Introduction au langage ECMASCRIPT 6 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 langage ECMASCRIPT 6

Le PDF de cet article 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]. **C'est le document présent** ;
3. [Introduction au framework VUE.JS par l'exemple] ;
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** :

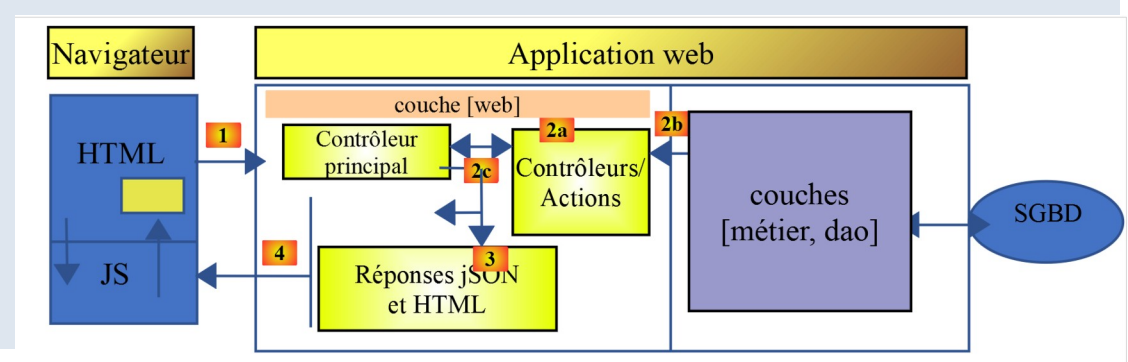
- le document [1] présente le langage PHP 7. Le lecteur seulement intéressé par le langage PHP et pas par le langage Javascript des articles suivants s'arrêtera là ;
- les documents [2-4] visent à construire un client Javascript au serveur de calcul de l'impôt développé dans le document [1] ;
- les frameworks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

- la version écrite est centrée sur le serveur. La tendance est désormais (juillet 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, fournis par le même serveur qui a fourni la page d'accueil ou bien entièrement construite par le Javascript ;
    - 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 code javascript est client :

- d'un service de pages ou fragments statiques ou non ;
- d'un service 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.

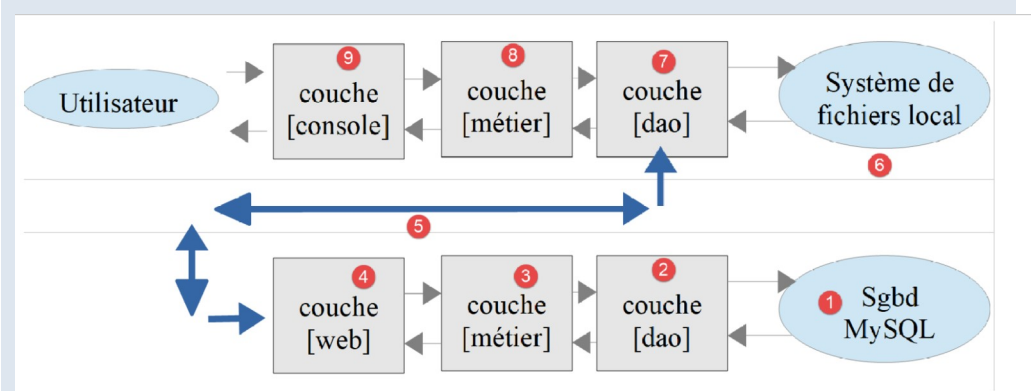
Ce type d'application fait également partie des applications dites **AJAX** : **A**synchronous **J**avascript **A**nd **X**ML
- **Asynchronous** : parce que les appels du client Javascript au serveur jSON sont asynchrones ;
- **XML** : parce que XML était la technologie utilisée avant l'avènement du jSON. On a cependant gardé l'acronyme AJAX ;

Nous allons étudier une telle architecture dans les chapitres à venir. 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. Pour le comprendre, il faut maîtriser ce langage. Nous présentons dans ce document la norme ECMAScript 6 qui est la normalisation la plus récente (en 2019) de ce langage. On trouvera l'historique et le rôle d'ECMAScript sur Wikipedia `[https://fr.wikipedia.org/wiki/ECMAScript]`.

