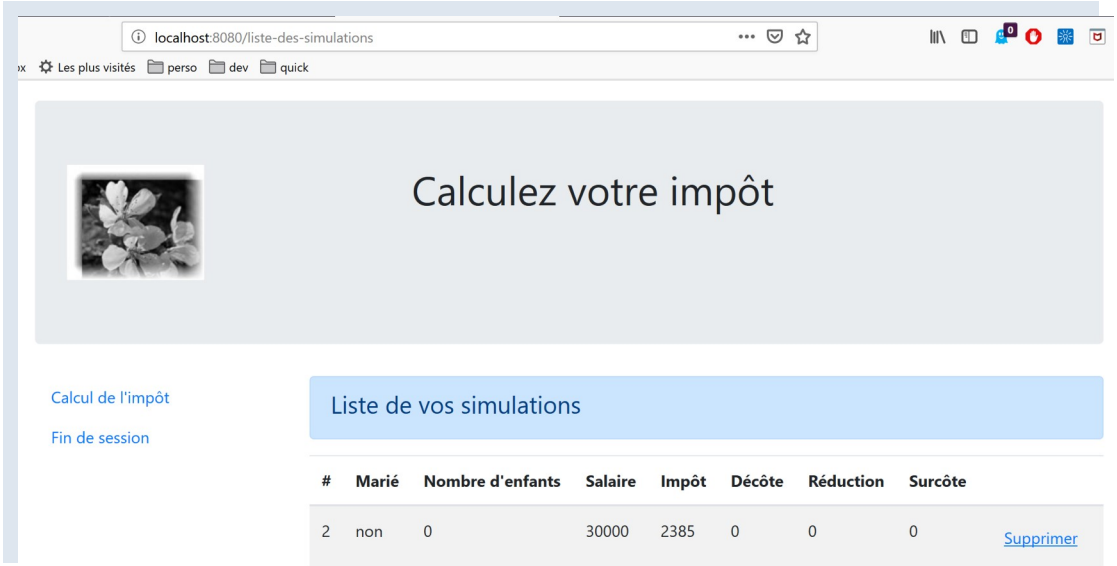
La seconde vue est celle du calcul de l'impôt :



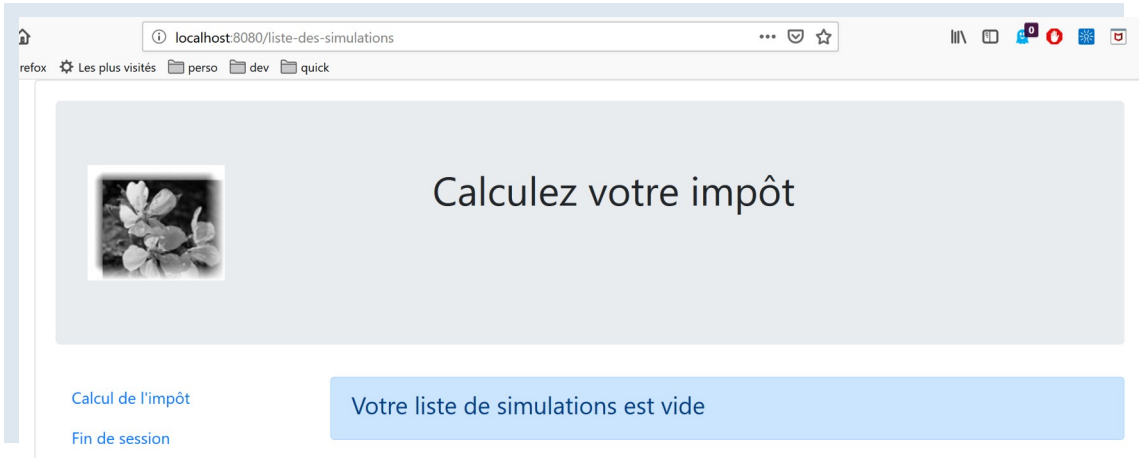
La 3ième vue est celle qui affiche la liste des simulations faites par l'utilisateur :



L'écran ci-dessus montre qu'on peut supprimer la simulation n° 1. On obtient alors la vue suivante :

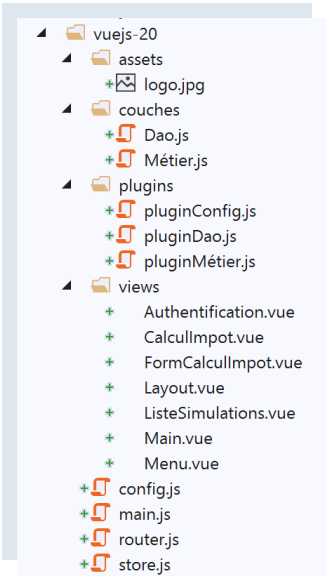


Si on supprime maintenant la dernière simulation, on obtient la nouvelle vue suivante :



18.3 Eléments du projet `[vuejs-20]`

L'arborescence du projet **`[vuejs-20]`** est la suivante :



Les éléments du projet sont les suivants :

- **[assets/logo.jpg]** : le logo du projet ;
- **[couches]** : les couches **[métier]** et **[dao]** de l'application ;
- **[plugins]** : les plugins de l'application ;
- **[views]** : les vues de l'application ;
- **[config.js]** : configure l'application ;
- **[router.js]** : définit le routage de l'application ;
- **[store.js]** : le store de **[Vuex]** ;
- **[main.js]** : le script principal de l'application ;

18.3.1 Les couches [métier] et [dao]

18.3.1.1 La couche [dao]

La couche **[dao]** est implémentée par la classe **[Dao]** du paragraphe |vuejs-10|

18.3.1.2 La couche [métier]

La couche **[métier]** est implémentée par la classe **[Métier]** du document |https://tahe.developpez.com/tutoriels-cours/php7|. On y a ajouté la méthode **[setTaxAdminData]** suivante :

```
1.  // constructeur
2.    constructor(taxAdmindata) {
3.      // this.taxAdminData : données de l'administration fiscale
4.      this.taxAdminData = taxAdmindata;
5.    }
6.
7.    // setter
8.    setTaxAdminData(taxAdmindata) {
9.      // this.taxAdminData : données de l'administration fiscale
10.     this.taxAdminData = taxAdmindata;
11. }
```

La méthode **[setTaxAdminData]** fait la même chose que le constructeur. Sa présence permet la séquence suivante :

1. instancier la classe **[Métier]** avec une instruction **[métier=new Métier()]** lorsqu'on veut instancier la classe mais qu'on n'a pas encore la donnée **[taxAdminData]** ;
2. puis renseigner ultérieurement sa propriété **[taxAdminData]** par une opération **[métier.setTaxAdminData(taxAdmindata)]** ;

18.3.2 Le fichier de configuration [config]

Le fichier **[config.js]** est le suivant :

```
1.  // utilisation de la bibliothèque [axios]
2.  const axios = require('axios');
3.  // timeout des requêtes HTTP
4.  axios.defaults.timeout = 2000;
5.  // la base des URL du serveur de calcul de l'impôt
6.  // le schéma [https] pose des problèmes à Firefox parce que le serveur de calcul
7.  // de l'impôt envoie un certificat autosigné. ok avec Chrome et Edge. Safari pas testé.
8.  axios.defaults.baseURL = 'https://localhost/php7/scripts-web/impots/version-14';
9.  // on va utiliser des cookies
10. axios.defaults.withCredentials = true;
11.
12. // export de la configuration
13. export default {
14.   axios: axios
15. }
```

Cette configuration est celle de la bibliothèque **[axios]** que la couche **[dao]** utilise pour faire ses requêtes HTTP. On notera ligne 8, que le serveur opère sur port sécurisé **[https]**.

18.3.3 Les plugins

Les plugins **[pluginDao, pluginMétier, pluginConfig]** ont pour but de créer trois nouvelles propriétés à la fonction / classe **[Vue]** :

- **[$dao]** : aura pour valeur une instance de la classe **[Dao]** ;

- **[$métier]** : aura pour valeur une instance de la classe **[Métier]** ;
- **[$config]** : aura pour valeur l'objet exporté par le fichier de configuration **[config]** ;

[pluginDao]

```
1.  export default {
2.    install(Vue, dao) {
3.      // ajoute une propriété [$dao] à la classe Vue
4.      Object.defineProperty(Vue.prototype, '$dao', {
5.        // lorsque Vue.$dao est référencé, on rend le 2ième paramètre [dao]
6.        get: () => dao,
7.      })
8.    }
9.  }
10.
11. [pluginMétier]
12.
13. export default {
14.   install(Vue, métier) {
15.     // ajoute une propriété [$métier] à la classe Vue
16.     Object.defineProperty(Vue.prototype, '$métier', {
17.       // lorsque Vue.$métier est référencé, on rend le 2ième paramètre [métier]
18.       get: () => métier,
19.     })
20.   }
21. }
```

[pluginConfig]

```
1.  export default {
2.    install(Vue, config) {
3.      // ajoute une propriété [$config] à la classe vue
4.      Object.defineProperty(Vue.prototype, '$config', {
5.        // lorsque Vue.$config est référencé, on rend le 2ième paramètre [config]
6.        get: () => config,
7.      })
8.    }
9.  }
```

18.3.4 Le store [Vuex]

Le store de **[Vuex]** est implémenté par le fichier **[store]** suivant :

```
1.  // plugin Vuex
2.  import Vue from 'vue'
3.  import Vuex from 'vuex'
4.  Vue.use(Vuex);
5.
6.  // store Vuex
7.  const store = new Vuex.Store({
8.    state: {
9.      // le tableau des simulations
10.     simulations: [],
11.     // le n° de la dernière simulation
12.     idSimulation: 0
13.   },
14.   mutations: {
15.     // suppression ligne n° index
16.     deleteSimulation(state, index) {
17.       // eslint-disable-next-line no-console
18.       console.log("mutation deleteSimulation");
19.       // on supprime la ligne n° [index]
20.       state.simulations.splice(index, 1);
21.       // eslint-disable-next-line no-console
22.       console.log("store simulations", state.simulations);
23.     },
24.     // ajout d'une simulation
25.     addSimulation(state, simulation) {
26.       // eslint-disable-next-line no-console
27.       console.log("mutation addSimulation");
28.       // n° de la simulation
29.       state.idSimulation++;
30.       simulation.id = state.idSimulation;
31.       // on ajoute la simulation au tableau des simulations
32.       state.simulations.push(simulation);
33.     },
```

```
34.     // nettoyage state
35.     clear(state) {
36.       state.simulations = [];
37.       state.idSimulation = 1;
38.     }
39.   }
40. });
41. // export de l'objet [store]
42. export default store;
```

Commentaires

- lignes 2-4 : le plugin **[Vuex]** est intégré au framework **[Vue]** ;
- lignes 8-13 : nous mettons dans le store de **[Vuex]** les éléments suivants :
  - **[simulations]** : la liste des simulations faites par l'utilisateur ;
  - **[idSimulation]** : le n° de la dernière simulation faite par l'utilisateur ;

  On rappelle que le store va être partagé entre les vues et que son contenu est réactif : lorsqu'il est modifié, les vues qui l'utilisent sont automatiquement mises à jour. Dans notre application, seul l'élément **[simulations]** a besoin d'être réactif, pas l'élément **[idSimulation]**. On a laissé cet élément dans le store par commodité ;
- lignes 14-40 : les mutations autorisées sur l'objet **[state]** des lignes 8-13. On rappelle que celles-ci reçoivent toujours l'objet **[state]** des lignes 8-13 en 1er paramètre ;
  - ligne 16 : la mutation **[deleteSimulation]** permet de supprimer une simulation dont on donne le n° **[index]** ;
  - ligne 25 : la mutation **[addSimulation]** permet d'ajouter une nouvelle simulation au tableau des simulations ;
  - ligne 35 : la mutation **[clear]** permet de réinitialiser l'objet **[state]** des lignes 8-13 ;

18.3.5 Le fichier de routage [router]

Le fichier de routage est le suivant :

```
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.
9.  // plugin de routage
10. Vue.use(VueRouter)
11.
12. // les routes de l'application
13. const routes = [
14.   // authentification
15.   {
16.     path: '/', name: 'authentification', component: Authentification
17.   },
18.   // calcul de l'impôt
19.   {
20.     path: '/calcul-impot', name: 'calculImpot', component: CalculImpot
21.   },
22.   // liste des simulations
23.   {
24.     path: '/liste-des-simulations', name: 'listeSimulations', component: ListeSimulations
25.   },
26.   // fin de session
27.   {
28.     path: '/fin-session', name: 'finSession', component: Authentification
29.   }
30. ]
31.
32. // le routeur
33. const router = new VueRouter({
34.   // les routes
35.   routes,
36.   // le mode d'affichage des routes dans le navigateur
37.   mode: 'history',
38. })
39.
40. // export du router
41. export default router
```

Commentaires

- ligne 16 : au démarrage de l'application, c'est la vue **[Authentification]** qui est affichée car son URL est la racine **[/]** ;
- ligne 20 : la vue **[CalculImpot]** est affichée lorsque l'URL **[/calcul-impot]** est demandée ;
- ligne 24 : la vue **[ListeSimulations]** est affichée lorsque l'URL **[/liste-des-simualtions]** est demandée ;
- ligne 28 : la vue **[Authentification]** est affichée lorsque l'URL **[/fin-session]** est demandée ;
- lignes 33-38 : un objet **[router]** est créé avec ces routes (ligne 35) et le mode **[history]** (ligne 37) de gestion des URL ;
- ligne 41 : ce routeur est exporté ;

18.3.6 Le script principal [main.js]

Le script **[main.js]** est le suivant :

```
1.  // imports
2.  import Vue from 'vue'
3.
4.  // vue principale
5.  import Main from './views/Main.vue'
6.
7.  // plugin [bootstrap-vue]
8.  import BootstrapVue from 'bootstrap-vue'
9.  Vue.use(BootstrapVue);
10.
11. // CSS bootstrap
12. import 'bootstrap/dist/css/bootstrap.css'
13. import 'bootstrap-vue/dist/bootstrap-vue.css'
14.
15. // routeur
16. import router from './router'
17.
18. // plugin [config]
19. import config from './config';
20. import pluginConfig from './plugins/pluginConfig'
21. Vue.use(pluginConfig, config)
22.
23. // instanciation couche [dao]
24. import Dao from './couches/Dao';
25. const dao = new Dao(config.axios);
26.
27. // plugin [dao]
28. import pluginDao from './plugins/pluginDao'
29. Vue.use(pluginDao, dao)
30.
31. // instanciation couche [métier]
32. import Métier from './couches/Métier';
33. const métier = new Métier();
34.
35. // plugin [métier]
36. import pluginMétier from './plugins/pluginMétier'
37. Vue.use(pluginMétier, métier)
38.
39. // store Vuex
40. import store from './store'
41.
42. // démarrage de l'UI
43. new Vue({
44.   el: '#app',
45.   // le routeur
46.   router: router,
47.   // le store Vuex
48.   store: store,
49.   // la vue principale
50.   render: h => h(Main),
51. })
```

On notera les points suivants :

- lignes 18-21, l'objet exporté par le script **[./config]** va être disponible dans l'attribut **[Vue.$config]** donc disponible à toutes les vues de l'application. C'était inutile ici car l'objet **[config]** n'est utilisé que par le script **[main]** (ligne 25). Néanmoins il est fréquent que la configuration soit nécessaire à plusieurs vues. On a donc voulu ici garder le principe de la rendre disponible dans un attribut de la vue ;
- lignes 24-25 : instanciation de la couche **[dao]**. La classe **[Dao]** est importée ligne 24 puis instanciée ligne 25. Son constructeur admet pour unique paramètre l'objet **[axios]**, propriété de configuration ;
- lignes 27-29 : la couche **[dao]** est rendue disponible dans l'attribut **[$dao]** de toutes les vues ;

- lignes 31-37 : on répète la même séquence pour la couche **[métier]**. Le constructeur de la classe **[Métier]** a pour paramètre **[taxAdminData]** qui représente les données de l'administration fiscale. Nous n'avons pas encore cette donnée. L'objet **[métier]** de la ligne 33 devra donc être complété ultérieurement ;
- ligne 40 : on importe le store **[Vuex]** ;
- lignes 43-51 : on instancie la vue principale **[Main]** (lignes 5 et 50), en lui passant deux paramètres :
  - ligne 46 : le routeur **[router]** défini ligne 16 ;
  - ligne 48 : le store **[Vuex] [store]** défini ligne 40 ;
  - dans les deux cas, le nom de la propriété est à gauche et sa valeur à droite. Les noms des propriétés **[router, store]** sont fixés par les frameworks **[vue-router]** et **[vuex]**. Les valeurs associées peuvent elles être quelconques ;

18.4 Les vues de l'application

18.4.1 La vue principale [Main]

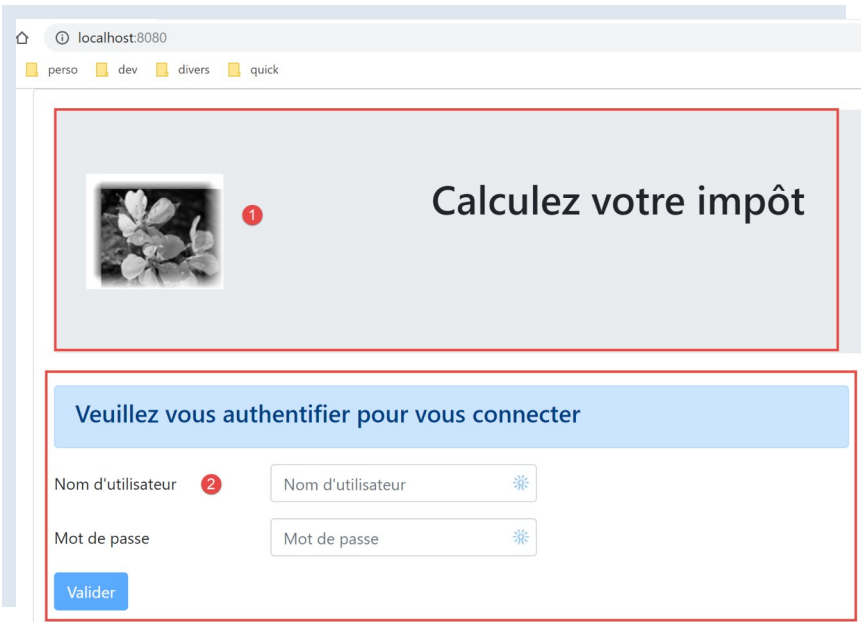
Le code de la vue principale **[Main]** est le suivant :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <div class="container">
4.      <b-card>
5.        <!-- jumbotron -->
6.        <b-jumbotron>
7.          <b-row>
8.            <b-col cols="4">
9.              <img src="../assets/logo.jpg" alt="Cerisier en fleurs" />
10.           </b-col>
11.           <b-col cols="8">
12.             <h1>Calculez votre impôt</h1>
13.           </b-col>
14.         </b-row>
15.       </b-jumbotron>
16.       <!-- erreur requête HTTP -->
17.       <b-alert
18.         show
19.         variant="danger"
20.         v-if="showError"
21.       >L'erreur suivante s'est produite : {{error.message}}</b-alert>
22.       <!-- vue courante -->
23.       <router-view v-if="showView" @loading="mShowLoading" @error="mShowError" />
24.       <!-- loading -->
25.       <b-alert show v-if="showLoading" variant="light">
26.         <strong>Requête au serveur de calcul d'impôt en cours...</strong>
27.         <div class="spinner-border ml-auto" role="status" aria-hidden="true"></div>
28.       </b-alert>
29.     </b-card>
30.   </div>
31. </template>
32.
33. <script>
34. export default {
35.   // nom
36.   name: "app",
37.   // état interne
38.   data() {
39.     return {
40.       // contrôle l'alerte d'attente
41.       showLoading: false,
42.       // contrôle l'alerte d'erreur
43.       showError: false,
44.       // contrôle l'affichage de la vue de routage courante
45.       showView: true,
46.       // un message d'erreur
47.       error: ""
48.     };
49.   },
50.   // gestionnaires d'évts
51.   methods: {
52.     // erreur requête asynchrone
53.     mShowError(error) {
54.       // eslint-disable-next-line
55.       console.log("Main evt error");
56.       // on affiche le msg d'erreur
57.       this.error = error;
58.       this.showError = true;
```