Ce document propose une liste de **scripts console** Javascript dans différents domaines (structures du langage, accès aux bases de données, au réseau internet, programmation en couches, programmation par interfaces). Le document se termine avec deux applications :

Une application console qui sera un client du serveur de calcul de l'impôt construit dans le document [1]. Ce client aura la même architecture que celle du client console PHP construit dans le document [1] :



- les couches [7-9] seront celles du client Javascript s'exécutant dans une console ;
- les couches [1-4] sont celles du serveur PHP 7 construit dans le document [1] ;
- contrairement au client PHP console construit dans le document [1], il n'y aura pas d'interactions avec le système de fichiers local [6] ;

On a là une application client / serveur où le client est une application console. Une seconde application sera écrite où le code des couches [7-9] **sera porté dans un navigateur**. Nous serons alors prêts à aborder les frameworks Javascript des navigateurs dans les documents [3] et [4].

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 Installation d'un environnement de travail

Nous allons utiliser les outils suivants (sous Windows 10 x 64 bits) :

- [Laragon] pour exécuter le serveur web PHP ;
- [Netbeans] pour modifier le code PHP ;
- [Visual Studio Code] pour écrire les codes Javascript ;
- `[node.js]` pour les exécuter ;
- `[npm]` pour télécharger et installer les bibliothèques Javascript dont nous aurons besoin ;

2.1 Environnement de travail pour le serveur web

Les scripts PHP ont été écrits et testés dans l'environnement suivant :

- un environnement serveur web Apache / SGBD MySQL / PHP 7.3 appelé **Laragon ;**
- l'IDE de développement **Netbeans 10.0 ;**

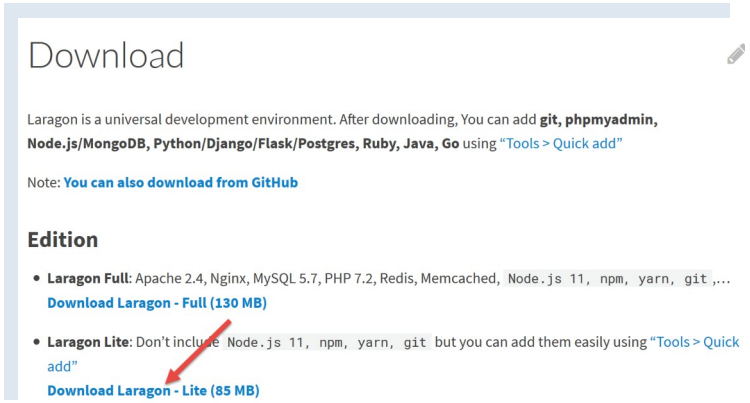
2.1.1 Installation de Laragon

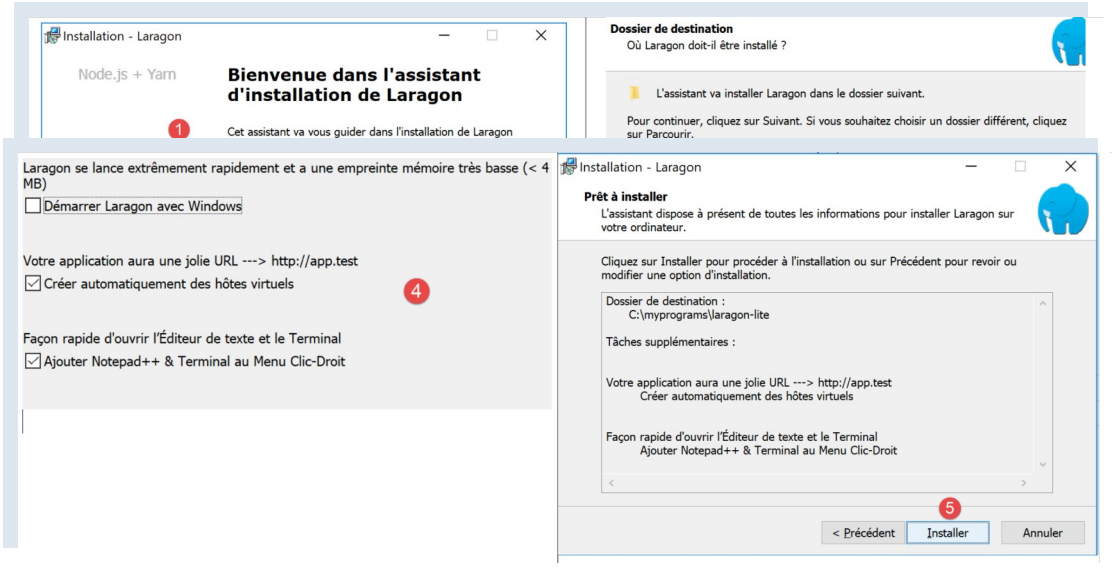
Laragon est un package réunissant plusieurs logiciels :

- un serveur web Apache. Nous l'utiliserons pour l'écriture de scripts web en PHP ;
- le SGBD MySQL ;
- le langage de script PHP ;
- un serveur Redis implémentant un cache pour des applications web :

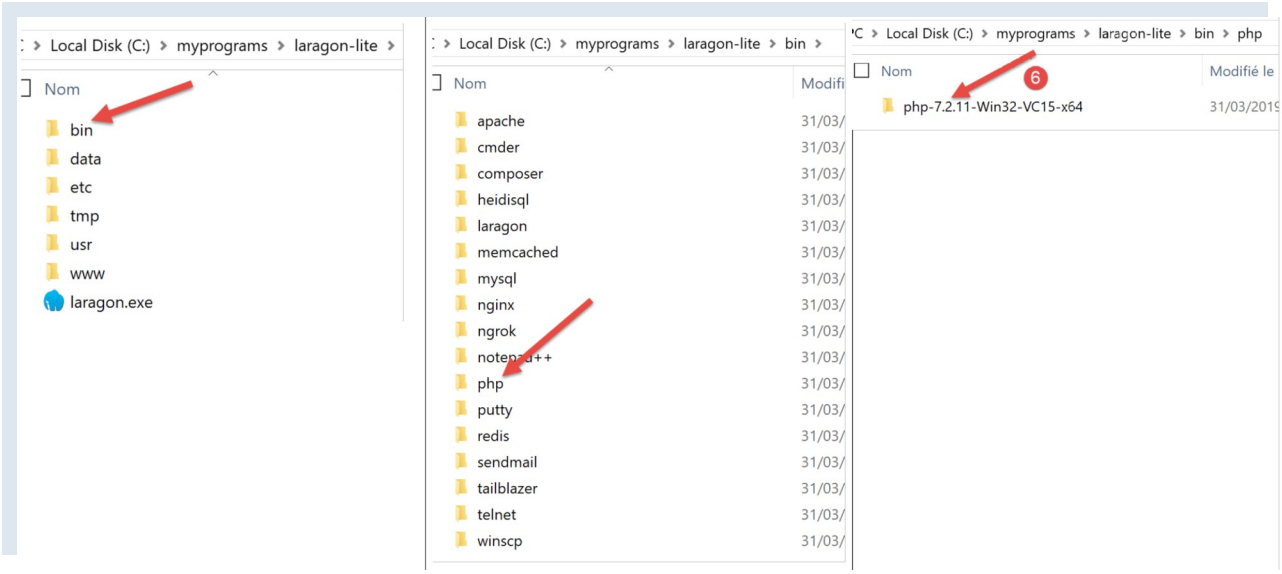
Laragon peut être téléchargé (mars 2019) à l'adresse suivante :

https://laragon.org/download/



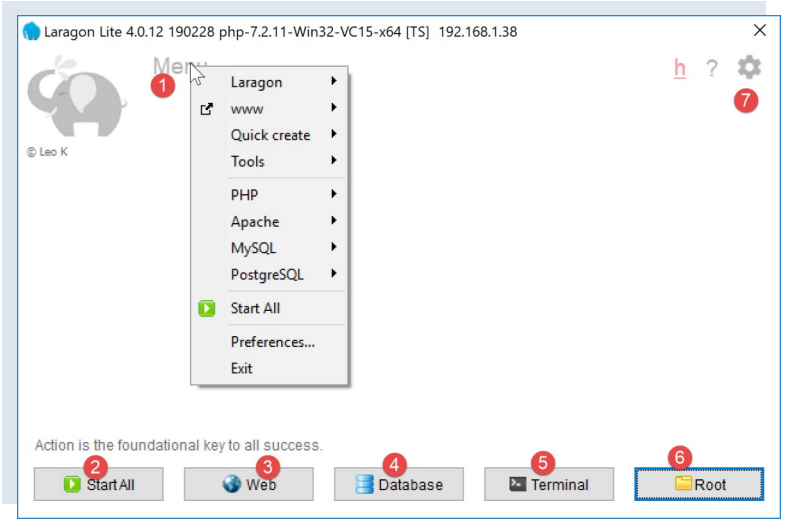


- l'installation **[1-5]** donne naissance à l'arborescence suivante :