```
59.         // on cache la vue routée
60.         this.showView = false;
61.         // on cache le message d'attente
62.         this.showLoading = false;
63.       },
64.       // affichage ou pas d'une icône d'attente
65.       mShowLoading(value) {
66.         // eslint-disable-next-line
67.         console.log("Main evt showLoading");
68.         // on affiche ou pas l'alerte d'attente
69.         this.showLoading = value;
70.       }
71.     }
72. };
73. </script>
```

Commentaires

- la vue **[Main]** assure une mise en page de la vue routée et affichée ligne 23 :



- les lignes 5-15 affichent la zone 1 ;
- le ligne 23 affiche la vue routée **[2]** ;
- lignes 16-19 : une alerte affichée seulement en cas d'erreur de communication avec le serveur de calcul de l'impôt ;
- lignes 25-28 : un message d'attente affiché à chaque requête HTTP faite au serveur ;
- toutes les vues vont être affichées avec cette mise en page puisque chaque vue routée est affichée par les lignes 20-24. La vue **[Main]** sert à factoriser ce qui peut être partagé par les différentes vues ;
- ligne 23 : chaque vue routée peut émettre trois événements :
  - **[loading]** : une requête HTTP a été lancée. Il faut montrer le message d'attente de la réponse ;
  - **[error]** : la requête HTTP s'est terminée sur une erreur. Il faut montrer le message d'erreur et cacher la vue routée ;
- lignes 38-49 : l'état de la vue :
  - ligne 41 : **[showLoading]** contrôle l'affichage du message d'attente de la fin d'une requête HTTP (ligne 25) ;
  - ligne 43 : **[showError]** contrôle l'affichage du message d'erreur d'une requête HTTP (lignes 17-21) ;
  - ligne 45 : **[showView]** contrôle l'affichage de la vue routée (ligne 23) ;
- lignes 53-63 : la méthode **[mShowError]** gère l'événement **[error]** émis par la vue routée (ligne 23) ;
- lignes 65-70 : la méthode **[mShowLoading]** gère l'événement **[loading]** émis par la vue routée (ligne 23) ;
- ligne 23 : on prêtera attention aux événements **[error]** et **[loading]**. Ils ne sont interceptés que si la vue routée est affichée **[showView=true]**. C'est pourquoi la vue routée est au départ affichée (ligne 45). Elle n'est cachée qu'en cas d'erreur (ligne 60). Pour éviter ce problème on aurait pu utiliser la directive **[v-show]** au lieu de **[v-if]**. la différence entre ces deux directives est la suivante :
  - **[v-if='false']** cache le bloc contrôlé en l'éliminant du code HTML global. Les événements de la vue routée ne peuvent plus alors être interceptés ;
  - **[v-show='false']** cache le bloc contrôlé en jouant sur son CSS, mais le code du bloc reste présent dans le HTML global et peut ainsi intercepter les événements de la vue routée ;

18.4.2 La vue de mise en page [Layout]

Le code de la vue **[Layout]** est le suivant :

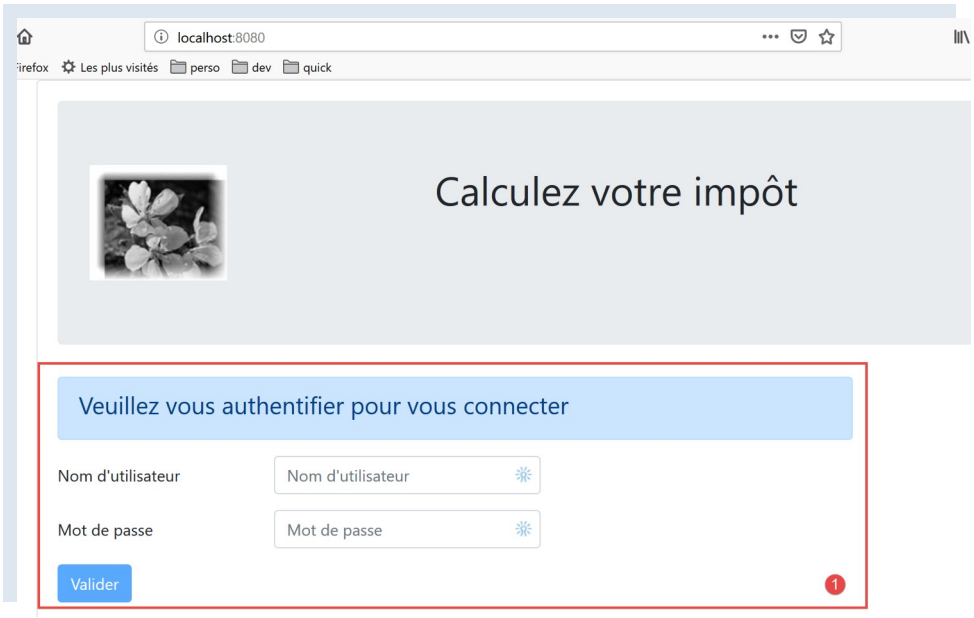
```
1.  <!-- définition HTML de la mise en page de la vue routée -->
2.  <template>
3.    <!-- ligne -->
4.    <div>
5.      <b-row>
6.        <!-- zone de trois colonnes à gauche -->
7.        <b-col cols="3" v-if="left">
8.          <slot name="left" />
9.        </b-col>
10.       <!-- zone de neuf colonnes à droite -->
11.       <b-col cols="9" v-if="right">
12.         <slot name="right" />
13.       </b-col>
14.     </b-row>
15.   </div>
16. </template>
17.
18. <script>
19.   export default {
20.     // paramètres de la vue
21.     props: {
22.       // contrôle la colonne de gauche
23.       left: {
24.         type: Boolean
25.       },
26.       // contrôle la colonne de droite
27.       right: {
28.         type: Boolean
29.       }
30.     }
31.   };
32. </script>
```

Commentaires

- la vue **[Layout]** permet de diviser la vue routée en deux zones :
  - une zone de 3 colonnes Bootstrap à gauche (lignes 7-9). Cette zone accueillera le menu de navigation lorsqu'il y en a un ;
  - une zone de 9 colonnes à droite (lignes 11-13). Cette zone accueillera l'information amenée par la vue routée ;

18.4.3 La vue [Authentification]

La vue d'authentification est la suivante :



Cette vue est obtenue à partir du **[Layout]** en supprimant la colonne de gauche pour n'afficher que la colonne de droite.

Son code est le suivant :

```
<!-- définition HTML de la vue -->
<template>
  <Layout :left="false" :right="true">
    <template slot="right">
      <!-- formulaire HTML - on poste ses valeurs avec l'action [authentifier-utilisateur] -->
      <b-form @submit.prevent="login">
        <!-- titre -->
        <b-alert show variant="primary">
          <h4>Bienvenue. Veuillez vous authentifier pour vous connecter</h4>
        </b-alert>
        <!-- 1ère ligne -->
        <b-form-group label="Nom d'utilisateur" label-for="user" label-cols="3">
          <!-- zone de saisie user -->
          <b-col cols="6">
            <b-form-input type="text" id="user" placeholder="Nom d'utilisateur" v-model="user" />
          </b-col>
        </b-form-group>
        <!-- 2ième ligne -->
        <b-form-group label="Mot de passe" label-for="password" label-cols="3">
          <!-- zone de saisie password -->
          <b-col cols="6">
            <b-input type="password" id="password" placeholder="Mot de passe" v-model="password" />
          </b-col>
        </b-form-group>
        <!-- 3ième ligne -->
        <b-alert
          show
          variant="danger"
          v-if="showError"
          class="mt-3"
        >L'erreur suivante s'est produite : {{message}}</b-alert>
        <!-- bouton de type [submit] sur une 3ième ligne -->
        <b-row>
          <b-col cols="2">
            <b-button variant="primary" type="submit" :disabled="!valid">Valider</b-button>
          </b-col>
        </b-row>
      </b-form>
    </template>
  </Layout>
</template>

<!-- dynamique de la vue -->
<script>
import Layout from "./Layout";
export default {
  // état du composant
  data() {
    return {
      // utilisateur
      user: "",
      // son mot de passe
      password: "",
      // contrôle l'affichage d'un msg d'erreur
      showError: false,
      // le message d'erreur
      message: "",
      // session démarrée
      sessionStarted: false
    };
  },

  // composants utilisés
  components: {
    Layout
  },

  // propriétés calculées
  computed: {
    // saisies valides
    valid() {
      return this.user && this.password && this.sessionStarted;
    }
```

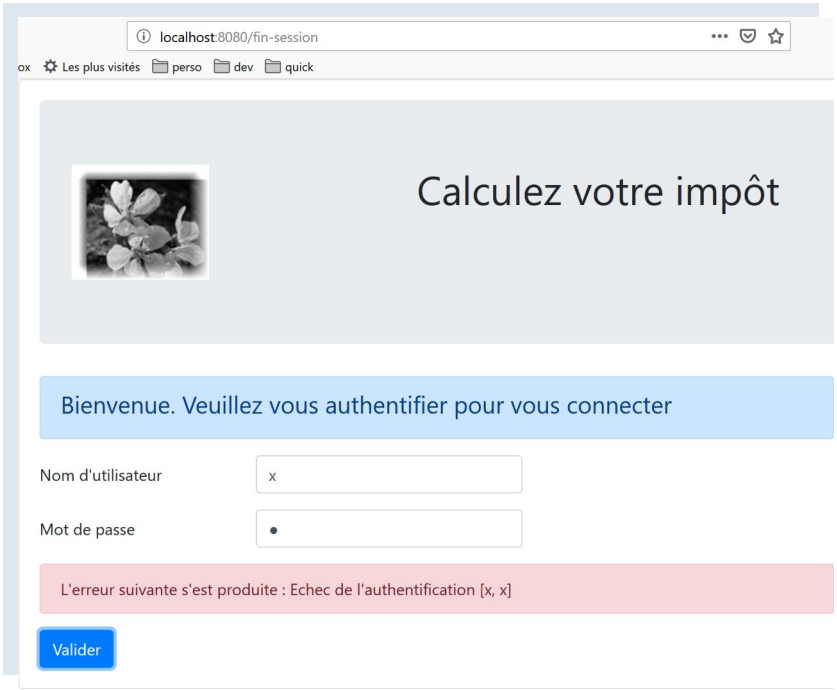
```
74.   },
75.
76.   // gestionnaires d'évts
77.   methods: {
78.     // ----------- authentification
79.     async login() {
80.       try {
81.         // début attente
82.         this.$emit("loading", true);
83.         // authentification bloquante auprès du serveur
84.         const response = await this.$dao.authentifierUtilisateur(
85.           this.user,
86.           this.password
87.         );
88.         // fin du chargement
89.         this.$emit("loading", false);
90.         // analyse de la réponse
91.         if (response.état != 200) {
92.           // on affiche l'erreur
93.           this.message = response.réponse;
94.           this.showError = true;
95.           return;
96.         }
97.         // pas d'erreur
98.         this.showError = false;
99.         // --------- on demande maintenant les données de l'administration fiscale
100.        // début attente
101.        this.$emit("loading", true);
102.        // demande bloquante auprès du serveur
103.        const response2 = await this.$dao.getAdminData();
104.        // fin du chargement
105.        this.$emit("loading", false);
106.        // analyse de la réponse
107.        if (response2.état != 1000) {
108.          // on affiche l'erreur
109.          this.message = response2.réponse;
110.          this.showError = true;
111.          return;
112.        }
113.        // pas d'erreur
114.        this.showError = false;
115.        // on mémorise la donnée reçue dans la couche [métier]
116.        this.$métier.setTaxAdminData(response2.réponse);
117.        // on passe à la vue du calcul de l'impôt
118.        this.$router.push({ name: "calculImpot" });
119.      } catch (error) {
120.        // on remonte l'erreur au composant principal
121.        this.$emit("error", error);
122.      }
123.    }
124.  },
125.  // cycle de vie : le composant vient d'être créé
126.  created() {
127.    // eslint-disable-next-line
128.    console.log("authentification", "created");
129.    // on démarre une session jSON avec le serveur
130.    // début attente
131.    this.$emit("loading", true);
132.    // on initialise la session avec le serveur - requête asynchrone
133.    // on utilise la promesse rendue par les méthodes de la couche [dao]
134.    this.$dao
135.      // on initialise une session jSON
136.      .initSession()
137.      // on a obtenu la réponse
138.      .then(response => {
139.        // fin attente
140.        this.$emit("loading", false);
141.        // analyse de la réponse
142.        if (response.état != 700) {
143.          // on affiche l'erreur
144.          this.message = response.réponse;
145.          this.showError = true;
146.          return;
147.        }
148.        // la session a démarré
149.        this.sessionStarted = true;
150.      })
```

```
151.       // en cas d'erreur
152.       .catch(error => {
153.         // on remonte l'erreur à la vue [Main]
154.         this.$emit("error", error);
155.       });
156.   }
157. };
158. </script>
```

Commentaires

- ligne 3 : la vue **[Authentification]** utilise uniquement la colonne de droite du **[Layout]** (lignes 3 et 4) ;
- lignes 6-38 : le formulaire Bootstrap qui génère la zone 1 de la copie d'écran ci-dessus ;
- ligne 6 : l'événement **[@submit]** se produit lorsque l'utilisateur va cliquer sur le bouton de type **[submit]** de la ligne 35. Le modificateur **[prevent]** demande à ce que la page ne soit pas rechargée lors du **[submit]**. On aurait pu écrire également :
  - une balise <b-form> sans gestion de l'événement **[submit]** ;
  - une balise <b-button> avec l'événement **[@click='login']** et sans l'attribut **[type='submit']** ;

  Ca marche également. L'avantage de la solution retenue est que le submit se fait non seulement avec un clic sur le bouton **[Valider]** mais également sur une validation (touche **[Entrée]**) dans les zones de saisie. C'est donc par commodité pour l'utilisateur que la solution **[<b-form @submit.prevent="login">]** a été retenue ici ;
- lignes 33-37 : une alerte qui apparaît lorsque le serveur a rejeté les identifiants saisis par l'utilisateur :



- ligne 35 : le bouton **[Valider]** n'est pas toujours actif. Son état dépend de l'attribut calculé **[valid]** des lignes 71-73. L'attribut **[valid]** est vrai si :
  - il y a quelque chose dans les champs **[user, password]** du formulaire ;
  - la session jSON a démarré. Au départ, cette session n'a pas démarré (ligne 59) et donc le bouton **[Valider]** est inactif.
- lignes 49-60 : l'état de la vue ;
  - **[user]** représente la saisie de l'utilisateur dans le champ **[user]** (lignes 12-17) du formulaire. La directive **[v-model]** de la ligne 15 établit une liaison bidirectionnelle entre la saisie de l'utilisateur et l'attribut **[user]** de la vue ;
  - **[password]** représente la saisie de l'utilisateur dans le champ **[password]** (lignes 19-24) du formulaire. La directive **[v-model]** de la ligne 22 établit une liaison bidirectionnelle entre la saisie de l'utilisateur et l'attribut **[password]** de la vue ;
  - **[showError]** contrôle (ligne 29) l'affichage de l'alerte des lignes 26-31 ;
  - **[message]** est le message d'erreur (ligne 31) à afficher dans l'alerte des lignes 26-31 ;
  - **[sessionStarted]** indique si la session jSON avec le serveur a démarré ou non. Au départ cet attribut a la valeur **[false]** (ligne 59). La session jSON avec le serveur est initialisée dans l'événement **[created]** du cycle de vie de la vue, lignes 126-156. Si le serveur répond positivement, alors l'attribut **[sessionStarted]** est passé à **[true]** (ligne 149) ;
- lignes 126-156 : la fonction **[created]** est exécutée lorsque la vue **[Authentification]** a été créée (pas forcément encore affichée). En tâche de fond, on initialise alors une session jSON avec le serveur. On sait que c'est la 1ère action à faire avec le serveur de calcul de l'impôt. Pour ce faire, on utilise la couche **[dao]** de l'application (ligne 134). Toutes les méthodes de cette couche sont asynchrones. On utilise ici la promesse (Promise) rendue par la méthode **[$dao.initSession]** qui initialise la session jSON avec le serveur.

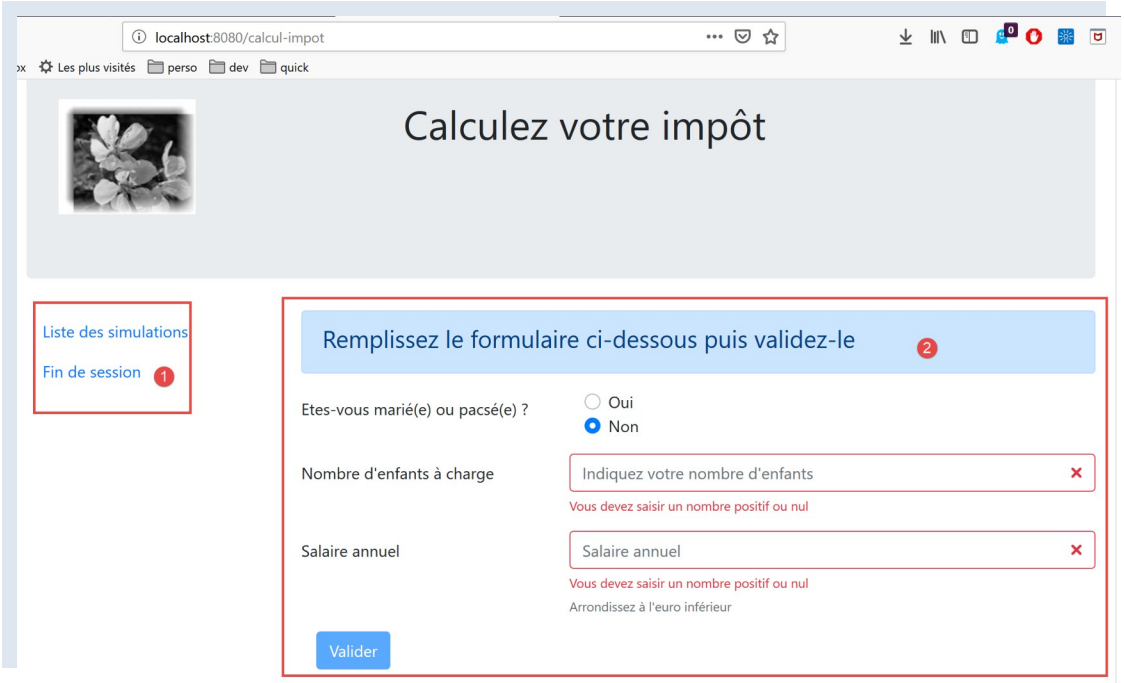
- lignes 138-150 : le code exécuté lorsque le serveur a rendu sa réponse sans erreur ;
- ligne 142 : on vérifie la propriété **[état]** de la réponse. Elle doit avoir la valeur **[700]** pour une opération réussie. Sinon, il s'est produit une erreur dont la cause est indiquée dans la propriété **[response.réponse]** (ligne 144). On affiche alors le message d'erreur de la vue (ligne 145) ;
- ligne 149 : on note que la session jSON a démarré ;
- lignes 152-155 : le code exécuté en cas d'erreur. Celle-ci est remontée à la vue parente **[Main]** qui
  - affichera l'erreur ;
  - cachera le message d'attente ;
  - cachera la vue routée, la vue **[Autentification]** ;
- lignes 79-124 : la méthode **[login]** traite le clic sur le bouton **[Valider]** ;
- ligne 79 : la méthode a été préfixée avec le mot clé **[async]** pour permettre l'utilisation du mot clé **[await]**, lignes 84 et 103 ;
- lignes 84-87 : appel bloquant à la méthode **[$dao.authentifierUtilisateur(user, password)]**. On aurait pu utiliser une promesse **[Promise]** comme il a été fait dans la fonction **[created]**. Nous avons voulu varier les styles. Il n'y a pas de risque à bloquer l'utilisateur car nous avons mis un **[timeout]** de 2 secondes à toutes les requêtes HTTP. Il n'attendra pas longtemps. De plus, il ne peut rien faire tant que le serveur n'a pas rendu sa réponse car alors le bouton **[Valider]** reste inactif ;
- ligne 91 : le serveur de calcul de l'impôt envoie des réponses jSON ayant toutes la structure **[{'action':action, 'état':val, 'réponse':réponse}]**. L'authentification a réussi si **[état==200]**. Si ce n'est pas le cas, un message d'erreur est affiché, lignes 93-94 ;
- ligne 98 : on cache un éventuel message d'erreur d'une opération précédente ;
- lignes 99-116 : on demande mainteant au serveur les données de l'administration fiscale qui permettent le calcul de l'impôt. Dans **[this.$métier]** nous avons une instance de la classe **[Métier]** qui pour l'instant ne peut rien faire car elle n'a pas ces données ;
- ligne 103 : les données de l'administration fiscale sont demandées au serveur par une opération bloquante ;
- lignes 107-112 : la réponse du serveur est analysée. Elle doit avoir une valeur d'état égale à 1000 sinon c'est qu'il s'est produit une erreur. Dans ce dernier cas, on affiche le message d'erreur (lignes 109-110) ;
- lignes 113-118 : en cas de réussite de l'opération, on :
  - cache le message d'erreur, ligne 114 ;
  - on transmet les données de l'administration fiscale à la couche **[métier]** (ligne 116) ;
  - on fait afficher la vue **[CalculImpot]**, ligne 118. On se rappelle que **[this.$router]** désigne le routeur de l'application. La méthode **[push]** permet de fixer la prochaine vue routée. Ici on la désigne par son attribut **[name]**. On aurait pu également la désigner par son attribut **[path]**. Ces informations sont dans le fichier de routage :