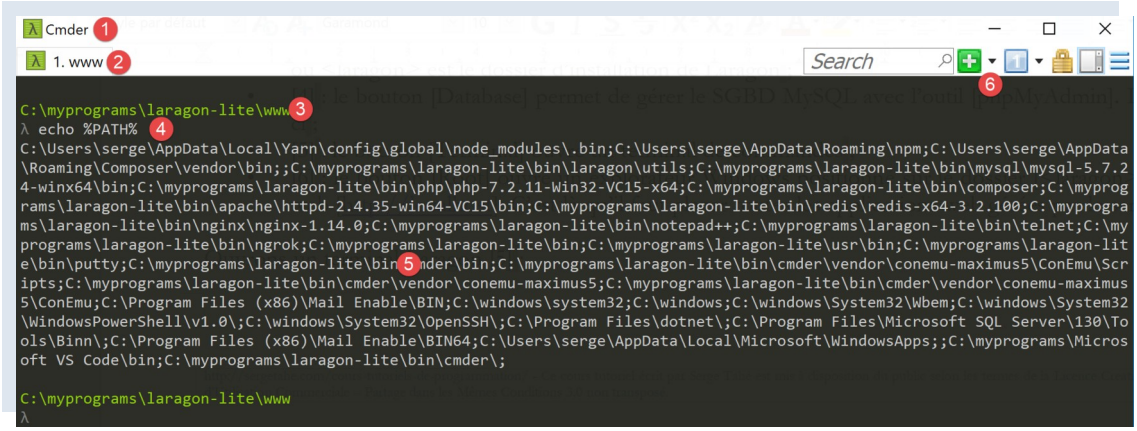
- en **[6]** le dossier d'installation de PHP ;

Le lancement de **[Laragon]** affiche la fenêtre suivante :



- **[1]** : le menu principal de Laragon ;
- **[2]** : le bouton **[Start All]** lance le serveur web Apache et le SGBD MySQL ;
- **[3]** : le bouton **[WEB]** affiche la page web **[http://localhost]** qui correspond au fichier PHP **[<laragon>/www/index.php]** où <laragon> est le dossier d'installation de Laragon ;
- **[4]** : le bouton **[Database]** permet de gérer le SGBD MySQL avec l'outil **[phpMyAdmin]**. Il faut auparavant installer celui-ci ;
- **[5]** : le bouton **[Terminal]** ouvre un terminal de commandes ;
- **[6]** : le bouton **[Root]** ouvre un explorateur Windows positionné sur le dossier **[<laragon>/www]** qui est la racine du site web **[http://localhost]**. C'est là qu'il faut placer toutes les applications web gérées par le serveur Apache de Laragon ;

Ouvrons un terminal Laragon **[5]** :

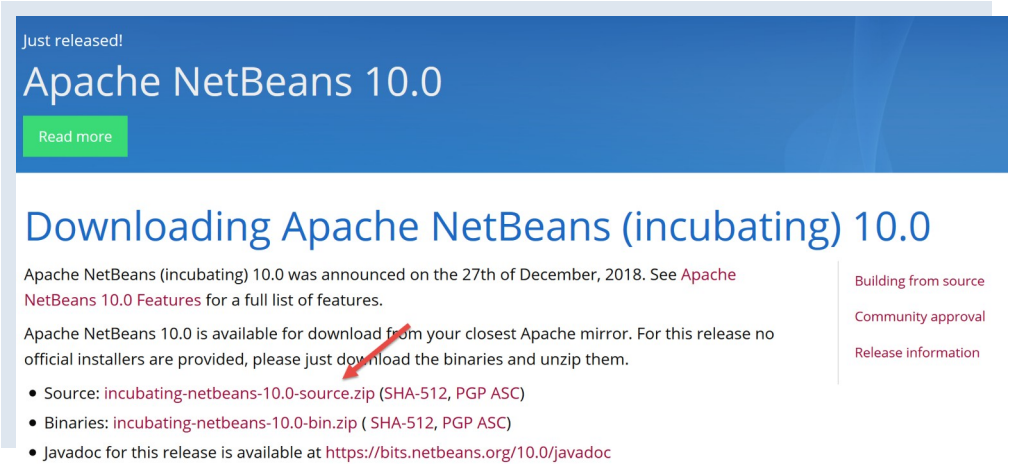


- en **[1]**, le type du terminal. Trois types de terminaux sont disponibles en **[6]** ;
- en **[2, 3]** : le dossier courant ;
- en **[4]**, on tape la commande **[echo %PATH%]** qui affiche la liste des dossiers explorés lors de la recherche d'un exécutable. Tous les principaux dossiers de Laragon sont inclus dans ce chemin des exécutables, ce qui ne serait pas le cas si on ouvrait une fenêtre de commandes **[cmd]** dans Windows. Dans ce document, lorsqu'on est amené à taper des commandes pour installer tel ou tel logiciel, c'est en général dans un terminal Laragon que ces commandes sont tapées ;