```
1.  // calcul de l'impôt
2.    {
3.      path: '/calcul-impot', name: 'calculImpot', component: CalculImpot
4.    },
```

- lignes 119-122 : le **[catch]** se déclenche lorsqu'une des deux requêtes HTTP a échoué (serveur pas présent, timeout dépassé, ...). On signale alors l'erreur à la vue parente **[Main]** qui l'affichera, cachera le message d'attente et la vue **[Authentification]** ;

18.4.4 La vue [CalculImpot]

La vue **[CalculImpot]** est la suivante :



- **[1]** : un menu de navigation occupe la colonne de gauche de la vue routée ;
- **[2]** : le formulaire de calcul de l'impôt occupe la colonne de droite de la vue routée ;

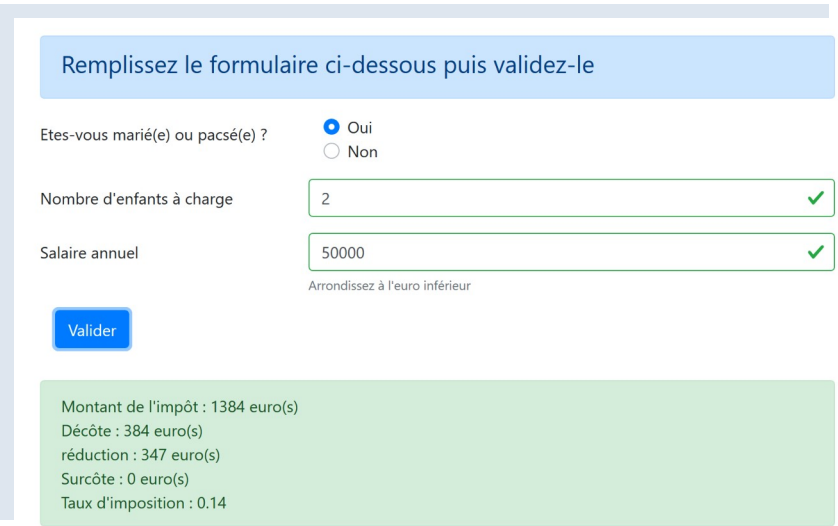
Le code de la vue **[CalculImpot]** est le suivant :

```
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 cols="3" />
14.       <!-- zone de neuf colonnes -->
15.       <b-col cols="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.   // état interne
32.   data() {
33.     return {
34.       // options du menu
35.       options: [
36.         {
37.           text: "Liste des simulations",
38.           path: "/liste-des-simulations"
39.         },
40.         {
41.           text: "Fin de session",
```

```
42.           path: "/fin-session"
43.         }
44.       ],
45.       // résultat du calcul de l'impôt
46.       résultat: "",
47.       résultatObtenu: false
48.     };
49.   },
50.   // composants utilisés
51.   components: {
52.     Layout,
53.     FormCalculImpot,
54.     Menu
55.   },
56.   // méthodes de gestion des évts
57.   methods: {
58.     // résultat du calcul de l'impôt
59.     handleResultatObtenu(résultat) {
60.       // on construit le résultat en chaîne HTML
61.       const impôt = "Montant de l'impôt : " + résultat.impôt + " euro(s)";
62.       const décôte = "Décôte : " + résultat.décôte + " euro(s)";
63.       const réduction = "Réduction : " + résultat.réduction + " euro(s)";
64.       const surcôte = "Surcôte : " + résultat.surcôte + " euro(s)";
65.       const taux = "Taux d'imposition : " + résultat.taux;
66.       this.résultat =
67.         impôt +
68.         "<br/>" +
69.         décôte +
70.         "<br/>" +
71.         réduction +
72.         "<br/>" +
73.         surcôte +
74.         "<br/>" +
75.         taux;
76.       // affichage du résultat
77.       this.résultatObtenu = true;
78.       // ---- maj du store [Vuex]
79.       // une simulation de +
80.       this.$store.commit("addSimulation", résultat);
81.     }
82.   }
83. };
84. </script>
```

Commentaires

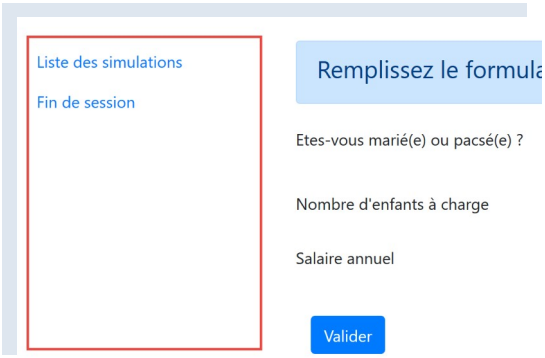
- ligne 4 : les deux colonnes du **[Layout]** sont ici présentes ;
- ligne 6 : le formulaire de calcul de l'impôt occupe la colonne de droite. Il émet l'événement **[resultatObtenu]** lorsque le résultat du calcul de l'impôt a été obtenu. On notera que les noms d'événements et les noms des méthodes qui les gèrent ne peuvent contenir de caractères accentués ;
- ligne 8 : le menu de navigation occupe la colonne de gauche ;
- lignes 11-20 : le résultat du calcul de l'impôt est affiché sous le formulaire :



- ligne 11 : le résultat n'est affiché que si l'attribut **[résultatObtenu]** (ligne 47) vaut **[true]** ;
- lignes 34-48 : l'état de la vue :
  - **[options]** : la liste des options du menu de navigation. Ce tableau est passé en paramètre au composant **[Menu]**, ligne 8 ;
  - **[résultat]** : le résultat du calcul de l'impôt. Ce résultat est une chaîne HTML. C'est pourquoi on a utilisé la directive **[v-html]** à la ligne 17 pour l'afficher ;
  - **[résultatObtenu]** : le booléen qui contrôle l'affichage du résultat, ligne 11 ;
- lignes 59-81 : la méthode **[handleResultatObtenu]** affiche le résultat du calcul de l'impôt que lui a envoyé la vue fille **[FormCalculImpot]**, ligne 6. Ce résultat est un objet avec les propriétés **[impot, décôte, réduction, surcôte, taux, marié, enfants, salaire]** ;
- lignes 61-75 : on inscrit l'objet **[impot, décôte, réduction, surcôte, taux]** dans un texte HTML qui est visualisé par la ligne 17 du template ;
- ligne 77 : on affiche ce résultat ;
- ligne 80 : on appelle la mutation **[addSimulation]** du store Vuex qui va ajouter **[résultat]** aux simulations déjà présentes dans le store ;

18.4.5 Le menu de navigation [Menu]

Le menu de navigation s'affiche dans la colonne de gauche des vues routées :



Le code de la vue **[Menu]** est le suivant :

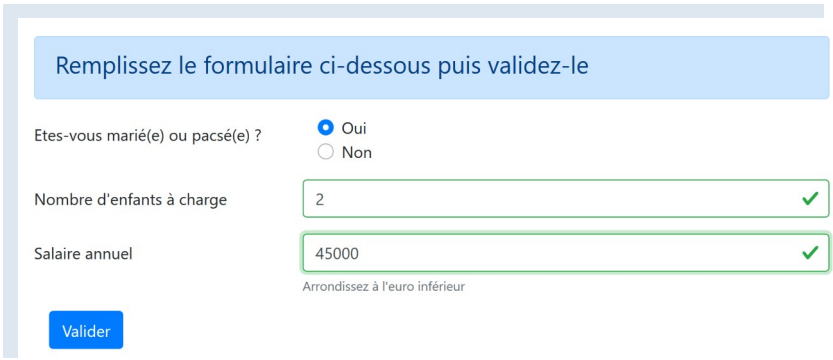
```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <!-- menu Bootstrap vertical -->
4.    <b-nav vertical>
5.      <!-- options du menu -->
6.      <b-nav-item
7.        v-for="(option,index) of options"
8.        :key="index"
9.        :to="option.path"
10.       exact
11.       exact-active-class="active"
12.     >{{option.text}}</b-nav-item>
13.   </b-nav>
14. </template>
15.
16. <script>
17. export default {
18.   // paramètres de la vue
19.   props: {
20.     options: {
21.       type: Array
22.     }
23.   }
24. };
25. </script>
```

Commentaires

- les options du menu sont fournies par le paramètre **[options]** (lignes 7, 20-22) ;
- chaque élément du tableau **[options]** a une propriété **[text]** (ligne 12) qui est le texte du lien et une propriété **[path]** (ligne 9) qui sera le chemin de la vue cible du lien ;

18.4.6 La vue [FormCalculImpot]

Cette vue fournit le formulaire de calcul de l'impôt :



Son code est le 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-alert show variant="primary">
7.        <h4>Remplissez le formulaire ci-dessous puis validez-le</h4>
8.      </b-alert>
9.      <!-- éléments du formulaire -->
10.     <!-- première ligne -->
11.     <b-form-group label="Etes-vous marié(e) ou pacsé(e) ?" label-cols="4">
12.       <!-- boutons radio sur 5 colonnes-->
13.       <b-col cols="5">
14.         <b-form-radio v-model="marié" value="oui">Oui</b-form-radio>
15.         <b-form-radio v-model="marié" value="non">Non</b-form-radio>
16.       </b-col>
17.     </b-form-group>
18.     <!-- deuxième ligne -->
19.     <b-form-group label="Nombre d'enfants à charge" label-cols="4" label-for="enfants">
20.       <b-input
21.         type="text"
22.         id="enfants"
23.         placeholder="Indiquez votre nombre d'enfants"
24.         v-model="enfants"
25.         :state="enfantsValide"
26.       />
27.       <!-- message d'erreur éventuel -->
28.       <b-form-invalid-feedback :state="enfantsValide">Vous devez saisir un nombre positif ou nul</b-form-
    invalid-feedback>
29.     </b-form-group>
30.     <!-- troisème ligne -->
31.     <b-form-group
32.       label="Salaire annuel"
33.       label-cols="4"
34.       label-for="salaire"
35.       description="Arrondissez à l'euro inférieur"
36.     >
37.       <b-input
38.         type="text"
39.         id="salaire"
40.         placeholder="Salaire annuel"
41.         v-model="salaire"
42.         :state="salaireValide"
43.       />
44.       <!-- message d'erreur éventuel -->
45.       <b-form-invalid-feedback :state="salaireValide">Vous devez saisir un nombre positif ou nul</b-form-
    invalid-feedback>
46.     </b-form-group>
47.     <!-- quatrième ligne, bouton [submit] sur 5 colonnes -->
48.     <b-col cols="5">
49.       <b-button type="submit" variant="primary" :disabled="formInvalide">Valider</b-button>
50.     </b-col>
51.   </b-form>
```

```
52.  </template>
53.
54.  <!-- script -->
55.  <script>
56.  export default {
57.    // état interne
58.    data() {
59.      return {
60.        // marié ou pas
61.        marié: "non",
62.        // nombre d'enfants
63.        enfants: "",
64.        // salaire annuel
65.        salaire: ""
66.      };
67.    },
68.    // état interne calculé
69.    computed: {
70.      // validation du formulaire
71.      formInvalide() {
72.        return (
73.          // salaire invalide
74.          !this.salaireValide ||
75.          // ou enfants invalide
76.          !this.enfantsValide ||
77.          // ou données fiscales pas obtenues
78.          !this.$métier.taxAdminData
79.        );
80.      },
81.      // validation du salaire
82.      salaireValide() {
83.        // doit être numérique >=0
84.        return Boolean(this.salaire.match(/^\s*\d+\s*$/));
85.      },
86.      // validation des enfants
87.      enfantsValide() {
88.        // doit être numérique >=0
89.        return Boolean(this.enfants.match(/^\s*\d+\s*$/));
90.      }
91.    },
92.    // gestionnaire d'évts
93.    methods: {
94.      calculerImpot() {
95.        // on calcule l'impôt à l'aide de la couche [métier]
96.        const résultat = this.$métier.calculerImpot(
97.          this.marié,
98.          this.enfants,
99.          this.salaire
100.       );
101.       // eslint-disable-next-line
102.       console.log("résultat=", résultat);
103.       // on complète le résultat
104.       résultat.marié = this.marié;
105.       résultat.enfants = this.enfants;
106.       résultat.salaire = this.salaire;
107.       // on émet l'évt [resultatObtenu]
108.       this.$emit("resultatObtenu", résultat);
109.     }
110.   }
111. };
112. </script>
```

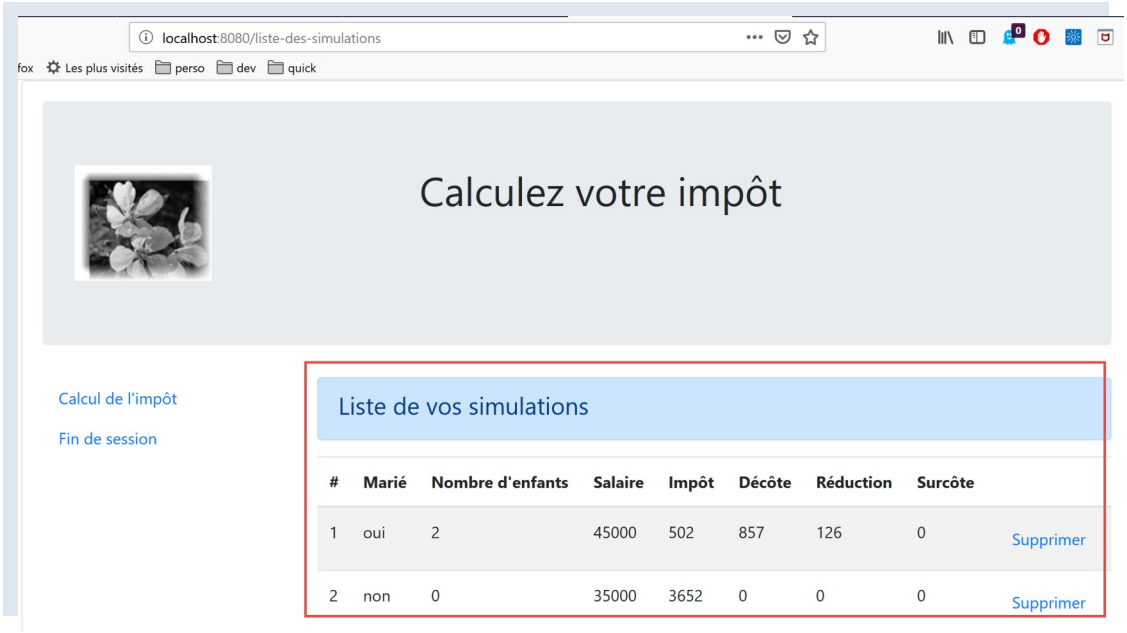
Commentaires

- lignes 4-51 : le formulaire Bootstrap ;
- lignes 11-17 : un groupe de boutons radio avec leur libellé ;
- lignes 14-15 : la balise <b-form-radio> assure l'affichage d'un bouton radio :
  - ligne 14 : la directive **`[v-model]`** assure que lors d'un clic sur le bouton, l'attribut **`[marié]`** de la ligne 61 recevra la valeur **`[oui]`** (attribut **`[value="oui"]`**) ;
  - ligne 15 : la directive **`[v-model]`** assure que lors d'un clic sur le bouton, l'attribut **`[marié]`** de la ligne 61 recevra la valeur **`[non]`** (attribut **`[value="non"]`**) ;
- lignes 19-29 : la partie saisie du nombre d'enfants :
  - ligne 24 : la saisie du nombre d'enfants est liée à l'attribut **`[enfants]`** de la ligne 63 ;
  - ligne 25 : la validité de la saisie est vérifiée par l'attribut calculé **`[enfantsValide]`** des lignes 87-89 ;
  - ligne 28 : assure l'affichage d'un message d'erreur si la saisie est invalide ;

- lignes 31-45 : la partie saisie du salaire annuel :
  - ligne 35 : affiche un message d'aide juste sous la zone de saisie ;
  - ligne 41 : la saisie du salaire est liée à l'attribut **[salaire]** de la ligne 65 ;
  - ligne 42 : la validité de la saisie est vérifiée par l'attribut calculé **[salaireValide]** des lignes 82-85 ;
  - ligne 45 : assure l'affichage d'un message d'erreur si la saisie est invalide ;
- lignes 48-50 : un bouton de type **[submit]**. Lorsqu'on clique sur ce bouton ou lorsqu'on valide une saisie avec la touche **[Entrée]**, la méthode **[calculerImpot]** est exécutée (ligne 94) ;
  - ligne 49 : l'état du bouton actif / inactif est contrôlé par l'attribut calculé **[formInvalide]** des lignes 71-80 ;
- lignes 71-80 : le formulaire est valide si :
  - le nombre d'enfants est valide ;
  - le salaire est valide ;
  - l'application a obtenu du serveur les données de l'administration fiscale permettant le calcul de l'impôt. On rappelle que cette donnée est enregistrée dans la propriété **[$métier.taxAdminData]**. La vue **[FormCalculImpot]** peut être affichée avant que cette donnée ait été obtenue car elle est demandée de façon asynchrone en même temps que se produit l'affichage de la vue. On s'assure ici que l'utilisateur ne peut pas cliquer sur le bouton **[Valider]** tant que la donnée n'a pas été obtenue ;
- lignes 94-109 : la méthode de calcul de l'impôt :
  - lignes 96-100 : c'est la couche **[métier]** qui fait ce calcul. C'est un calcul synchrone. Une fois la donnée **[taxAdminData]** a été obtenue, le client **[Vue]** n'a plus à communiquer avec le serveur. Tout se fait localement. On obtient un objet **[résultat]** avec les propriétés **[impôt, décôte, surcôte, réduction, taux]** ;
  - lignes 104-106 : on ajoute les propriétés **[marié, enfants, salaire]** au résultat ;
  - ligne 108 : le résultat est passé à la vue parent **[CalculImpot]** via l'événement **[resultatObtenu]**. Cette vue est chargée d'afficher le résultat ;

18.4.7 La vue [ListeSimulations]

La vue **[ListeSimulations]** affiche la liste des simulations faites par l'utilisateur :



Le code de la vue est le suivant :

```
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">
```

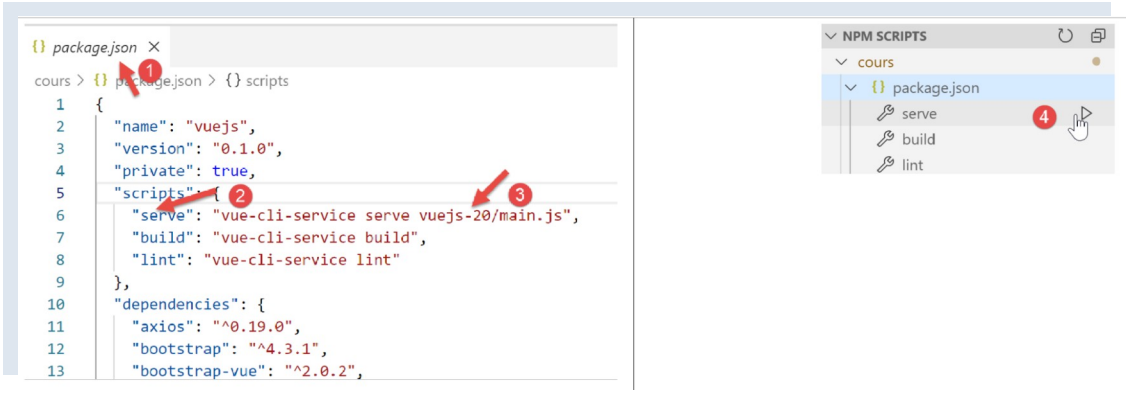
```vue
            <!-- il y a des simulations -->
            <b-alert show variant="primary">
              <h4>Liste de vos simulations</h4>
            </b-alert>
            <!-- tableau des simulations -->
            <b-table striped hover responsive :items="simulations" :fields="fields">
              <template v-slot:cell(action)="data">
                <b-button variant="link" @click="supprimerSimulation(data.index)">Supprimer</b-button>
              </template>
            </b-table>
          </template>
        </template>
        <!-- menu de navigation dans colonne de gauche -->
        <Menu slot="left" :options="options" />
      </Layout>
    </div>
  </template>

  <script>
    // imports
    import Layout from "./Layout";
    import Menu from "./Menu";
    export default {
      // composants
      components: {
        Layout,
        Menu
      },
      // état interne
      data() {
        return {
          // options du menu de navigation
          options: [
            {
              text: "Calcul de l'impôt",
              path: "/calcul-impot"
            },
            {
              text: "Fin de session",
              path: "/fin-session"
            }
          ],
          // paramètres de la table HTML
          fields: [
            { label: "#", key: "id" },
            { label: "Marié", key: "marié" },
            { label: "Nombre d'enfants", key: "enfants" },
            { label: "Salaire", key: "salaire" },
            { label: "Impôt", key: "impôt" },
            { label: "Décôte", key: "décôte" },
            { label: "Réduction", key: "réduction" },
            { label: "Surcôte", key: "surcôte" },
            { label: "", key: "action" }
          ]
        };
      },
      // état interne calculé
      computed: {
        // liste des simulations prise dans le store Vuex
        simulations() {
          return this.$store.state.simulations;
        }
      },
      // méthodes
      methods: {
        supprimerSimulation(index) {
          // eslint-disable-next-line
          console.log("supprimerSimulation", index);
          // suppression de la simulation n° [index]
          this.$store.commit("deleteSimulation", index);
        }
      }
    };
  </script>
```

Commentaires

- ligne 5 : la vue occupe les deux colonnes de la mise en page **[Layout]** des vues routées ;
- lignes 7-26 : les simulations vont dans la colonne de droite ;
- ligne 28 : le menu de navigation va dans la colonne de gauche ;
- lignes 8, 14, 20, 75 : les simulations proviennent du store **[Vuex]** **[$this.store]** ;
- lignes 8-13 : alerte affichée lorsque la liste des simulations est vide ;
- lignes 14-25 : la table HTML affichée lorsque la liste des simulations n'est pas vide ;
- lignes 20-24 : la table HTML est générée par une balise <b-table> ;
  - ligne 20 : le tableau des simulations est fourni par l'attribut calculé **[simulations]** des lignes 74-76 ;
  - ligne 20 : la configuration de la table HTML est faite par l'attribut calculé **[fields]** des lignes 58-69. Ligne 67, la colonne de clé **[action]** est la dernière colonne de la table HTML ;
  - lignes 21-23 : template de la dernière colonne de la table HTML ;
  - ligne 22 : on y met un bouton de type lien. Lorsqu'on clique dessus, la méthode **[supprimerSimulation(data.index)]** est appelée, où **[data]** représente la ligne courante (ligne 21). **[data.index]** représente le n° de cette ligne dans la liste des lignes affichées ;
- ligne 28 : génération du menu de navigation. Les options de celui-ci sont fournies par l'attribut **[options]** des lignes 47-56 ;
- lignes 80-85 : la méthode qui réagit au clic sur un lien **[Supprimer]** de la page HTML ;
  - ligne 84 : on fait appel à la mutation **[deleteSimulation]** du store **[Vuex]** (cf paragraphe |vuejs-15|) ;

18.5 Exécution du projet



Il faut également lancer le serveur **[Laragon]** (cf document |https://tahe.developpez.com/tutoriels-cours/php7|) pour que le serveur de calcul d'impôt soit en ligne.