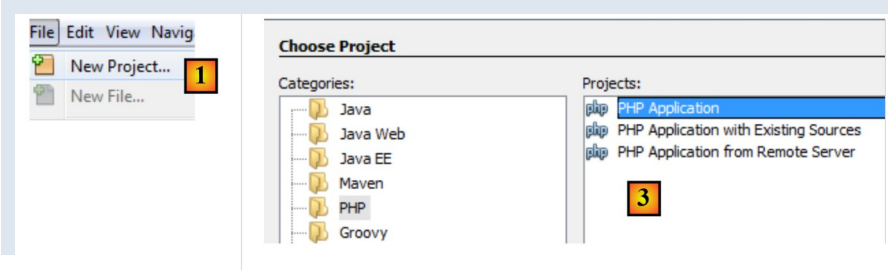
2.1.2 Installation de l'IDE Netbeans 10.0

L'IDE Netbeans 10.0 peut être téléchargé à l'adresse suivante (mars 2019) :

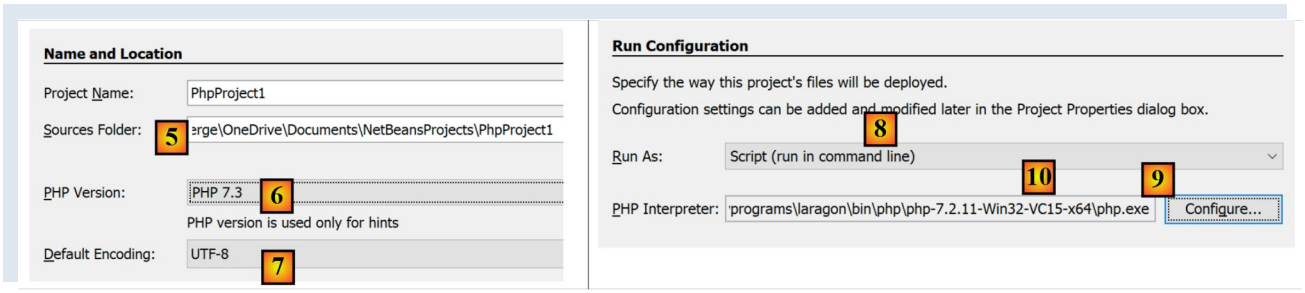
https://netbeans.apache.org/download/index.HTML



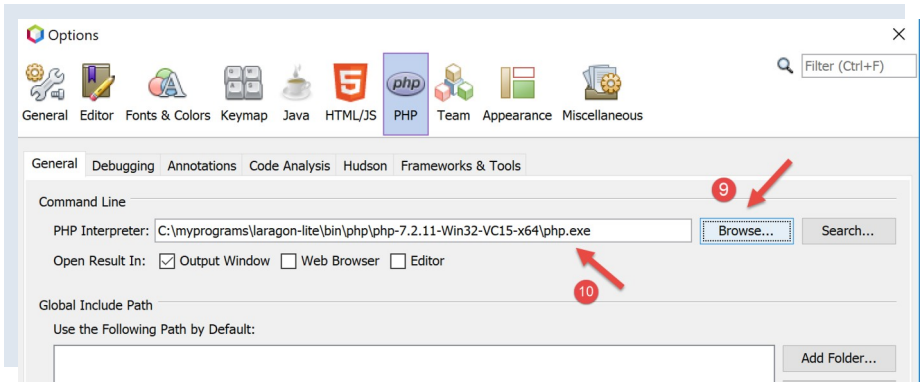
Le fichier téléchargé est un zip qu'il suffit de dézipper. Une fois Netbeans installé et lancé, on peut créer un premier projet PHP.



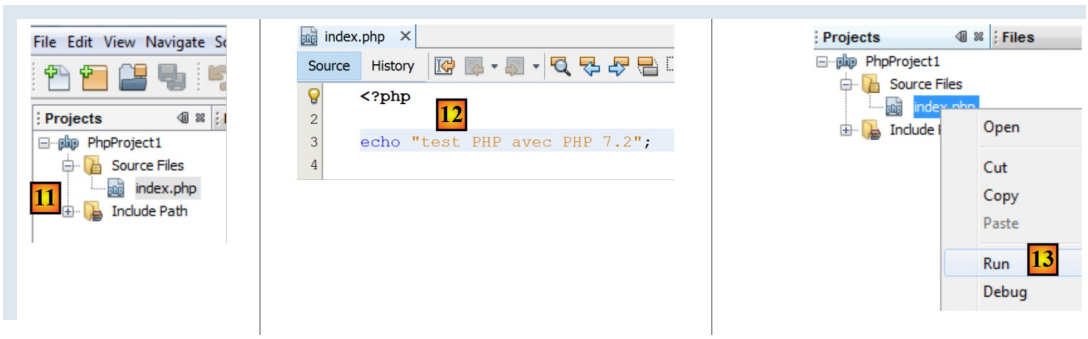
- en **[1]**, prendre l'option File / New Project ;
- en **[2]**, prendre la catégorie **[PHP]** ;
- en **[3]**, prendre le type de projet **[PHP Application]** ;



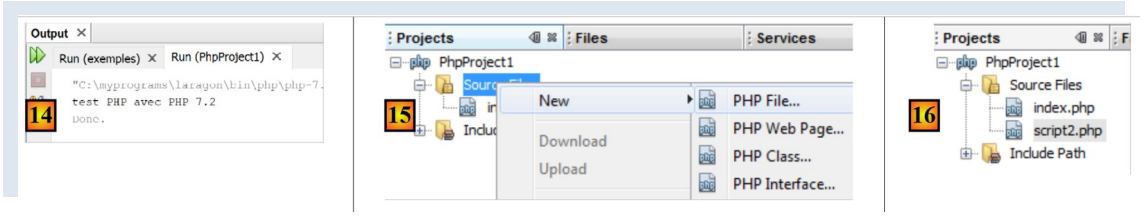
- en **[4]**, donner un nom au projet ;
- en **[5]**, choisir un dossier pour le projet ;
- en **[6]**, choisir la version de PHP téléchargée ;
- en **[7]**, choisir l'encodage UTF-8 pour les fichiers PHP ;
- en **[8]**, choisir le mode **[Script]** pour exécuter les scripts PHP en mode ligne de commande. Choisir **[Local WEB Server]** pour exécuter un script PHP dans un environnement web ;
- en **[9,10]**, indiquer le répertoire d'installation de l'interpréteur PHP du package Laragon :



- choisir **[Finish]** pour terminer l'assistant de création du projet PHP ;

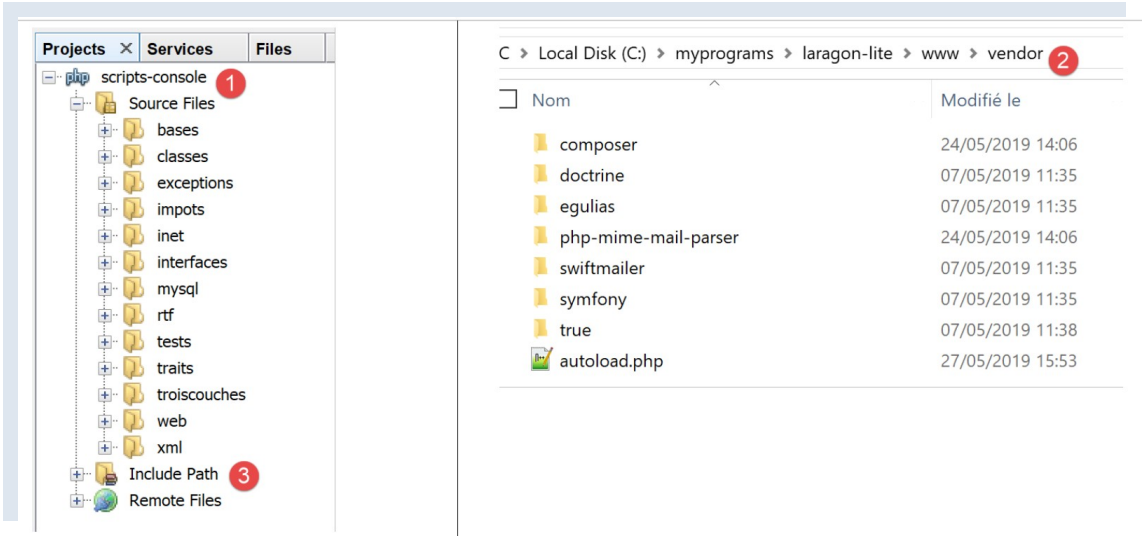


- en **[11]**, le projet est créé avec un script **[index.php]** ;
- en **[12]**, on écrit un script PHP minimal ;
- en **[13]**, on exécute **[index.php]** ;