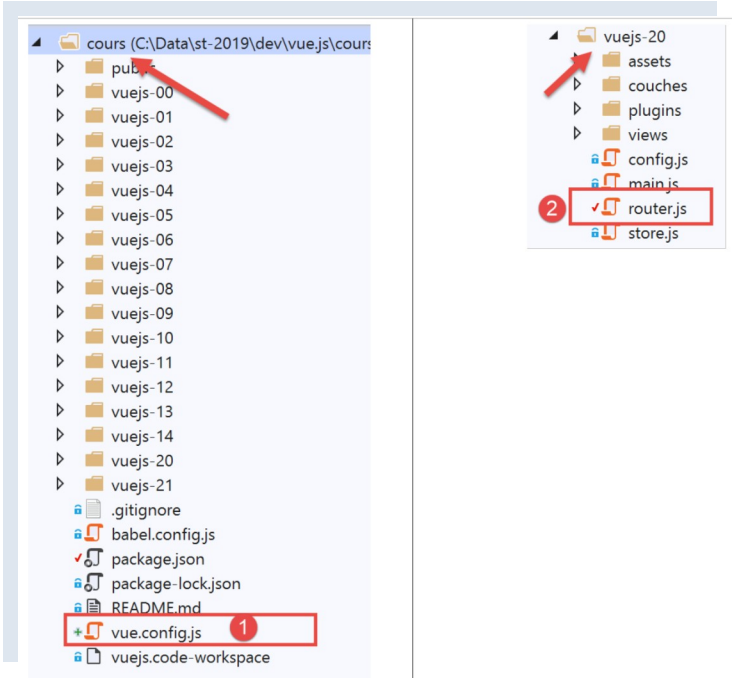
18.6 Déploiement de l'application sur un serveur local

Actuellement, notre client **[Vue]** est déployé sur un serveur de test à l'URL **[http://localhost:8080]**. Nous allons le déployer sur le serveur **[Laragon]** à l'URL **[http://localhost:80]**. Il y a plusieurs étapes à effectuer pour en arriver là.

étape 1

Tout d'abord, nous allons faire en sorte que le client **[Vue]** soit déployé sur le serveur de test à l'URL **[http://localhost:8080/client-vuejs-impot/]**.

Nous créons un fichier **[vue.config.js]** à la racine de notre projet **[VSCode]** actuel :



Le fichier **[vue.config.js] [1]** aura le contenu suivant :

```
1.  // vue.config.js
2.  module.exports = {
3.    // l'URL de service du client [vuejs] du serveur de calcul de l'impôt
4.    publicPath: '/client-vuejs-impot/'
5.  }
```

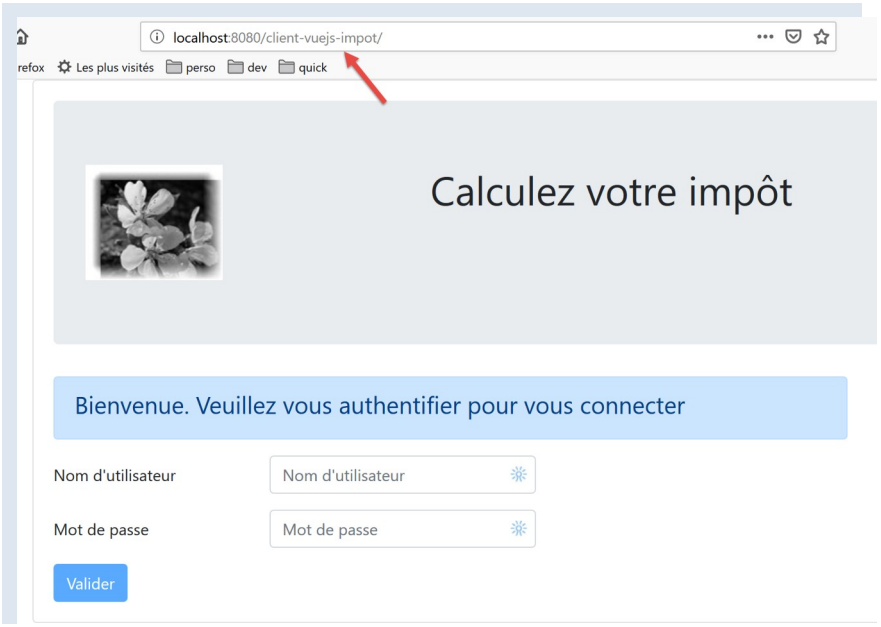
Il nous faut également modifier le fichier de routage **[router.js] [2]** :

```
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.
9.  // plugin de routage
10. Vue.use(VueRouter)
11.
12. // les routes de l'application
13. const routes = [
14.   // authentification
15.   {
16.     path: '/', name: 'authentification', component: Authentification
17.   },
18.   // calcul de l'impôt
19.   {
20.     path: '/calcul-impot', name: 'calculImpot', component: CalculImpot
21.   },
22.   // liste des simulations
23.   {
24.     path: '/liste-des-simulations', name: 'listeSimulations', component: ListeSimulations
25.   },
26.   // fin de session
27.   {
28.     path: '/fin-session', name: 'finSession', component: Authentification
29.   }
30. ]
31.
32. // le routeur
33. const router = new VueRouter({
34.   // les routes
35.   routes,
36.   // le mode d'affichage des routes dans le navigateur
```

```
37.    mode: 'history',
38.    // l'URL de base de l'application
39.    base: '/client-vuejs-impot/'
40.  })
41.
42.  // export du router
43.  export default router
```

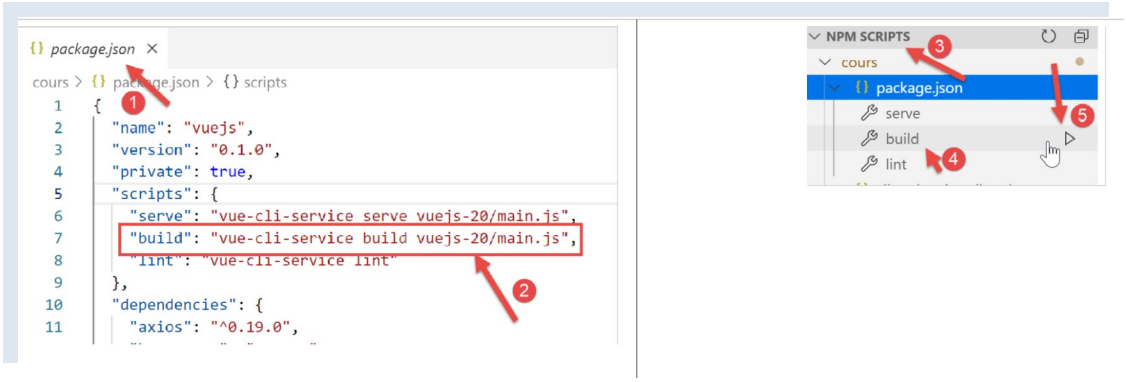
- ligne 39 : on indique au routeur que les chemins des routes définies lignes 13-30 sont relatives au chemin défini ligne 39. Par exemple, le chemin de la ligne 20 **[/calcul-impot]** deviendra **[/client-vuejs-impot/calcul-impot]** ;

On peut alors tester de nouveau le projet **[vuejs-20]** pour vérifier le changement des chemins de l'application :



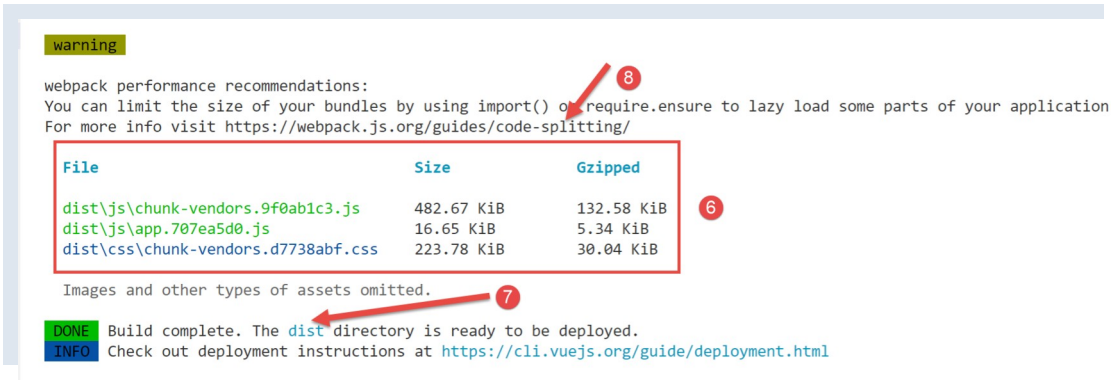
étape 2

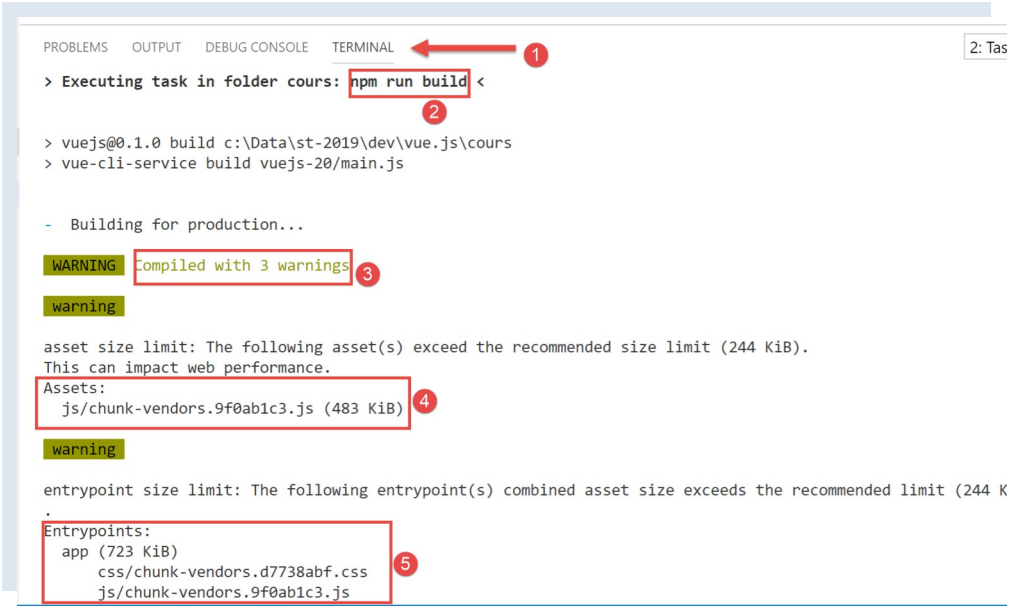
Nous construisons maintenant la version de production du projet **[vuejs-20]** :



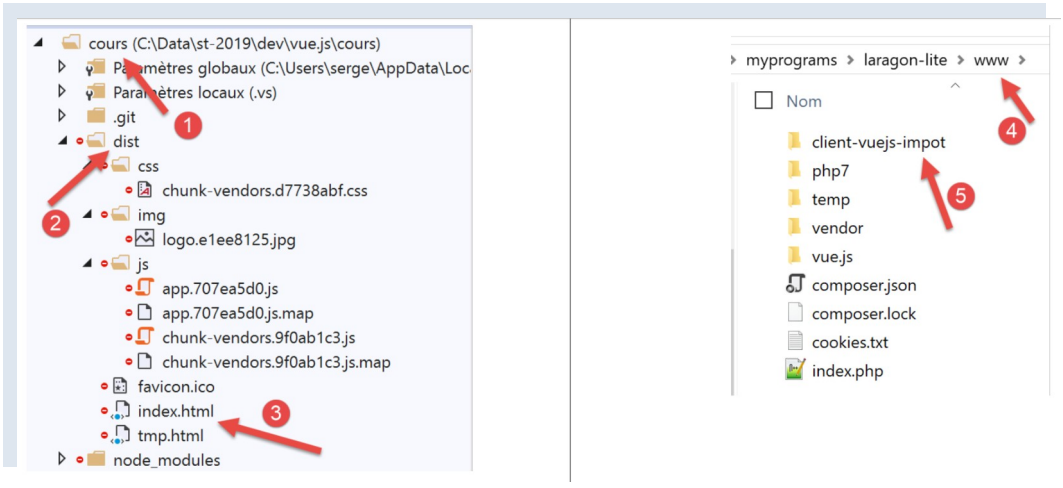
- en **[1-2]**, nous configurons la tâche **[build] [2]** dans le fichier **[package.json] [1]** ;
- en **[3-5]**, nous exécutons cette tâche. C'est elle qui va construire la version de production du projet **[vuejs-20]** ;

L'exécution de la tâche **[build]** se passe dans un terminal de **[VSCode]** :





- en **[3-6]**, des avertissements nous disent que le code généré est trop gros et qu'il faudrait le découper **[8]**. Cela relève de l'optimisation de l'architecture du code que nous n'aborderons pas ici ;
- en **[7]**, on nous dit que le dossier **[dist]** contient la version de production générée :



- en **[3]**, le fichier **[index.html]** est le fichier qui sera utilisé lorsqu'on demandera l'URL **[https://localhost:80/client-vue-js-impot/]**;

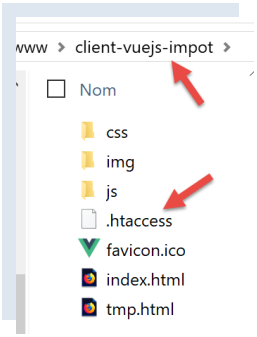
On a ici un site **statique** qui peut être déployé sur n'importe quel serveur. Nous allons le déployer sur le serveur Laragon local (cf document |https://tahe.developpez.com/tutoriels-cours/php7|). Le dossier **[dist] [2]** est copié dans le dossier **[<laragon>/www] [4]**

où <laragon> est le dossier d'installation du serveur Laragon. Nous renommons ce dossier **[client-vuejs-impot] [5]** puisque nous avons configuré la version de production pour fonctionner à l'URL **[/client-vuejs-impot/]**.

étape 3

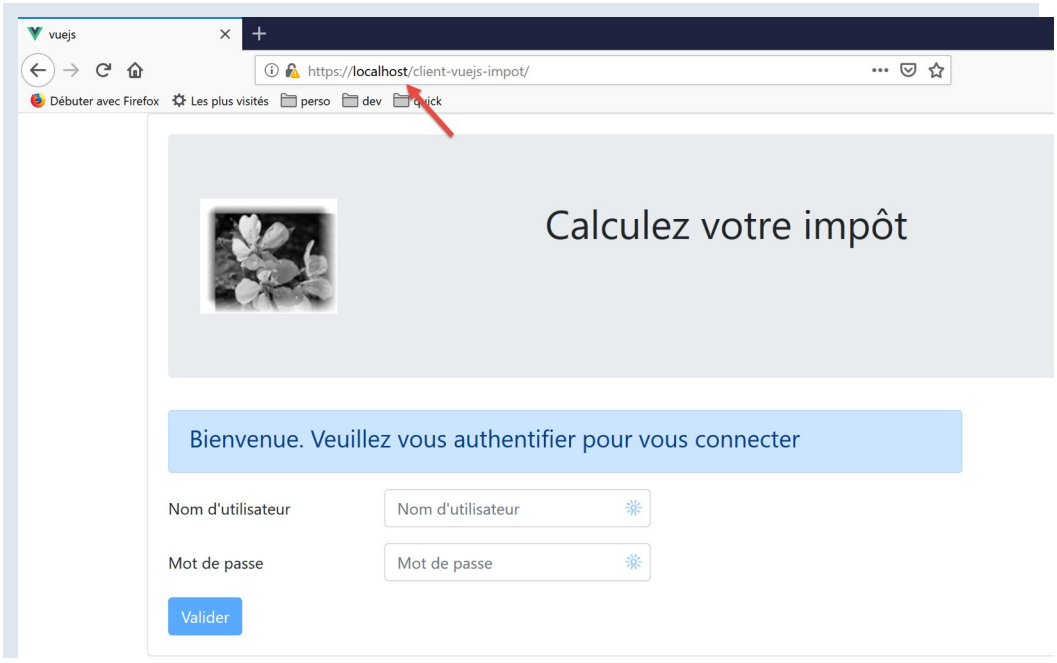
Nous ajoutons dans le dossier **[client-vuejs-impot]** qui vient d'être créé le fichier **[.htaccess]** suivant :

```
1.  <IfModule mod_rewrite.c>
2.    RewriteEngine On
3.    RewriteBase /client-vuejs-impot/
4.    RewriteRule ^index\.html$ - [L]
5.    RewriteCond %{REQUEST_FILENAME} !-f
6.    RewriteCond %{REQUEST_FILENAME} !-d
7.    RewriteRule . /client-vuejs-impot/index.html [L]
8.  </IfModule>
```



Ce fichier est un fichier de configuration du serveur web Apache. Si nous ne le mettons pas et que nous demandons directement l'URL **[https://localhost/client-vuejs-impot/calcul-impot]**, sans passer d'abord par l'URL **[https://localhost/client-vuejs-impot/]** nous obtenons une erreur 404. Avec ce fichier, nous obtenons bien la vue **[CalculImpot]**.

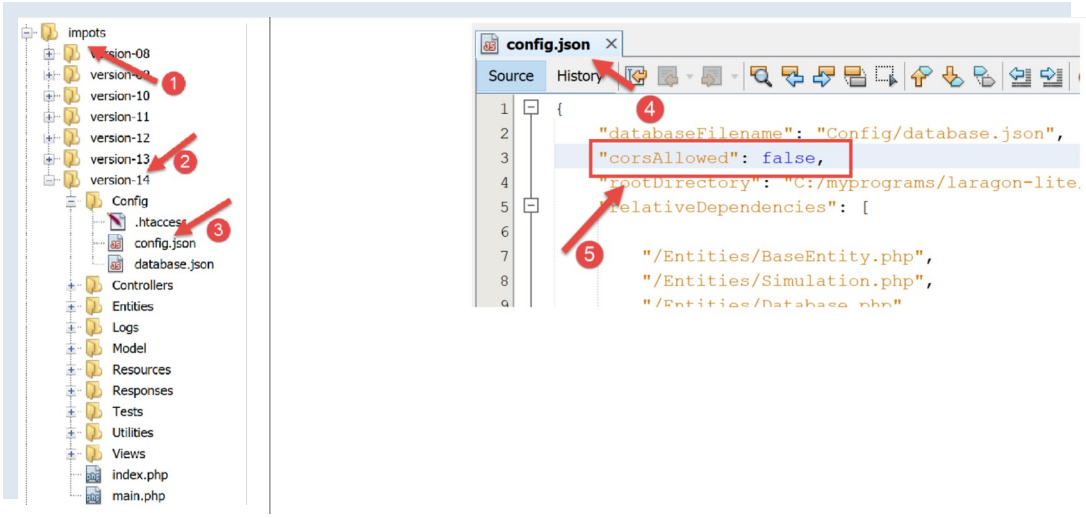
Ceci fait, nous lançons le serveur Laragon si ce n'est déjà fait et demandons l'URL **[https://localhost/client-vuejs-impot/]** :



Le lecteur est invité à tester la version de production de notre application.

Nous pouvons modifier le serveur de calcul de l'impôt sur un point : les entêtes CORS qu'il envoie systématiquement à ses clients. Cela avait été nécessité pour la version du client exécutée à partir du domaine **[localhost:8080]**. Maintenant que client et serveur s'exécutent tous deux dans le domaine **[localhost:80]**, les entêtes CORS deviennent inutiles.