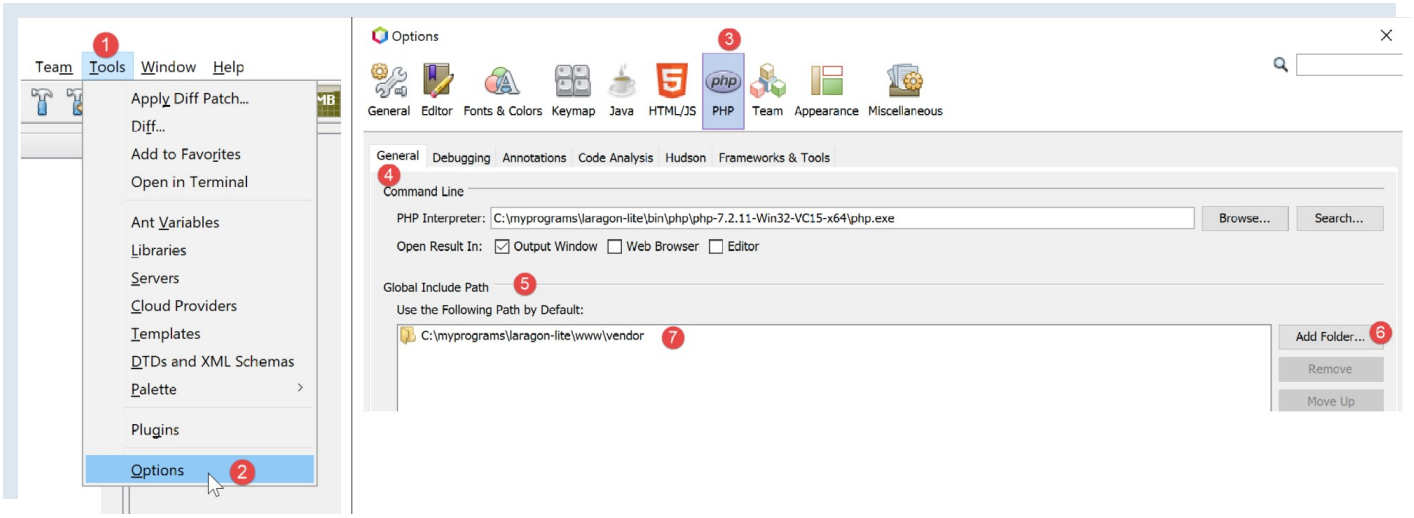


- en **[14]**, les résultats dans la fenêtre **[output]** de Netbeans ;
- en **[15]**, on crée un nouveau script ;
- en **[16]**, le nouveau script ;

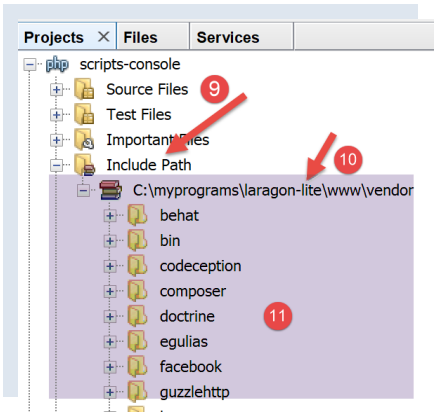
Le lecteur pourra créer tous les scripts qui vont suivre dans différents dossiers du même projet PHP. Les codes source des scripts de ce document sont disponibles sous la forme de l'arborescence Netbeans suivante :



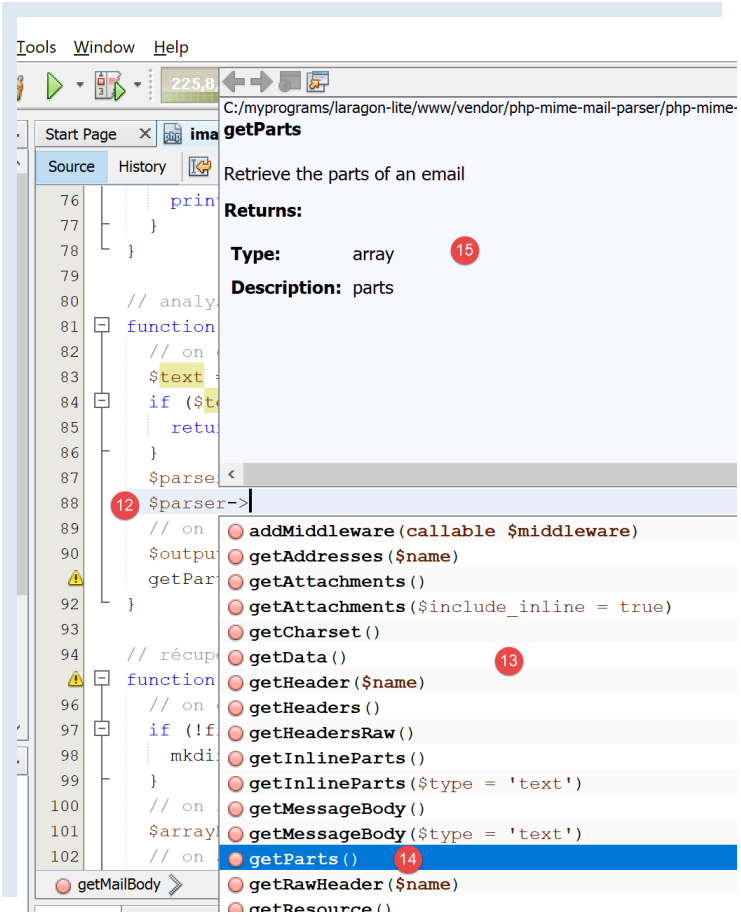
Les scripts de ce document sont placés dans l'arborescence du projet **[scripts-console] [1]**. Nous allons utiliser également des bibliothèques PHP qui seront placées dans le dossier **[<laragon-lite>/www/vendor] [2]** où <laragon-lite> est le dossier d'installation du logiciel Laragon. Pour que Netbeans reconnaisse les bibliothèques de **[2]** comme faisant partie du projet **[scripts-console]**, il nous faut inclure le dossier **[vendor] [2]** dans la branche **[Include Path] [3]** du projet. Nous allons configurer Netbeans pour que le dossier **[<laragon-lite>/www/vendor] [2]** soit inclus dans tout nouveau projet PHP et pas seulement dans le projet **[scripts-console]** :