Nous modifions le fichier **[config.json]** de la version 14 du serveur :

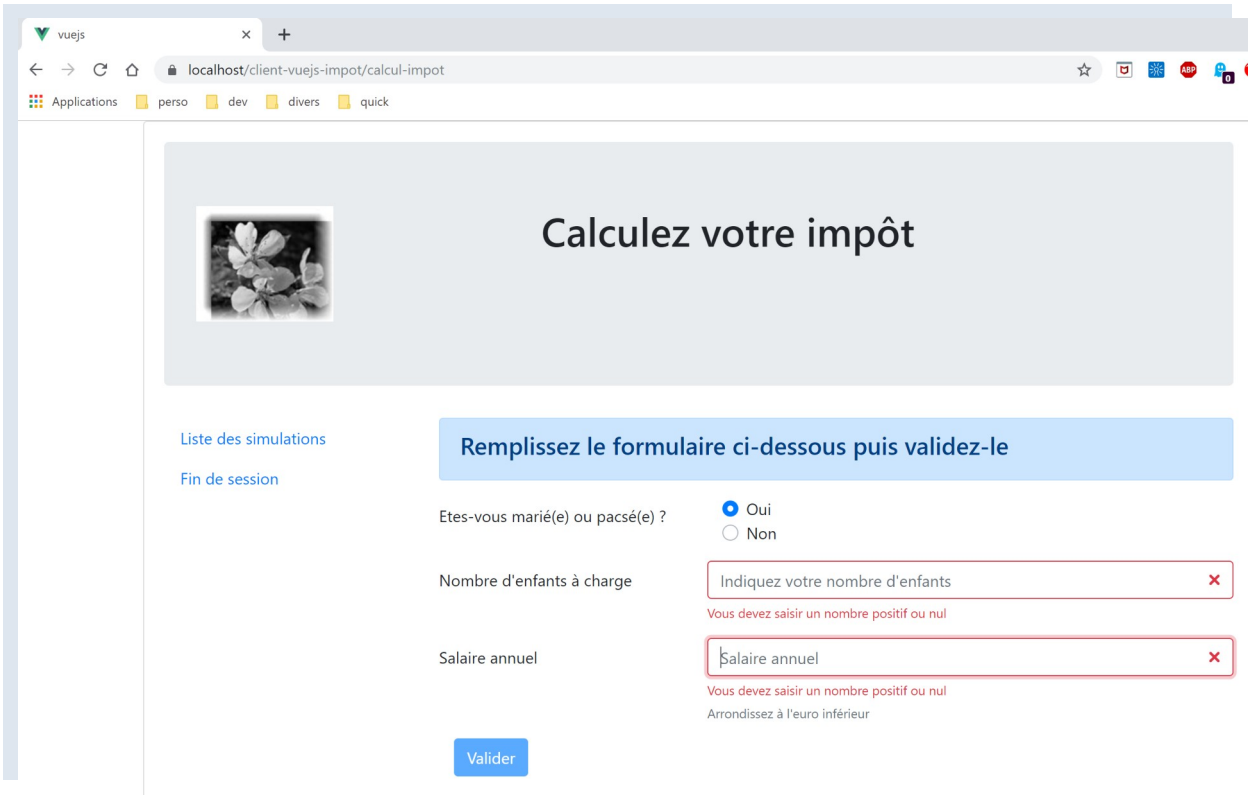


- en **[4]**, nous indiquons que désormais les requêtes CORS sont refusées ;

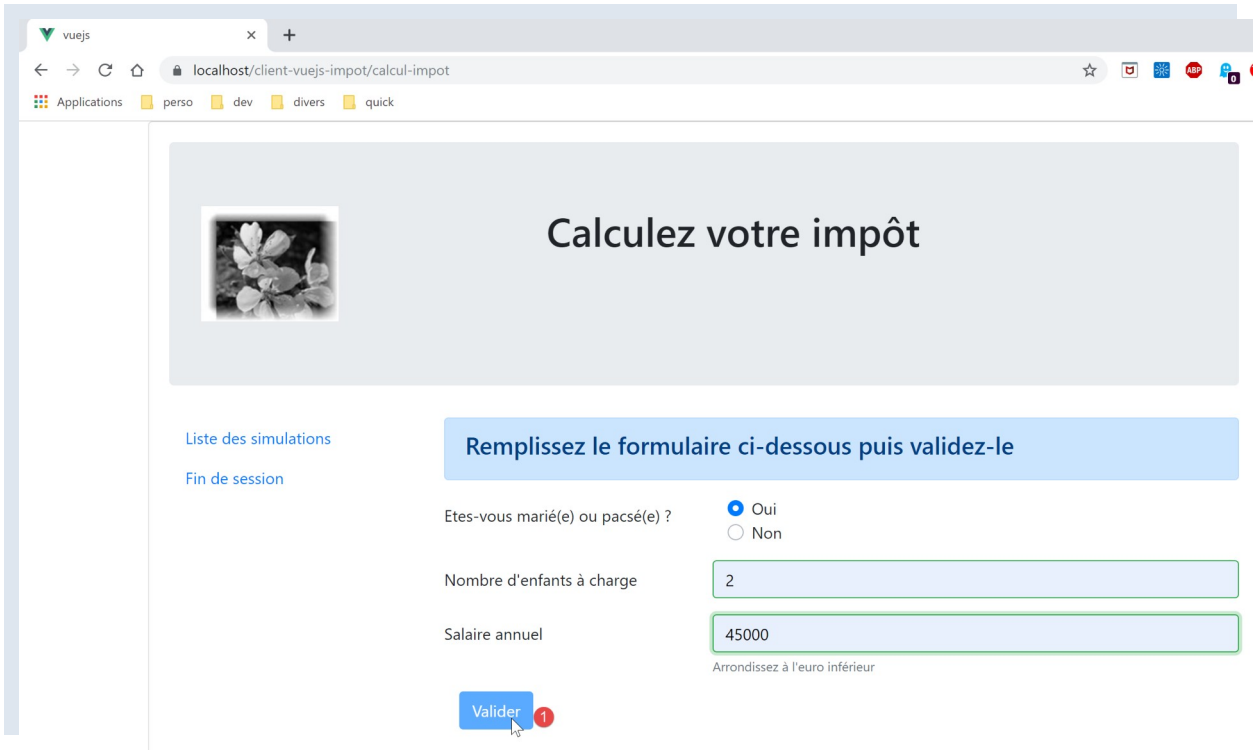
Sauvegardons cette modification et redemandons l'URL **[https://localhost/client-vuejs-impot/]**. Ca doit continuer à marcher.

18.7 Gestion des URL manuelles

Au lieu d'utiliser sagement les liens du menu de navigation, l'utilisateur peut vouloir taper les URL de l'application manuellement dans le champ d'adresse du navigateur. Demandons par exemple l'URL **[https://client-vuejs-impot/calcul-impot]** sans passer par la case d'authentification. Un hacker tenterait sûrement ça. On obtient la vue suivante :



On obtient bien la vue du calcul de l'impôt. Maintenant essayons de remplir les zones de saisie et de les valider :



On découvre alors que le bouton **[1] [Valider]** reste toujours désactivé même si les saisies sont correctes. Regardons le code de la vue **[FormCalculImpot]** :

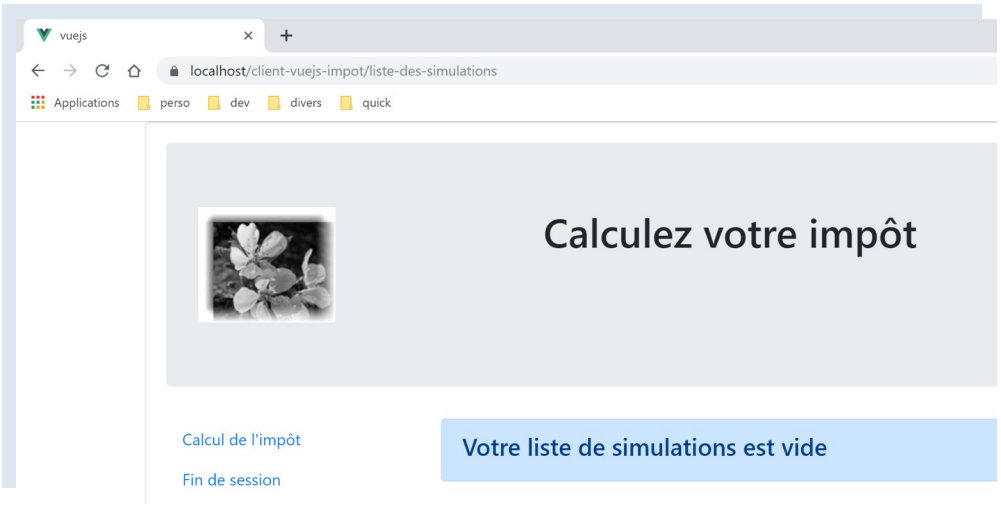
```
1.  <b-col cols="5">
2.        <b-button type="submit" variant="primary" :disabled="formInvalide">Valider</b-button>
3.  </b-col>
```

Ligne 2, on voit que son état actif / inactif dépend de la propriété **[formInvalide]**. Celle-ci est la propriété calculée suivante :

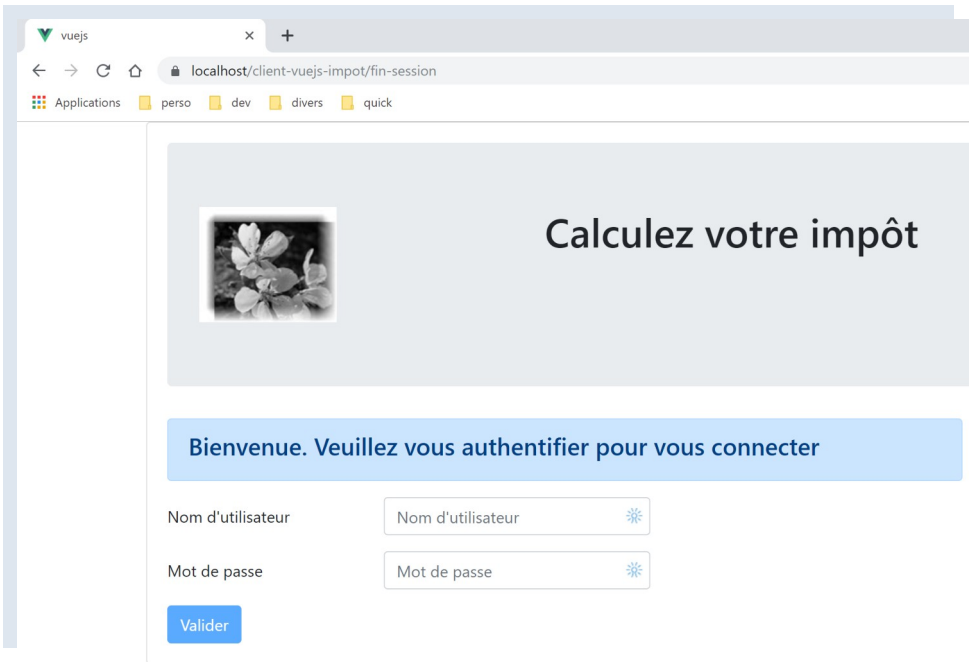
```
1.  formInvalide() {
2.        return (
3.          // salaire invalide
4.          !this.salaireValide ||
5.          // ou enfants invalide
6.          !this.enfantsValide ||
7.          // ou données fiscales pas obtenues
8.          !this.$métier.taxAdminData
9.        );
10. },
```

Ligne 8, on voit que pour que le formulaire soit valide, il faut avoir obtenu les données fiscales. Or celles-ci sont obtenues lors de la validation de la vue **[Authentification]** que l'utilisateur a 'sautée'. Il ne pourra donc pas valider le formulaire. S'il avait pu le faire, il aurait reçu un message d'erreur du serveur lui indiquant qu'il n'était pas authentifié. Les vérifications doivent toujours être faites côté serveur. Les vérifications côté navigateur peuvent toujours être contournées. Il suffit de prendre un client de type **[Postman]** qui enverra des requêtes brutes au serveur.

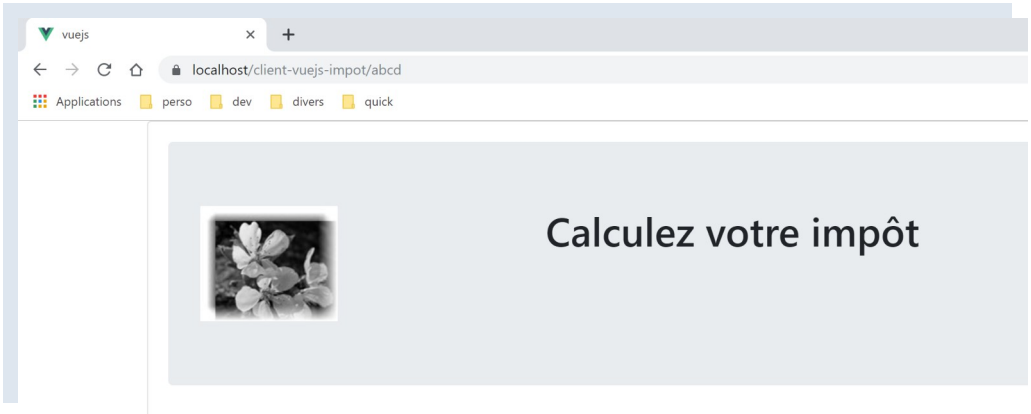
Maintenant demandons l'URL **[https://localhost/client-vuejs-impot/liste-des-simulations]**. On obtient la vue suivante :



Maintenant l'URL **[https://localhost/client-vuejs-impot/fin-session]**. Nous obtenons la vue suivante :

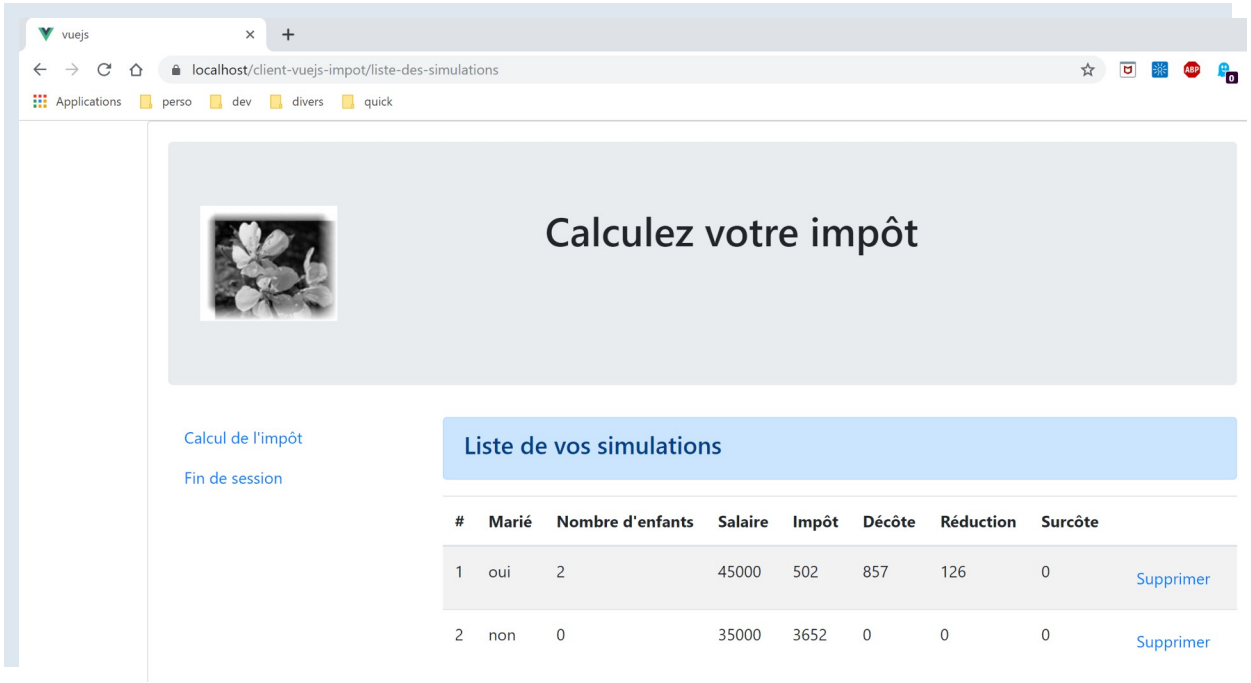


Maintenant une vue qui n'existe pas **[https://localhost/client-vuejs-impot/abcd]** :

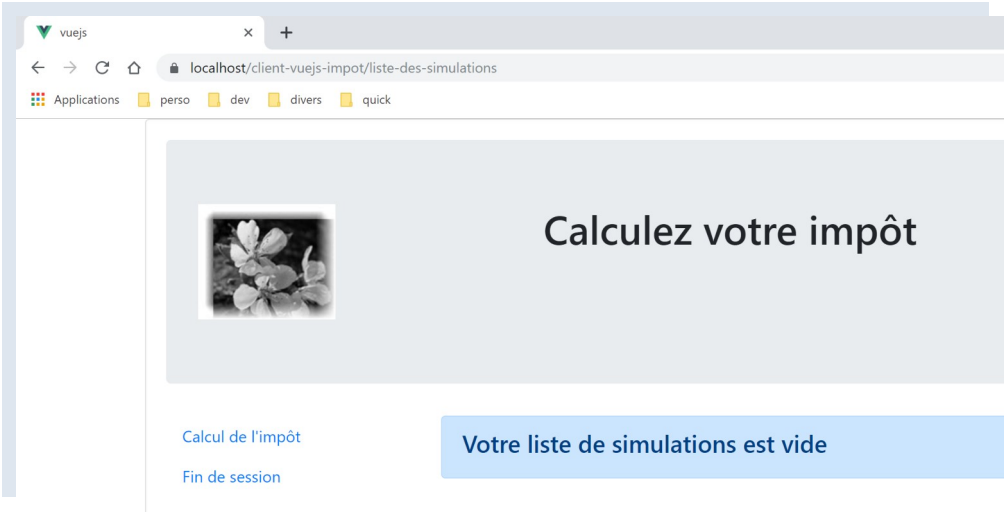


Notre application résiste plutôt bien aux URL tapées à la main. Lorsque celles-ci sont appelées, le routeur de l'application le sait. Il est donc possible d'intervenir avant que la vue ne soit finalement affichée. Nous allons regarder ce point dans le projet **[vuejs-21]**.

Un autre point à regarder est le suivant. Imaginons que l'utilisateur ait fait quelques simulations dans les règles :



Maintenant rafraîchissons la page par un F5 :



On a fait quelque chose de déconseillé : taper l'URL à la main (faire F5 revient à ça). Nous avons alors perdu nos simulations.

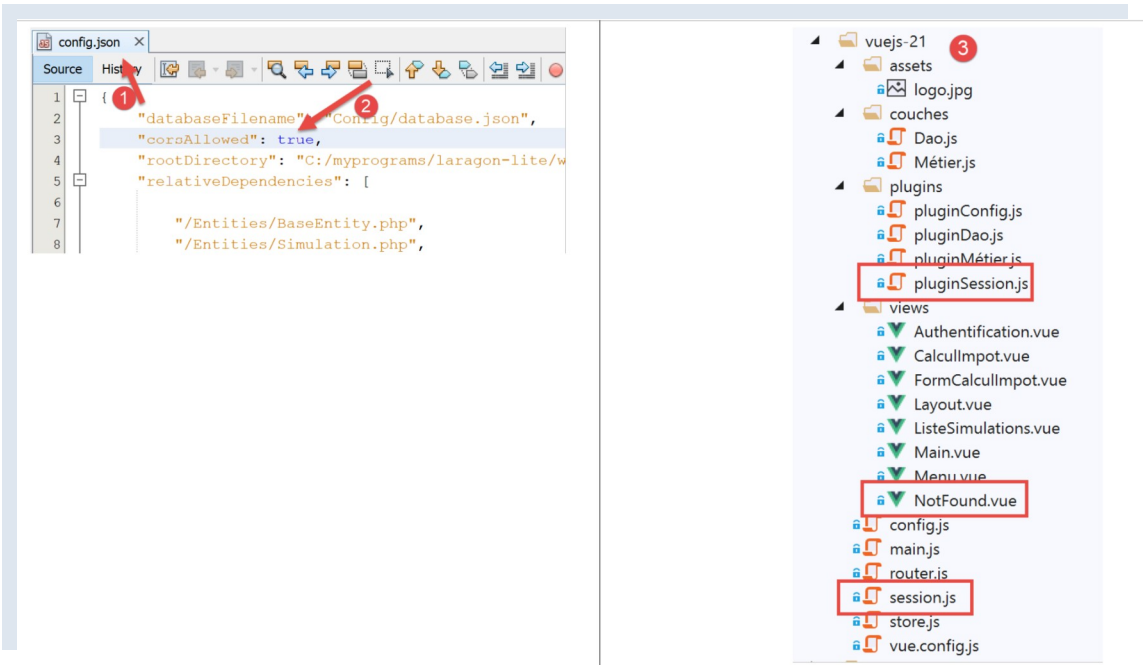
Le projet suivant **`[vuejs-21]`** se propose d'apporter deux améliorations :

- contrôler les URL tapées par l'utilisateur ;
- garder une mémoire de l'application même si l'utilisateur tape une URL. Ci-dessus, on voit qu'on a perdu la liste des simulations ;

19 Améliorations du client Vue.js

19.1 Introduction

Nous allons tester le projet **[vuejs-21]** avec le serveur de développement. Nous allons donc avoir besoin de nouveau que le serveur envoie les entêtes CORS. Il faut donc que le fichier **[config.json]** de la version 14 du serveur de calcul de l'impôt autorise ces entêtes :



Le projet **[vuejs-21]** est créé initialement par duplication du projet **[vuejs-20]**. Il est ensuite modifié **[3]**.

De nouveaux fichiers apparaissent :

- **[session.js]** : exporte un objet **[session]** qui va encapsuler des informations sur la session courante ;
- **[pluginSession]** : rend disponible l'objet **[session]** précédent dans la propriété **[$session]** des vues ;
- **[NotFound.vue]** : une nouvelle vue affichée lorsque l'utilisateur demande manuellement une URL qui n'existe pas ;

Des fichiers seront modifiés :

- **[main.js]** : va initialiser la session courante puis, lorsque l'utilisateur va taper manuellement des URL va la restaurer ;
- **[router.js]** : des contrôles sont ajoutés pour traiter le cas des URL tapées par l'utilisateur ;
- **[store.js]** : une nouvelle mutation est ajoutée ;
- **[config.js]** : une nouvelle configuration est ajoutée ;
- différentes vues essentiellement pour sauver la session courante à des moments clés de la vie de l'application. Celle-ci est ensuite restaurée à chaque fois que l'utilisateur tape manuellement des URL ;

19.2 Le store [Vuex]

Le script **[./store]** évolue de la façon suivante :

```
1.  // plugin Vuex
2.  import Vue from 'vue'
3.  import Vuex from 'vuex'
4.  Vue.use(Vuex);
5.
6.  // store Vuex
7.  const store = new Vuex.Store({
8.    state: {
9.      // le tableau des simulations
10.     simulations: [],
11.     // le n° de la dernière simulation
12.     idSimulation: 0
13.   },
```

```
14.   mutations: {
15.     // suppression ligne n° index
16.     deleteSimulation(state, index) {
17.       ...
18.     },
19.     // ajout d'une simulation
20.     addSimulation(state, simulation) {
21.       ...
22.     },
23.     // nettoyage state
24.     clear(state) {
25.       // plus de simulations
26.       state.simulations = [];
27.       // la numérotation des simulations repart de 0
28.       state.idSimulation = 0;
29.     }
30.   }
31. });
32. // export de l'objet [store]
33. export default store;
```

- lignes 24-29 : la mutation **[clear]** supprime la liste des simulations enregistrées et remet à 0 le n° de la dernière simulation.

19.3 La session

Le besoin d'une session vient du fait que lorsque l'utilisateur tape une URL dans le champ adresse du navigateur, le script **[main.js] est exécuté de nouveau**. Or celui-ci contient l'instruction :

```
1.  // store Vuex
2.  import store from './store'
```

Cette instruction importe le fichier **[./store]** suivant :

```
1.  // plugin Vuex
2.  import Vue from 'vue'
3.  import Vuex from 'vuex'
4.  Vue.use(Vuex);
5.
6.  // store Vuex
7.  const store = new Vuex.Store({
8.    state: {
9.      // le tableau des simulations
10.     simulations: [],
11.     // le n° de la dernière simulation
12.     idSimulation: 0
13.   },
14.   mutations: {
15.     ...
16.   }
17. });
18. // export de l'objet [store]
19. export default store;
```

On voit, lignes 7-13, qu'on importe un tableau de simulations vide. Si donc on avait des simulations avant que l'utilisateur ne tape une URL dans le champ adresse du navigateur, après on n'en a plus. L'idée est :

- d'utiliser une session qui stockerait les informations qu'on veut conserver si l'utilisateur tape manuellement des URL ;
- de la sauvegarder à des moments clés de l'application ;
- de la restaurer dans **[main.js]** qui **est toujours exécuté** lorsqu'une URL est tapée manuellement ;

Le script **[./session]** est le suivant :

```
1.  // on importe le store Vuex
2.  import store from './store'
3.  // on importe la configuration
4.  import config from './config';
5.
6.  // l'objet [session]
7.  const session = {
8.    // session démarrée
9.    started: false,
10.   // authentification
11.   authenticated: false,
12.   // heure de sauvegarde
```

```
13.   saveTime: "",
14.   // couche [métier]
15.   métier: null,
16.   // état Vuex
17.   state: null,
18.
19.   // sauvegarde de la session dans une chaîne jSON
20.   save() {
21.     // on ajoute à la session quelques propriétés
22.     this.saveTime = Date.now();
23.     this.state = store.state;
24.     // on la transforme en jSON
25.     const json = JSON.stringify(this);
26.     // on la stocke sur le navigateur
27.     localStorage.setItem("session", json);
28.     // eslint-disable-next-line no-console
29.     console.log("session save", json);
30.   },
31.
32.   // restauration de la session
33.   restore() {
34.     // on récupère la session jSON à partir du navigateur
35.     const json = localStorage.getItem("session")
36.     // si on a récupéré qq chose
37.     if (json) {
38.       // on restaure toutes les clés de la session
39.       const restore = JSON.parse(json);
40.       for (var key in restore) {
41.         if (restore.hasOwnProperty(key)) {
42.           this[key] = restore[key];
43.         }
44.       }
45.       // si on a dépassé une certaine durée d'inactivité depuis le début de la session, on repart de zéro
46.       let durée = Date.now() - this.saveTime;
47.       if (durée > config.duréeSession) {
48.         // on vide la session - elle sera également sauvegardée
49.         session.clear();
50.       } else {
51.         // on régénère le store Vuex
52.         store.replaceState(JSON.parse(JSON.stringify(this.state)));
53.       }
54.     }
55.     // eslint-disable-next-line no-console
56.     console.log("session restore", this);
57.   },
58.
59.     // on nettoie la session
60.   clear() {
61.     // eslint-disable-next-line no-console
62.     console.log("session clear");
63.     // raz de certains champs de la session
64.     this.authenticated = false;
65.     this.saveTime = "";
66.     this.started = false;
67.     if (this.métier) {
68.       // on réinitialise le champ [taxAdminData]
69.       this.métier.taxAdminData = null;
70.     }
71.     // le store Vuex est nettoyé également
72.     store.commit("clear");
73.     // on sauvegarde la nouvelle session
74.     this.save();
75.   },
76. }
77.
78. // export de l'objet [session]
79. export default session;
```

Commentaires

- ligne 2 : la session va encapsuler également le store **[Vuex]** (liste des simulations, n° de la dernière simulation faite) ;
- lignes 7-17 : les informations conservées par la session :
  - **[started]** : la session jSON avec le serveur a démarré ou non ;
  - **[authenticated]** : l'utilisateur s'est authentifié ou pas ;
  - **[saveTime]** : la date en millisecondes de la dernière sauvegarde ;

    - **[métier]** : une référence sur la couche **[métier]**. Celle-ci contient la donnée **[taxAdminData]** qui permet le calcul de l'impôt ;
    - **[state]** : le state du store **[Vuex]** (liste des simulations, n° de la dernière simulation faite) ;
- lignes 20-30 : la méthode **[save]** sauvegarde la session localement sur le navigateur exécutant l'application ;
    - ligne 22 : on note l'heure de sauvegarde ;
    - ligne 23 : on récupère le **[state]** du store **[Vuex]** ;
    - ligne 25 : on crée la chaîne jSON de la session ;
    - ligne 27 : on la stocke localement sur le navigateur associée à la clé **[session]** ;
- lignes 33-57 : la méthode **[restore]** permet de restaurer une session à partir de sa sauvegarde locale sur le navigateur ;
    - ligne 35 : on récupère la sauvegarde jSON locale ;
    - ligne 37 : si on a récupéré quelque chose ;
    - lignes 39-44 : l'objet **[session]** est reconstitué ;
    - ligne 46 : on calcule la durée qui nous sépare de la dernière sauvegarde ;
    - lignes 47-50 : si cette durée est supérieure à une valeur **[config.duréeSession]** fixée par configuration, la session est réinitialisée (ligne 49) et à cette occasion sauvegardée ;
    - ligne 52 : sinon on régénère l'attribut **[state]** du store **[Vuex]** ;
- lignes 60-75 : la méthode **[clear]** réinitialise la session ;
    - lignes 64-70 : les propriétés de la session sont réinitialisées à leurs valeurs initiales ;
    - ligne 72 : ainsi que le store **[Vuex]** ;
    - ligne 74 : la nouvelle session est sauvegardée ;

19.4 Le fichier de configuration [config]

Le fichier **[./config]** évolue de la façon suivante :

```
1.  // utilisation de la bibliothèque [axios]
2.  const axios = require('axios');
3.  // timeout des requêtes HTTP
4.  axios.defaults.timeout = 2000;
5.  ...
6.
7.  // export de la configuration
8.  export default {
9.    // objet [axios]
10.   axios: axios,
11.   // délai maximal d'inactivité de la session : 5 mn = 300 s = 300000 ms
12.   duréeSession: 300000
13. }
```

- ligne 12 : on va gérer la session de l'application un peu comme on gère une session web. On fixe ici une durée d'inactivité maximale de 5 minutes ;

19.5 Le plugin [pluginSession]

Comme il a été fait déjà de nombreuses fois, le plugin **[pluginSession]** va permettre aux vues d'avoir accès à la session via la propriété **[this.$session]** :

```
1.  export default {
2.    install(Vue, session) {
3.      // ajoute une propriété [$session] à la classe vue
4.      Object.defineProperty(Vue.prototype, '$session', {
5.        // lorsque Vue.$session est référencé, on rend le 2ième paramètre [session]
6.        get: () => session,
7.      })
8.    }
9.  }
```

19.6 Le script principal [main]

Le script principal **[./main.js]** évolue de la façon suivante :

```
1.  // log de démarrage
2.  // eslint-disable-next-line no-console
3.  console.log("main started");
4.
5.  // imports
6.  import Vue from 'vue'
7.
8.  ...
```

```
9.
10. // instanciation couche [métier]
11. import Métier from './couches/Métier';
12. const métier = new Métier();
13.
14. // plugin [métier]
15. import pluginMétier from './plugins/pluginMétier'
16. Vue.use(pluginMétier, métier)
17.
18. // store Vuex
19. import store from './store'
20.
21. // session
22. import session from './session';
23. import pluginSession from './plugins/pluginSession'
24. Vue.use(pluginSession, session)
25.
26. // on restore la session avant de redémarrer
27. session.restore();
28.
29. // on restaure la couche [métier]
30. if (session.métier && session.métier.taxAdminData) {
31.   métier.setTaxAdminData(session.métier.taxAdminData);
32. }
33.
34. // démarrage de l'UI
35. new Vue({
36.   el: '#app',
37.   // le routeur
38.   router: router,
39.   // le store Vuex
40.   store: store,
41.   // la vue principale
42.   render: h => h(Main),
43. })
44.
45. // log de fin
46. // eslint-disable-next-line no-console
47. console.log("main terminated, session=", session);
```

- ligne 19 : on importe la session ;
- ligne 20 : on importe son plugin ;
- ligne 21 : le plugin **[pluginSession]** est intégré à **[Vue]**. Après cette instruction toutes les vues disposent de la session dans leur attribut **[$session]** ;
- ligne 27 : la session est restaurée. La session importée ligne 11 est alors initialisée avec le contenu de sa dernière sauvegarde ;
- après la ligne 16, les vues disposent d'une propriété **[$métier]** initialisée ligne 12. Cette propriété n'a pas l'information **[taxAdminData]** qui permet de calculer l'impôt ;
- lignes 30-32 : si la restauration qui vient d'être faite a restauré la propriété **[session.métier.taxAdminData]** alors la propriété **[$métier]** des vues est initialisée avec cette valeur ;

19.7 Le fichier de routage [router]

Le fichier de routage **[./router]** évolue comme suit :

```
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: 'authentification', 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
```

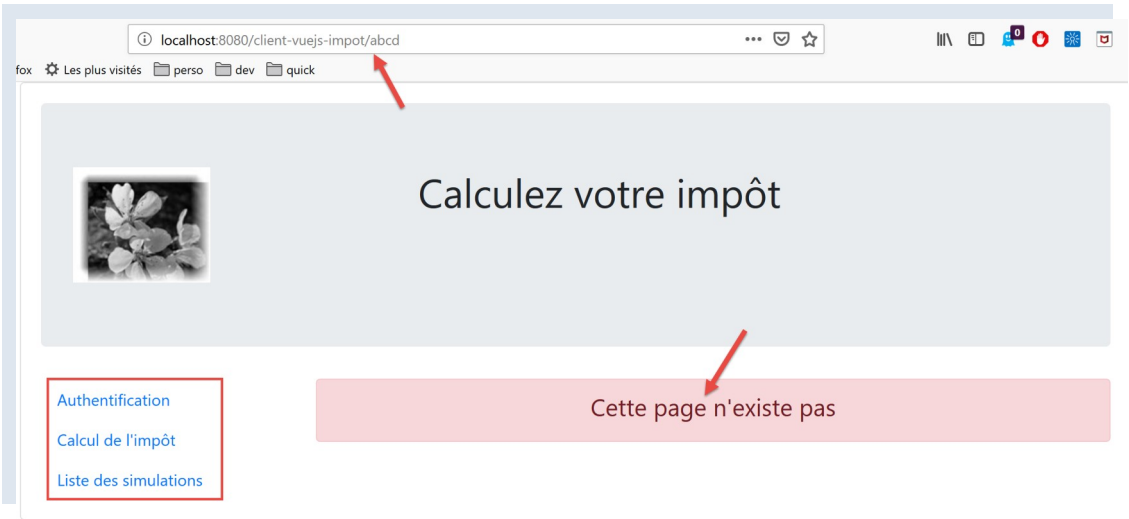
Commentaires

- lignes 16-38 : certaines routes ont été enrichies d'informations supplémentaires ;
- ligne 19 : on a créé une nouvelle route pour aller à la vue **`[Authentification]`** ;
- lignes 21-24 : la route qui mène à la vue **`[CalculImpot]`** a maintenant une propriété **`[meta]`** (ce nom est obligatoire). Le contenu de cet objet peut être quelconque et est fixé par le développeur ;
- ligne 23 : on met dans **`[meta]`**, la propriété **`[authenticated]`** (ce nom peut être quelconque). Il signifiera pour nous que pour aller à la vue **`[CalculImpot]`**, l'utilisateur doit être authentifié ;
- lignes 26-29 : on fait la même chose pour la route qui mène à la vue **`[ListeSimulations]`**. Là aussi, l'utilisateur doit être authentifié ;
- la propriété **`[meta.authenticated]`** va nous permettre de vérifier qu'un utilisateur qui tape manuellement les URL des vues **`[CalculImpot, ListeSimulations]`** ne peut pas les obtenir s'il n'est pas authentifié ;

- lignes 51-76 : la méthode **[beforeEach]** est exécutée avant qu'une vue ne soit routée. C'est le bon moment pour faire des vérifications ;
  - **[to]** : la prochaine route si on ne fait rien ;
  - **[from]** : la dernière route affichée ;
  - **[next]** : fonction permettant de changer la prochaine route affichée ;
- ligne 55 : on regarde si la prochaine route demande à ce que l'utilisateur soit authentifié ;
- lignes 56-59 : si oui et que l'utilisateur n'est pas authentifié, on change la prochaine route vers la vue **[Authentification]** ;
- lignes 64-73 : on traite le cas particulier de la route **[finSession]** des lignes 30-32. Celle-ci n'a pas de vue associée ;
  - ligne 66 : on réinitialise la session à sa valeur initiale ;
  - lignes 68-70 : on programme la vue **[Authentification]** comme prochaine vue ;
- ligne 75 : si on n'est pas dans les deux cas précédents, on se contente de passer à la route prévue par le fichier de routage ;
- lignes 35-37 : on prévoit une vue **[NotFound]** si la route tapée par l'utilisateur ne correspond à aucune route connue. Cette vue est importée ligne 8. Les routes sont vérifiées dans l'ordre du fichier de routage. Si donc on arrive à la ligne 36, c'est que la route demandée n'est aucune des routes des lignes 18-33 ;

19.8 La vue [NotFound]

La vue **[NotFound]** est affichée si la route tapée par l'utilisateur ne correspond à aucune route connue :



Le code de la 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="danger" align="center">
9.          <h4>Cette page n'existe pas</h4>
10.       </b-alert>
11.     </template>
12.     <!-- menu de navigation dans la colonne de gauche -->
13.     <Menu slot="left" :options="options" />
14.   </Layout>
15. </template>
16.
17. <script>
18. // imports
19. import Layout from "./Layout";
20. import Menu from "./Menu";
21. export default {
22.   // composants
23.   components: {
24.     Layout,
25.     Menu
26.   },
27.   // état interne du composant
28.   data() {
29.     return {
```

```
30.       // options du menu de navigation
31.       options: [
32.         {
33.           text: "Authentification",
34.           path: "/"
35.         }
36.       ]
37.     };
38.   },
39.   // cycle de vie
40.   created() {
41.     // eslint-disable-next-line
42.     console.log("NotFound created");
43.     // on regarde quelles options de menu offrir
44.     if (this.$session.authenticated && this.$métier.taxAdminData) {
45.       // l'utilisateur peut faire des simulations
46.       Array.prototype.push.apply(this.options, [
47.         {
48.           text: "Calcul de l'impôt",
49.           path: "/calcul-impot"
50.         },
51.         {
52.           text: "Liste des simulations",
53.           path: "/liste-des-simulations"
54.         }
55.       ]);
56.     }
57.   }
58. };
59. </script>
```

Commentaires

- ligne 4 : elle utilise les deux colonnes des vues routées ;
- lignes 6-11 : un message d'erreur ;
- ligne 13 : le menu de navigation occupe la colonne de gauche ;
- lignes 31-36 : les options par défaut du menu ;
- lignes 40-57 : code exécuté lorsque la vue est créée ;
- ligne 44 : on regarde si l'utilisateur peut faire des simulations ;
- lignes 45-55 : si oui, on ajoute deux options au menu de navigation, celles où il faut être authentifié et avoir une couche **`[métier]`** opérationnelle (lignes 46-55) ;

19.9 La vue [Authentification]

La vue **`[Authentification]`** évolue comme suit :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <Layout :left="false" :right="true">
4.      ...
5.    </Layout>
6.  </template>
7.
8.  <!-- dynamique de la vue -->
9.  <script>
10. import Layout from "./Layout";
11. export default {
12.   // état du composant
13.   data() {
14.     return {
15.       // utilisateur
16.       user: "",
17.       // son mot de passe
18.       password: "",
19.       // contrôle l'affichage d'un msg d'erreur
20.       showError: false,
21.       // le message d'erreur
22.       message: ""
23.     };
24.   },
25.
26.   // composants utilisés
27.   components: {
28.     Layout
```

```
29.   },
30.
31.   // propriétés calculées
32.   computed: {
33.     // saisies valides
34.     valid() {
35.       return this.user && this.password && this.$session.started;
36.     }
37.   },
38.
39.   // gestionnaires d'évts
40.   methods: {
41.     // ----------- authentification
42.     async login() {
43.       try {
44.         // début attente
45.         this.$emit("loading", true);
46.         // on n'est pas encore authentifié
47.         this.$session.authenticated = false;
48.         // authentification bloquante auprès du serveur
49.         const response = await this.$dao.authentifierUtilisateur(
50.           this.user,
51.           this.password
52.         );
53.         // fin du chargement
54.         this.$emit("loading", false);
55.         // analyse de la réponse du serveur
56.         if (response.état != 200) {
57.           // on affiche l'erreur
58.           this.message = response.réponse;
59.           this.showError = true;
60.           // retour à la boucle événementielle
61.           return;
62.         }
63.         // pas d'erreur
64.         this.showError = false;
65.         // on est authentifié
66.         this.$session.authenticated = true;
67.         // --------- on demande maintenant les données de l'administration fiscale
68.         // au départ, pas de donnée
69.         this.$métier.setTaxAdminData(null);
70.         // début attente
71.         this.$emit("loading", true);
72.         // demande bloquante auprès du serveur
73.         const response2 = await this.$dao.getAdminData();
74.         // fin du chargement
75.         this.$emit("loading", false);
76.         // analyse de la réponse
77.         if (response2.état != 1000) {
78.           // on affiche l'erreur
79.           this.message = response2.réponse;
80.           this.showError = true;
81.           // retour à la boucle événementielle
82.           return;
83.         }
84.         // pas d'erreur
85.         this.showError = false;
86.         // on mémorise dans la couche [métier] la donnée reçue
87.         this.$métier.setTaxAdminData(response2.réponse);
88.         // on peut passer au calcul de l'impôt
89.         this.$router.push({ name: "calculImpot" });
90.       } catch (error) {
91.         // on remonte l'erreur au composant principal
92.         this.$emit("error", error);
93.       } finally {
94.         // maj session
95.         this.$session.métier = this.$métier;
96.         // on sauvegarde la session
97.         this.$session.save();
98.       }
99.     }
100.  },
101.  // cycle de vie : le composant vient d'être créé
102.  created() {
103.    // eslint-disable-next-line
104.    console.log("Authentification created");
105.    // l'utilisateur peut-il faire des simulations ?
```

```
106.     if (
107.       this.$session.started &&
108.       this.$session.authenticated &&
109.       this.$métier.taxAdminData
110.     ) {
111.       // alors l'utilisateur peut faire des simulations
112.       this.$router.push({ name: "calculImpot" });
113.       // retour à la boucle événementielle
114.       return;
115.     }
116.     // si la session jSON a déjà été démarrée, on ne la redémarre pas de nouveau
117.     if (!this.$session.started) {
118.       // début attente
119.       this.$emit("loading", true);
120.       // on initialise la session avec le serveur - requête asynchrone
121.       // on utilise la promesse rendue par les méthodes de la couche [dao]
122.       this.$dao
123.         // on initialise une session jSON
124.         .initSession()
125.         // on a obtenu la réponse
126.         .then(response => {
127.           // fin attente
128.           this.$emit("loading", false);
129.           // analyse de la réponse
130.           if (response.état != 700) {
131.             // on affiche l'erreur
132.             this.message = response.réponse;
133.             this.showError = true;
134.             // retour à la boucle événementielle
135.             return;
136.           }
137.           // la session a démarré
138.           this.$session.started = true;
139.         })
140.         // en cas d'erreur
141.         .catch(error => {
142.           // on remonte l'erreur à la vue [Main]
143.           this.$emit("error", error);
144.         })
145.         // dans tous les cas
146.         .finally(() => {
147.           // on sauvegarde la session
148.           this.$session.save();
149.         });
150.     }
151.   }
152. };
153. </script>
```

Commentaires

- on a surligné en jaune les instructions qui utilisent la session introduite dans cette version du client **[Vue.js]** ;
- lignes 97, 148 : à la fin des méthodes **[login, created]**, la session est sauvegardée quelque soit le résultat des requêtes HTTP qui ont lieu dans ces méthodes (clause **[finally]** dans les deux cas) ;
- la méthode **[created]** des lignes 102-150 est exécutée à chaque fois que la vue **[Authentification]** est créée. Si c'est l'utilisateur qui a tapé l'URL de la vue, la session va nous permettre de savoir quoi faire ;
- lignes 106-115 : si la session jSON est démarrée, l'utilisateur authentifié et la donnée **[this.$métier.taxAdminData]** initialisée alors l'utilisateur peut directement aller au formulaire de calcul de l'impôt (ligne 112) ;
- ligne 117 : la méthode **[created]** était utilisée dans la version précédente pour initialiser une session jSON avec le serveur. Cette phase est inutile si elle a déjà eu lieu ;
- lignes 42-66 : la méthode d'authentification ;
- ligne 66 : si l'authentification réussit, on le note dans la session ;
- lignes 67-92 : la demande au serveur des données de l'administration fiscale **[taxAdminData]** ;
- ligne 95 : à la fin de cette phase, on met à jour la propriété **[métier]** de la session que l'opération ait réussi ou pas ;

19.10 La vue [CalculImpot]

Le code de la vue **[CalculImpot]** évolue comme suit :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    ...
4.  </template>
```

```
5.
6.  <script>
7.  // imports
8.  import FormCalculImpot from "./FormCalculImpot";
9.  import Menu from "./Menu";
10. import Layout from "./Layout";
11.
12. export default {
13.   // état interne
14.   data() {
15.     return {
16.       // options du menu
17.       options: [
18.         {
19.           text: "Liste des simulations",
20.           path: "/liste-des-simulations"
21.         },
22.         {
23.           text: "Fin de session",
24.           path: "/fin-session"
25.         }
26.       ],
27.       // résultat du calcul de l'impôt
28.       résultat: "",
29.       résultatObtenu: false
30.     };
31.   },
32.   // composants utilisés
33.   components: {
34.     Layout,
35.     FormCalculImpot,
36.     Menu
37.   },
38.   // méthodes de gestion des évts
39.   methods: {
40.     // résultat du calcul de l'impôt
41.     handleResultatObtenu(résultat) {
42.       // on construit le résultat en chaîne HTML
43.       ...
44.       // une simulation de +
45.       this.$store.commit("addSimulation", résultat);
46.       // on sauvegarde la session
47.       this.$session.save();
48.     }
49.   },
50.   // cycle de vie
51.   created() {
52.     // eslint-disable-next-line
53.     console.log("CalculImpot created");
54.   }
55. };
56. </script>
```

Commentaires

- ligne 45 : la simulation calculée est ajoutée au store **[Vuex]**. Cela a un impact sur la session qui englobe la propriété **[state]** du store. Aussi sauvegarde-t-on la session (ligne 47) ;
- ligne 51 : on crée une méthode **[created]** pour suivre dans les logs les créations des vues ;

19.11 La vue [ListeSimulations]

La vue **[ListeSimulations]** évolue comme suit :

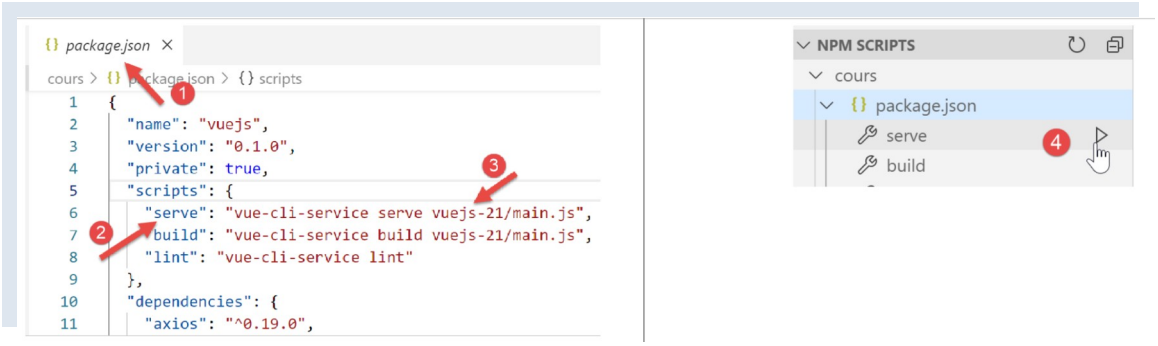
```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    ...
4.    </div>
5.  </template>
6.
7.  <script>
8.  // imports
9.  import Layout from "./Layout";
10. import Menu from "./Menu";
11. export default {
12.   // composants
```

```
13.   components: {
14.     Layout,
15.     Menu
16.   },
17.   // état interne
18.   data() {
19.     ...
20.   },
21.   // état interne calculé
22.   computed: {
23.     // liste des simulations prise dans le store Vuex
24.     simulations() {
25.       return this.$store.state.simulations;
26.     }
27.   },
28.   // méthodes
29.   methods: {
30.     supprimerSimulation(index) {
31.       // eslint-disable-next-line
32.       console.log("supprimerSimulation", index);
33.       // suppression de la simulation n° [index]
34.       this.$store.commit("deleteSimulation", index);
35.       // on sauvegarde la session
36.       this.$session.save();
37.     }
38.   },
39.   // cycle de vie
40.   created() {
41.     // eslint-disable-next-line
42.     console.log("ListeSimulations created");
43.   }
44. };
45. </script>
```

Commentaires

- ligne 36 : après la suppression d'une simulation ligne 34, on sauvegarde la session pour tenir compte de ce changement d'état ;
- lignes 40-43 : on continue à suivre la création des vues ;

19.12 Exécution du projet



Lors des tests vérifiez les points suivants :

- si l'utilisateur 'utilise' l'application via les liens du menu de navigation et les boutons / liens d'action, celle-ci fonctionne ;
- si l'utilisateur tape manuellement des URL, l'application continue à fonctionner. Faites en particulier le test suivant :
  - faites simulations ;
  - une fois sur la vue **[ListeSimulations]**, rechargez (F5) la vue. Dans l'application précédente **[vuejs-20]**, on perdait alors les simulations. Ici ce n'est pas le cas : on retrouve bien les simulations déjà faites ;
- regardez les logs pour comprendre :
  - à quel moment le script **[main]** est exécuté. Vous devez voir qu'il l'est à chaque fois que l'utilisateur tape une URL à la main ;
  - à quels moments les vues sont créées. Vous devez voir qu'elles le sont à chaque fois qu'elles vont être affichées ;
  - le fonctionnement du routage. Avant chaque routage un log est fait qui vous indique :
    - la route d'où vous venez ;
    - la route où vous allez ;

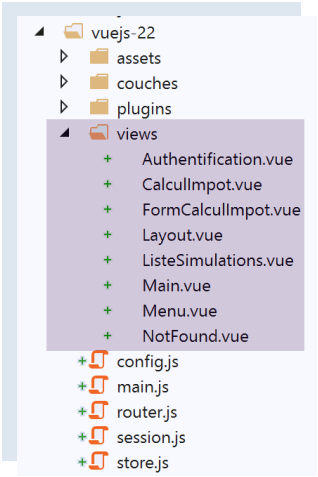
19.13 Déploiement de l'application sur un serveur local

Comme exercice, suivez le paragraphe |Déploiement sur un serveur local|, pour déployer le projet **[vuejs-21]** sur le serveur Laragon local. Puis testez-le.

19.14 Mise au point de la version mobile

Théoriquement, l'utilisation de Bootstrap devrait nous permettre d'avoir une application utilisable sur différents média : smartphone, tablette, ordinateurs portable et de bureau. Cequi différencie ces média c'est la taille de leur écran.

Si on teste la version **[vuejs-21]** sur un mobile, on constate que c'est le chaos dans l'affichage des vues. La version **[vuejs-22]** corrige ce point. Les modifications ont toutes lieu dans les templates des vues. Elles ont consisté essentiellement à mettre au point un affichage pour un écran de smartphone. Lorsque celui-ci est au point, l'affichage sur des écrans de taille plus importante se passe de façon fluide grâce à Bootstrap.



19.14.1 La vue [Main]

La vue **[Main]** évolue de la façon suivante :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <div class="container">
4.      <b-card>
5.        <!-- jumbotron -->
6.        <b-jumbotron>
7.          <b-row>
8.            <b-col sm="4">
9.              <img src="../assets/logo.jpg" alt="Cerisier en fleurs" />
10.           </b-col>
11.           <b-col sm="8">
12.             <h1>Calculez votre impôt</h1>
13.           </b-col>
14.         </b-row>
15.       </b-jumbotron>
16.       ....
17.     </b-card>
18.   </div>
19. </template>
```

Commentaires

- ligne 8 : là où il y avait **[cols='4']** on écrit **[sm='4']**. **[sm]** signifie **[small]**. Les écrans des smartphones tombent dans cette catégorie. Les autres catégories sont **[xs=extra small, md=medium, lg=large, xl=extra large]** ;
- ligne 11 : idem ;

19.14.2 La vue [Layout]

La vue **[Layout]** évolue comme suit :

```
<!-- définition HTML de la mise en page de la vue routée -->
<template>
  <!-- ligne -->
  <div>
    <b-row>
      <!-- zone de trois colonnes à gauche -->
      <b-col sm="3" v-if="left">
        <slot name="left" />
      </b-col>
      <!-- zone de neuf colonnes à droite -->
      <b-col sm="9" v-if="right">
        <slot name="right" />
      </b-col>
    </b-row>
  </div>
</template>
```

19.14.3 La vue [Authentification]

La vue **[Authentification]** évolue comme suit :

```
<!-- définition HTML de la vue -->
<template>
  <Layout :left="false" :right="true">
    <template slot="right">
      <!-- formulaire HTML - on poste ses valeurs avec l'action [authentifier-utilisateur] -->
      <b-form @submit.prevent="login">
        <!-- titre -->
        <b-alert show variant="primary">
          <h4>Bienvenue. Veuillez vous authentifier pour vous connecter</h4>
        </b-alert>
        <!-- 1ère ligne -->
        <b-form-group label="Nom d'utilisateur" label-for="user" description="Tapez admin">
          <!-- zone de saisie user -->
          <b-col sm="6">
            <b-form-input type="text" id="user" placeholder="Nom d'utilisateur" v-model="user" />
          </b-col>
        </b-form-group>
        <!-- 2ième ligne -->
        <b-form-group label="Mot de passe" label-for="password" description="Tapez admin">
          <!-- zone de saisie password -->
          <b-col sm="6">
            <b-input type="password" id="password" placeholder="Mot de passe" v-model="password" />
          </b-col>
        </b-form-group>
        <!-- 3ième ligne -->
        <b-alert
          show
          variant="danger"
          v-if="showError"
          class="mt-3"
        >L'erreur suivante s'est produite : {{message}}</b-alert>
        <!-- bouton de type [submit] sur une 3ième ligne -->
        <b-row>
          <b-col sm="2">
            <b-button variant="primary" type="submit" :disabled="!valid">Valider</b-button>
          </b-col>
        </b-row>
      </b-form>
    </template>
  </Layout>
</template>
```

Commentaires

- lignes 11 et 19 : on a supprimé l'attribut **[label-cols]** qui fixait un nombre de colonnes au label de la saisie. En l'absence de cet attribut, le label est au-dessus de la zone de saisie. Cela convient mieux aux écrans des smartphones ;

19.14.4 La vue [CalculImpot]

La vue **[CalculImpot]** évolue comme suit :

```
<!-- définition HTML de la vue -->
<template>
  <div>
```

```
      <Layout :left="true" :right="true">
        <!-- formulaire de calcul de l'impôt à droite -->
        <FormCalculImpot slot="right" @resultatObtenu="handleResultatObtenu" />
        <!-- menu de navigation à gauche -->
        <Menu slot="left" :options="options" />
      </Layout>
      <!-- zone d'affichage des résultat du calcul de l'impôt sous le formulaire -->
      <b-row v-if="résultatObtenu" class="mt-3">
        <!-- zone de trois colonnes vide -->
        <b-col sm="3" />
        <!-- zone de neuf colonnes -->
        <b-col sm="9">
          <b-alert show variant="success">
            <span v-html="résultat"></span>
          </b-alert>
        </b-col>
      </b-row>
    </div>
  </template>
```

19.14.5 La vue [FormCalculImpot]

La vue `[FormCalculImpot]` évolue comme suit :

```
  <!-- définition HTML de la vue -->
    <template>
    <!-- formulaire HTML -->
    <b-form @submit.prevent="calculerImpot" class="mb-3">
      <!-- message sur 12 colonnes sur fond bleu -->
      <b-row>
        <b-col sm="12">
          <b-alert show variant="primary">
            <h4>Remplissez le formulaire ci-dessous puis validez-le</h4>
          </b-alert>
        </b-col>
      </b-row>
      <!-- éléments du formulaire -->
      <!-- première ligne -->
      <b-form-group label="Etes-vous marié(e) ou pacsé(e) ?">
        <!-- boutons radio sur 5 colonnes-->
        <b-col sm="5">
          <b-form-radio v-model="marié" value="oui">Oui</b-form-radio>
          <b-form-radio v-model="marié" value="non">Non</b-form-radio>
        </b-col>
      </b-form-group>
      <!-- deuxième ligne -->
      <b-form-group label="Nombre d'enfants à charge" label-for="enfants">
        <b-form-input
          type="text"
          id="enfants"
          placeholder="Indiquez votre nombre d'enfants"
          v-model="enfants"
          :state="enfantsValide"
        ></b-form-input>
        <!-- message d'erreur éventuel -->
        <b-form-invalid-feedback :state="enfantsValide">Vous devez saisir un nombre positif ou nul</b-form-
    invalid-feedback>
      </b-form-group>
      <!-- troisème ligne -->
      <b-form-group
        label="Salaire annuel net imposable"
        label-for="salaire"
        description="Arrondissez à l'euro inférieur"
      >
        <b-form-input
          type="text"
          id="salaire"
          placeholder="Salaire annuel"
          v-model="salaire"
          :state="salaireValide"
        ></b-form-input>
        <!-- message d'erreur éventuel -->
        <b-form-invalid-feedback :state="salaireValide">Vous devez saisir un nombre positif ou nul</b-form-
    invalid-feedback>
      </b-form-group>
      <!-- quatrième ligne, bouton [submit] -->
      <b-col sm="3">
```

```
52.         <b-button type="submit" variant="primary" :disabled="formInvalide">Valider</b-button>
53.       </b-col>
54.     </b-form>
55.   </template>
```

Commentaires

- lignes 15, 23, 35 : on a supprimé l'attribut `[label-cols]` ;

Par ailleurs, on fait évoluer les tests de validité :

```
1.   ...
2.   // état interne calculé
3.     computed: {
4.       // validation du formulaire
5.       formInvalide() {
6.         return (
7.           // salaire invalide
8.           !this.salaire.match(/^\s*\d+\s*$/) ||
9.           // ou enfants invalide
10.          !this.enfants.match(/^\s*\d+\s*$/) ||
11.          // ou données fiscales pas obtenues
12.          !this.$métier.taxAdminData
13.        );
14.      },
15.      // validation du salaire
16.      salaireValide() {
17.        // doit être numérique >=0
18.        return Boolean(
19.          this.salaire.match(/^\s*\d+\s*$/) || this.salaire.match(/^\s*$/)
20.        );
21.      },
22.      // validation des enfants
23.      enfantsValide() {
24.        // doit être numérique >=0
25.        return Boolean(
26.          this.enfants.match(/^\s*\d+\s*$/) || this.enfants.match(/^\s*$/)
27.        );
28.      }
29.  },
30.  ...
```

Commentaires

- ligne 19 : lorsque rien n'a été saisi, la saisie est considérée comme valide. Cela permet d'avoir une saisie valide lorsque la vue est initialement affichée. Dans la version précédente, la saisie apparaissait initialement comme erronée ;
- ligne 26 : idem ;
- lignes 5-14 : le bouton de validation n'est actif que si les deux saisies contiennent quelque chose et sont valides ;

19.14.6 La vue `[Menu]`

La vue `[Menu]` évolue comme suit :

```
1.   <!-- définition HTML de la vue -->
2.   <template>
3.     <b-card class="mb-3">
4.       <!-- menu Bootstrap vertical -->
5.       <b-nav vertical>
6.         <!-- options du menu -->
7.         <b-nav-item
8.           v-for="(option,index) of options"
9.           :key="index"
10.          :to="option.path"
11.          exact
12.          exact-active-class="active"
13.        >{{option.text}}</b-nav-item>
14.      </b-nav>
15.    </b-card>
16.  </template>
```

Commentaires

- ligne 3 : on ajoute la balise <b-card> pour entourer le menu d'une fine bordure. Cela permet de mieux localiser le menu sur le smartphone ;

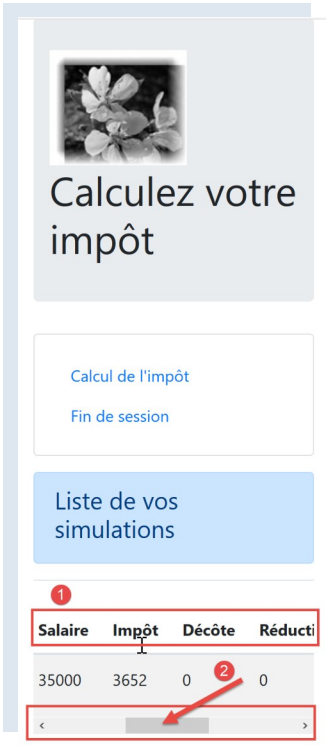
19.14.7 La vue [ListeSimulations]

La vue `[ListeSimulations]` reste inchangée :

```
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 striped hover responsive :items="simulations" :fields="fields">
21.             <template v-slot:cell(action)="data">
22.               <b-button variant="link" @click="supprimerSimulation(data.index)">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>
```

Commentaires

- ligne 20 : on notera l'attribut `[responsive]` qui fait que l'affichage de la table s'adapte à la taille de l'écran :

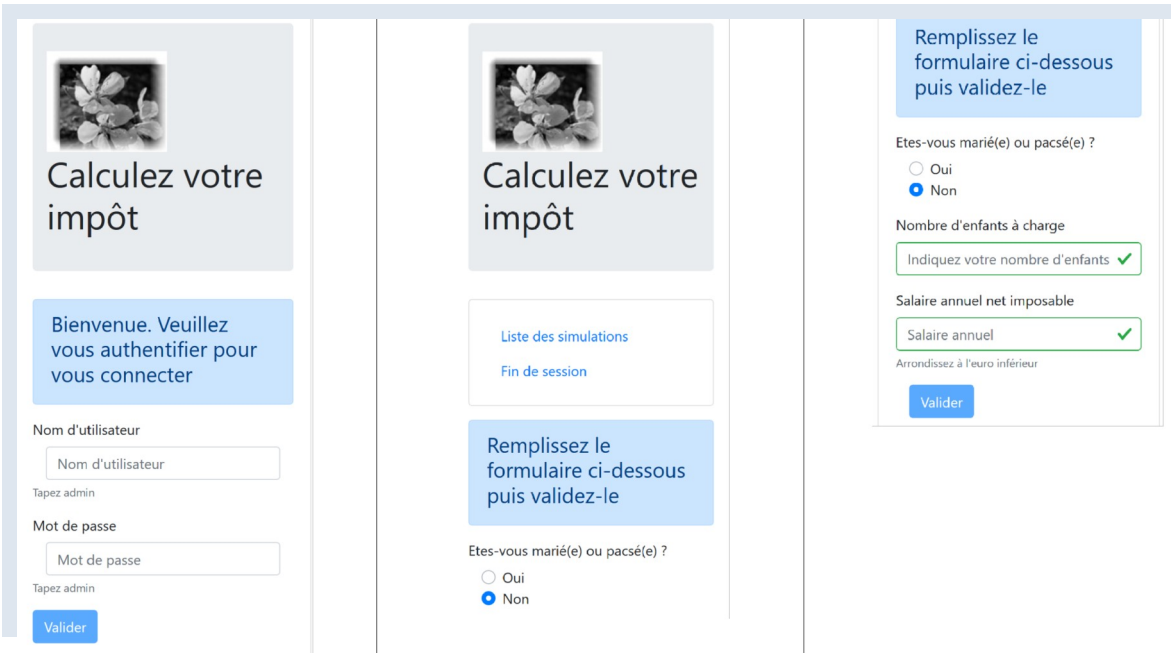


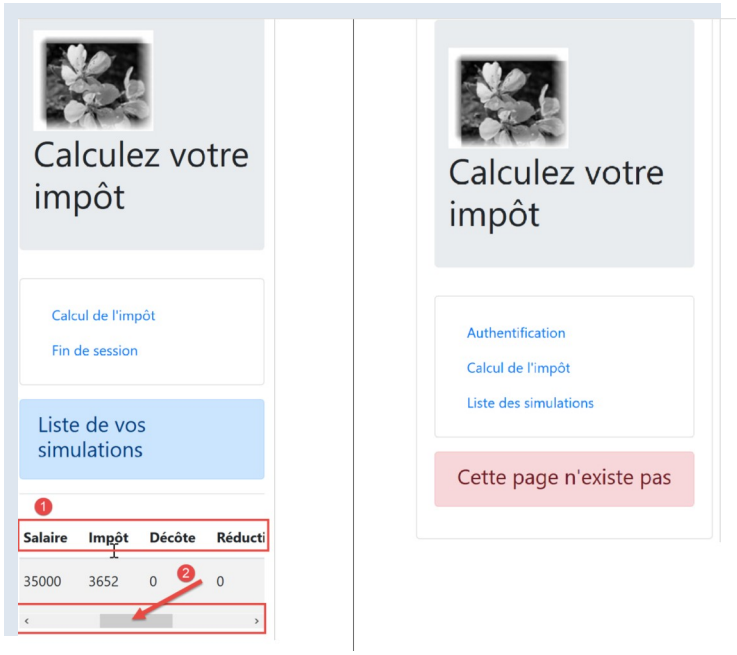
- en [2], sur les petits écrans, une barre de défilement horizontal permet d'afficher la table ;

19.14.8 La vue [NotFound]

Elle reste inchangée.

19.14.9 Les vues sur mobile





Note : il y a sûrement possibilité d'obtenir des vues encore mieux adaptées au mobile. Je pense notamment au menu de navigation qui pourrait être amélioré mais il y a d'autres points. Ce document n'avait pas pour objectif premier la création d'une application mobile. Dans ce cas, on se serait peut-être tourné vers un framework comme Ionic |https://ionicframework.com/|.

20 Déploiement de l'application client / serveur sur un service d'hébergement

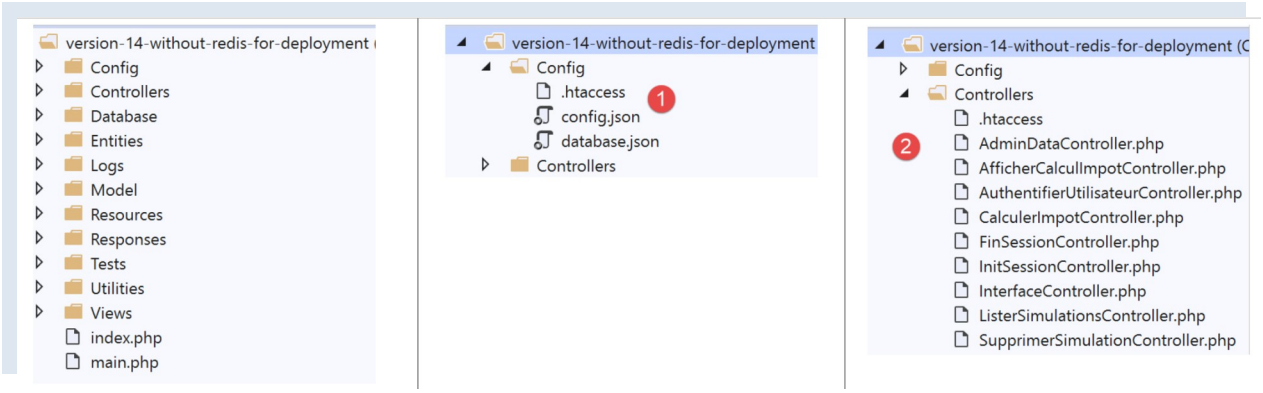
Nous donnons ici les grandes lignes du déploiement de l'application client / serveur que nous avons développée sur un serveur OVH **[https://www.ovh.com/fr/]**. Le déploiement sur d'autres fournisseurs d'hébergement ne devrait pas être très différent. On veut simplement montrer que notre application se prête bien à ce déploiement.

20.1 Déploiement du serveur

L'hébergement OVH visé est un hébergement basique :

1. un environnement PHP 7.3 ;
2. un SGBD MySQL ;
3. pas de serveur **[Redis]** ;

Le 3ième point nous oblige à modifier la version 14 de notre serveur de calcul de l'impôt.



Il nous faut modifier :

- les fichiers de configuration **[1]** ;
- les contrôleurs **[AdminDataController]** et **[CalculerImpotController]** pour tenir compte du fait qu'il n'y a pas de serveur **[Redis]** ;

Le fichier **[config.json]** évolue de la façon suivante :

```
1.  {
2.      "databaseFilename": "Config/database.json",
3.      "corsAllowed": false,
4.      "redisAvailable":false,
5.      "rootDirectory": "/.../www/apps/impot/serveur-php7",
6.      "relativeDependencies": [
7.
8.          "/Entities/BaseEntity.php",
9.          ...
10.         "/Controllers/AdminDataController.php"
11.     ],
12.     "absoluteDependencies": [
13.         "/.../vendor/autoload.php",
14.         "/.../vendor/predis/predis/autoload.php"
15.     ],
16.     "users": [
17.         {
18.             "login": "admin",
19.             "passwd": "admin"
20.         }
21.     ],
22.     ...
23. }
```

Commentaires

- ligne 4 : on introduit un booléen **[redisAvailable]** pour indiquer si on a accès ou non à un serveur **[Redis]** ;
- lignes 5, 13, 14 : les chemins absolus vont changer ;

Le fichier **[database.json]** évolue comme suit :

```
1.  {
2.      "dsn": "mysql:host=...;dbname=...",
3.      "id": "...",
4.      "pwd": "...",
5.      "tableTranches": "dbimpots_tbtranches",
6.      "colLimites": "limites",
7.      "colCoeffR": "coeffr",
8.      "colCoeffN": "coeffn",
9.      "tableConstantes": "dbimpots_tbconstantes",
10.     "colPlafondQfDemiPart": "plafondQfDemiPart",
11.     ...
12. }
```

Commentaires

- lignes 2-4 : l'identité de la base de données ainsi que les identifiants de son propriétaire vont changer ;

Le contrôleur **[AdminDataController]** évolue de la façon suivante :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
6.  use \Symfony\Component\HttpFoundation\Response;
7.  use \Symfony\Component\HttpFoundation\Request;
8.  use \Symfony\Component\HttpFoundation\Session\Session;
9.  // alias de la couche [dao]
10. use \Application\ServerDaoWithSession as ServerDaoWithRedis;
11.
12. class AdminDataController implements InterfaceController {
13.
14.   // $config est la configuration de l'application
15.   // traitement d'une requête Request
16.   // utile la session Session et peut la modifier
17.   // $infos sont des informations supplémentaires propres à chaque contrôleur
18.   // rend un tableau [$statusCode, $état, $content, $headers]
19.   public function execute(
20.     array $config,
21.     Request $request,
22.     Session $session,
23.     array $infos = NULL): array {
24.
25.     // on doit avoir un unique paramètre GET
26.     $method = strtolower($request->getMethod());
27.     ...
28.
29.     // on peut travailler
30.     // Redis
31.     if ($config["redisAvailable"]) {
32.       \Predis\Autoloader::register();
33.       ...
34.     } else {
35.       try {
36.         // on va chercher les données fiscales en base de données
37.         $dao = new ServerDaoWithRedis($config["databaseFilename"], NULL);
38.         // taxAdminData
39.         $taxAdminData = $dao->getTaxAdminData();
40.       } catch (\Throwable $ex) {
41.         // ça s'est mal passé
42.         // retour résultat avec erreur au contrôleur principal
43.         $état = 1051;
44.         return [Response::HTTP_INTERNAL_SERVER_ERROR, $état,
45.           ["réponse" => utf8_encode($ex->getMessage())], []];
46.       }
47.     }
48.
49.     // retour résultat au contrôleur principal
50.     $état = 1000;
51.     return [Response::HTTP_OK, $état, ["réponse" => $taxAdminData], []];
52.   }
53.
```

```
54.  }
```

Commentaires

- ligne 31 : on teste désormais si on a ou non un serveur **[Redis]** ;
- lignes 32-34 : si oui, le code précédent est repris dans son intégralité ;
- lignes 35-46 : sinon, les données de l'administration fiscale sont prises dans la base de données ;

Le contrôleur **[CalculerImpotController]** qui lui également a besoin des données de l'administration fiscale évolue de façon identique.

Ceci fait. Le déploiement sur le serveur OVH a consisté à faire du FTP. On a téléchargé sur OVH :

- la version **[vuejs-14-without-redis]** ;
- le dossier **[vendor]** qui contient toutes les dépendances du serveur **[vuejs-14-without-redis]** ;

Le transfert FTP fait, on a généré les tables nécessaires au serveur avec le script SQL suivant :

```
1.   -- phpMyAdmin SQL Dump
2.   -- version 4.8.5
3.   -- https://www.phpmyadmin.net/
4.   --
5.   -- Host: localhost:3306
6.   -- Generation Time: Oct 12, 2019 at 07:45 AM
7.   -- Server version: 5.7.24
8.   -- PHP Version: 7.2.11
9.
10.  SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
11.  SET AUTOCOMMIT = 0;
12.  START TRANSACTION;
13.  SET time_zone = "+00:00";
14.
15.
16.  /*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
17.  /*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
18.  /*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
19.  /*!40101 SET NAMES utf8mb4 */;
20.
21.  --
22.  -- Table structure for table `dbimpots_tbconstantes`
23.  --
24.
25.  CREATE TABLE `dbimpots_tbconstantes` (
26.    `id` int(11) NOT NULL,
27.    `plafondQfDemiPart` decimal(10,2) NOT NULL,
28.    `plafondRevenusCelibatairePourReduction` decimal(10,2) NOT NULL,
29.    `plafondRevenusCouplePourReduction` decimal(10,2) NOT NULL,
30.    `valeurReducDemiPart` decimal(10,2) NOT NULL,
31.    `plafondDecoteCelibataire` decimal(10,2) NOT NULL,
32.    `plafondDecoteCouple` decimal(10,2) NOT NULL,
33.    `plafondImpotCelibatairePourDecote` decimal(10,2) NOT NULL,
34.    `plafondImpotCouplePourDecote` decimal(10,2) NOT NULL,
35.    `abattementDixPourcentMax` decimal(10,2) NOT NULL,
36.    `abattementDixPourcentMin` decimal(10,2) NOT NULL
37.  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
38.
39.  --
40.  -- Dumping data for table `dbimpots_tbconstantes`
41.  --
42.
43.  INSERT INTO `dbimpots_tbconstantes` (`id`, `plafondQfDemiPart`, `plafondRevenusCelibatairePourReduction`,
     `plafondRevenusCouplePourReduction`, `valeurReducDemiPart`, `plafondDecoteCelibataire`,
     `plafondDecoteCouple`, `plafondImpotCelibatairePourDecote`, `plafondImpotCouplePourDecote`,
     `abattementDixPourcentMax`, `abattementDixPourcentMin`) VALUES
44.  (8, '1551.00', '21037.00', '42074.00', '3797.00', '1196.00', '1970.00', '1595.00', '2627.00', '12502.00',
     '437.00');
45.
46.  -- --------------------------------------------------------
47.
48.  --
49.  -- Table structure for table `dbimpots_tbtranches`
50.  --
51.
52.  CREATE TABLE `dbimpots_tbtranches` (
53.    `id` int(11) NOT NULL,
```

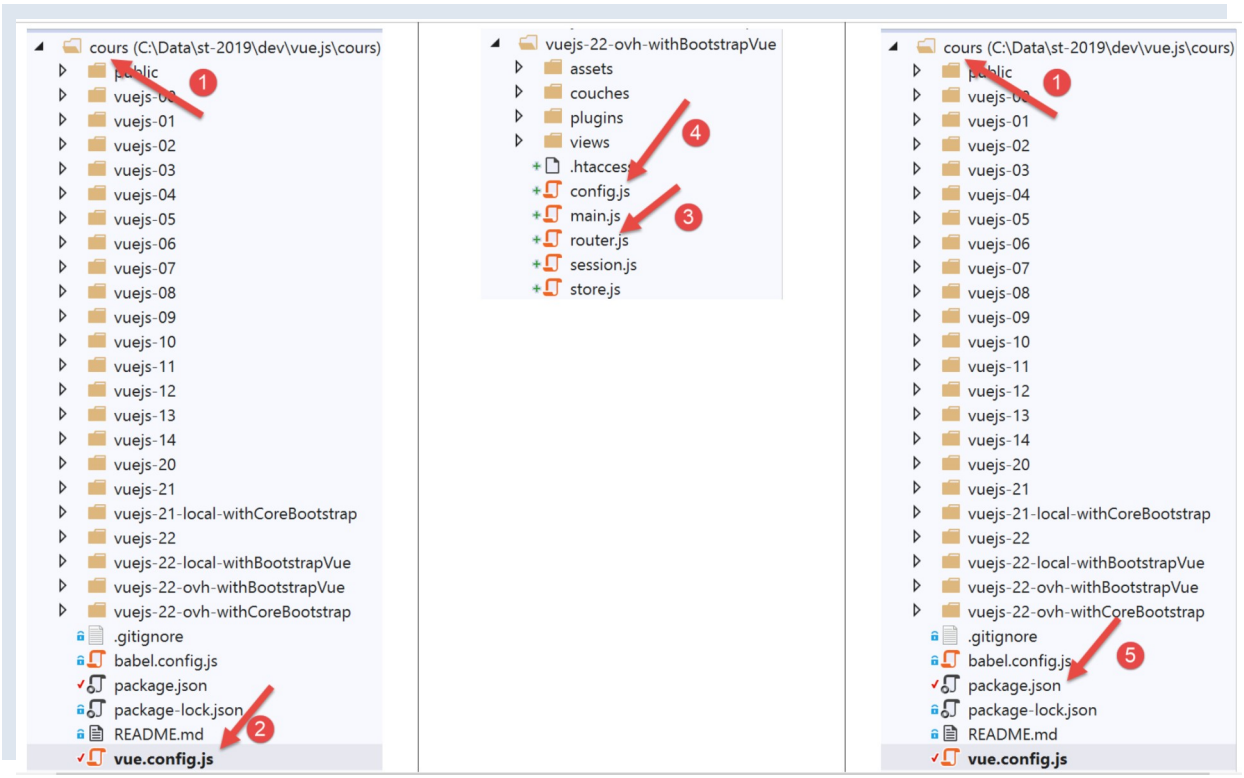
```
54.    `limites` decimal(10,2) NOT NULL,
55.    `coeffR` decimal(10,2) NOT NULL,
56.    `coeffN` decimal(10,2) NOT NULL
57.  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
58.
59.  --
60.  -- Dumping data for table `dbimpots_tbtranches`
61.  --
62.
63.  INSERT INTO `dbimpots_tbtranches` (`id`, `limites`, `coeffR`, `coeffN`) VALUES
64.  (36, '9964.00', '0.00', '0.00'),
65.  (37, '27519.00', '0.14', '1394.96'),
66.  (38, '73779.00', '0.30', '5798.00'),
67.  (39, '156244.00', '0.41', '13913.69'),
68.  (40, '0.00', '0.45', '20163.45');
69.
70.  --
71.  -- Indexes for dumped tables
72.  --
73.
74.  --
75.  -- Indexes for table `dbimpots_tbconstantes`
76.  --
77.  ALTER TABLE `dbimpots_tbconstantes`
78.    ADD PRIMARY KEY (`id`);
79.
80.  --
81.  -- Indexes for table `dbimpots_tbtranches`
82.  --
83.  ALTER TABLE `dbimpots_tbtranches`
84.    ADD PRIMARY KEY (`id`);
85.
86.  --
87.  -- AUTO_INCREMENT for dumped tables
88.  --
89.
90.  --
91.  -- AUTO_INCREMENT for table `dbimpots_tbconstantes`
92.  --
93.  ALTER TABLE `dbimpots_tbconstantes`
94.    MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=9;
95.
96.  --
97.  -- AUTO_INCREMENT for table `dbimpots_tbtranches`
98.  --
99.  ALTER TABLE `dbimpots_tbtranches`
100.   MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=41;
101. COMMIT;
102.
103. /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
104. /*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
105. /*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

Lorsque tout ceci a été fait, on a adapté les fichiers **[config.json, database.json]** à leur nouvel environnement.

20.2 Déploiement du client [Vue.js]

Il a été décidé de déployer le client **[Vue.js]** à l'URL **[http://machine/apps/impot/client-vuejs/]**. Cela a entraîné les modifications suivantes :

A la racine du **[workspace]** de **[VSCode]** on a créé le fichier **[vue.config.js]** suivant :



Le fichier **[vue.config.js]** est le suivant :

```
1.  // vue.config.js
2.  module.exports = {
3.    // l'URL de service du client [vuejs] du serveur de calcul de l'impôt
4.    publicPath: '/apps/impot/client-vuejs/'
5.  }
```

Le fichier **[router.js] [3]** a été également modifié :

```
1.  // imports
2.  import Vue from 'vue'
3.  import VueRouter from 'vue-router'
4.  ...
5.
6.  // plugin de routage
7.  Vue.use(VueRouter)
8.
9.  // les routes de l'application
10. const routes = [
11.   ...
12. ]
13.
14. // le routeur
15. const router = new VueRouter({
16.   // les routes
17.   routes,
18.   // le mode d'affichage des URL
19.   mode: 'history',
20.   // l'URL de base de l'application
21.   base: '/apps/impot/client-vuejs/'
22. })
23.
24. // vérification des routes
25. router.beforeEach((to, from, next) => {
26.   ...
27. })
28.
29. // export du router
30. export default router
```

Commentaires

- ligne 21 : la base des URL a été modifiée ;

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

```
1.  // utilisation de la bibliothèque [axios]
2.  const axios = require('axios');
3.  // timeout des requêtes HTTP
4.  axios.defaults.timeout = 5000;
5.  // la base des URL du serveur de calcul de l'impôt
6.  // le schéma [https] pose des problèmes à Firefox parce que le serveur de calcul
7.  // de l'impôt envoie un certificat autosigné. ok avec Chrome et Edge. Safari pas testé.
8.  // avec Firefox c'est possible en demandant l'URL ci-dessous directement et en disant à Firefox
9.  // que vous acceptez le risque d'un certificat non signé. Ensuite le client [vuejs] fonctionnera.
10. axios.defaults.baseURL = 'http://.../apps/impot/serveur-php7';
11. // on va utiliser des cookies
12. axios.defaults.withCredentials = true;
13.
14. // export de la configuration
15. export default {
16.   // objet [axios]
17.   axios: axios,
18.   // délai maximal d'inactivité de la session : 5 mn = 300 s = 300000 ms
19.   duréeSession: 300000
20. }
```

Commentaires

- ligne 10 : on met l'URL du serveur de calcul de l'impôt ;

La version de production du projet a été générée avec la commande **[build]** du fichier **[package.json] [5]** suivant :

```
1.  {
2.    "name": "vuejs",
3.    "version": "0.1.0",
4.    "private": true,
5.    "scripts": {
6.      "serve": "vue-cli-service serve vuejs-22/main.js",
7.      "build": "vue-cli-service build vuejs-22-ovh-withBootstrapVue/main.js",
8.      "lint": "vue-cli-service lint"
9.    },
10.   ...
11. }
```

Ceci fait, le dossier **[dist]** qui contenait la version de production générée a été 'uploadée' sur le serveur OVH dans le dossier **[/.../apps/impot]** puis renommé **[client-vuejs]** pour que le code du client soit dans le dossier **[/.../apps/impot/client-vuejs/]** comme il était prévu. Puis dans ce dossier nous avons téléchargé le fichier **[.htaccess]** suivant :

```
1.  <IfModule mod_rewrite.c>
2.    RewriteEngine On
3.    RewriteBase /apps/impot/client-vuejs/
4.    RewriteRule ^index\.html$ - [L]
5.    RewriteCond %{REQUEST_FILENAME} !-f
6.    RewriteCond %{REQUEST_FILENAME} !-d
7.    RewriteRule . /apps/impot/client-vuejs/index.html [L]
8.  </IfModule>
```

ceci parce que le serveur web d'OVH utilisé ici est un serveur Apache. Pour d'autres types de serveurs, on se reportera à la documentation |https://cli.vuejs.org/guide/deployment.html|.

L'application serveur PHP 7 peut être testée |**ici**|.
Le client [Vue.js] peut être testé |**ici**|.

20.3 Conclusion

La version **[vuejs-21]** n'était pas indispensable. On avait vu que la version **[vuejs-20]** résistait correctement aux URL tapées par l'utilisateur. Néanmoins la nouvelle version amène un confort supplémentaire à l'utilisateur. Il peut naviguer en tapant des URL. L'application lui propose alors la vue qui convient le mieux à l'état actuel (la session) de l'application. Par ailleurs, la version **[vuejs-22]** amène des améliorations pour l'affichage de l'application sur mobiles.