- en **[1-2]**, on va dans les options de Netbeans ;
- en **[3-4]**, on configure les options de PHP ;
- en **[5-7]**, on configure le **[Global Include Path]** de PHP : les dossiers indiqués en **[7]** sont automatiquement inclus dans le **[Include Path]** de tout projet PHP ;



- en **[9]**, on accède aux propriétés de la branche **[Include Path]** ;
- en **[10-11]**, les nouvelles bibliothèques explorées par Netbeans. Netbeans explore le code PHP de ces bibliothèques et mémorise leurs classes, interfaces, fonctions… afin de pouvoir proposer de l'aide au développeur ;



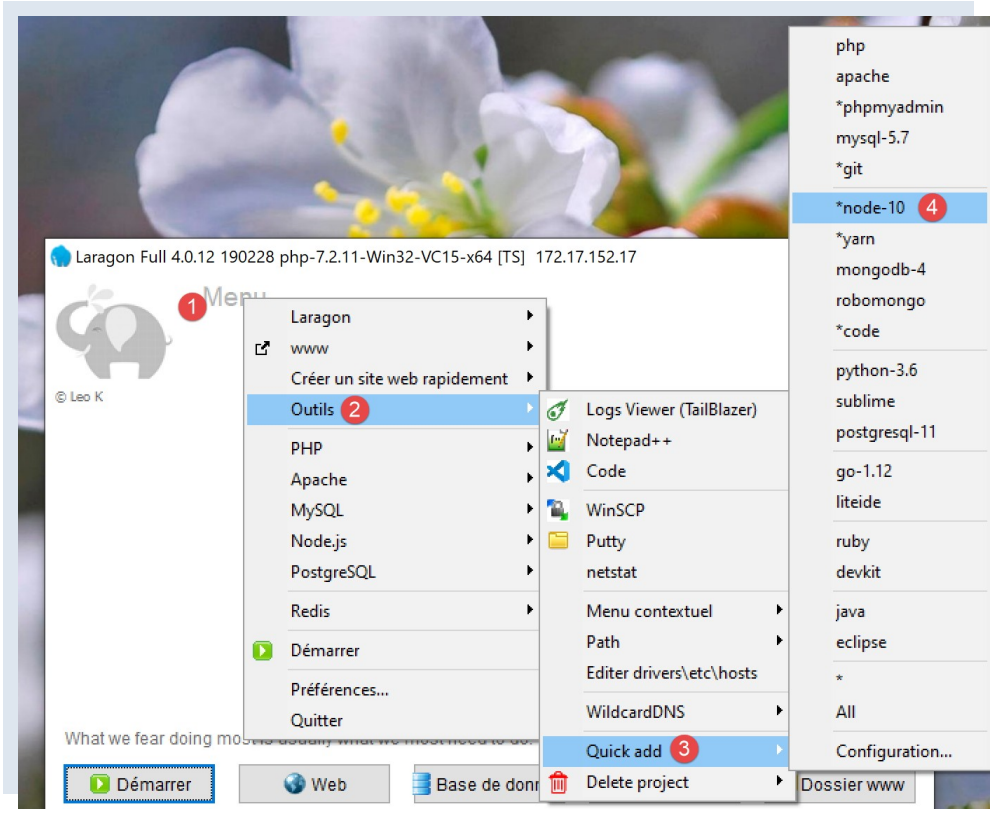
- en **[12]**, un code utilise la classe **[PhpMimeMailParser\Parser]** de la bibliothèque **[vendor/php-mime-mail-parser]** ;
- en **[13]**, Netbeans propose les méthodes de cette classe ;
- en **[14-15]**, Netbeans affiche la documentation de la méthode sélectionnée ;

La notion d'**[Include Path]** est ici propre à Netbeans. PHP a également cette notion mais ce sont a priori deux notions différentes.

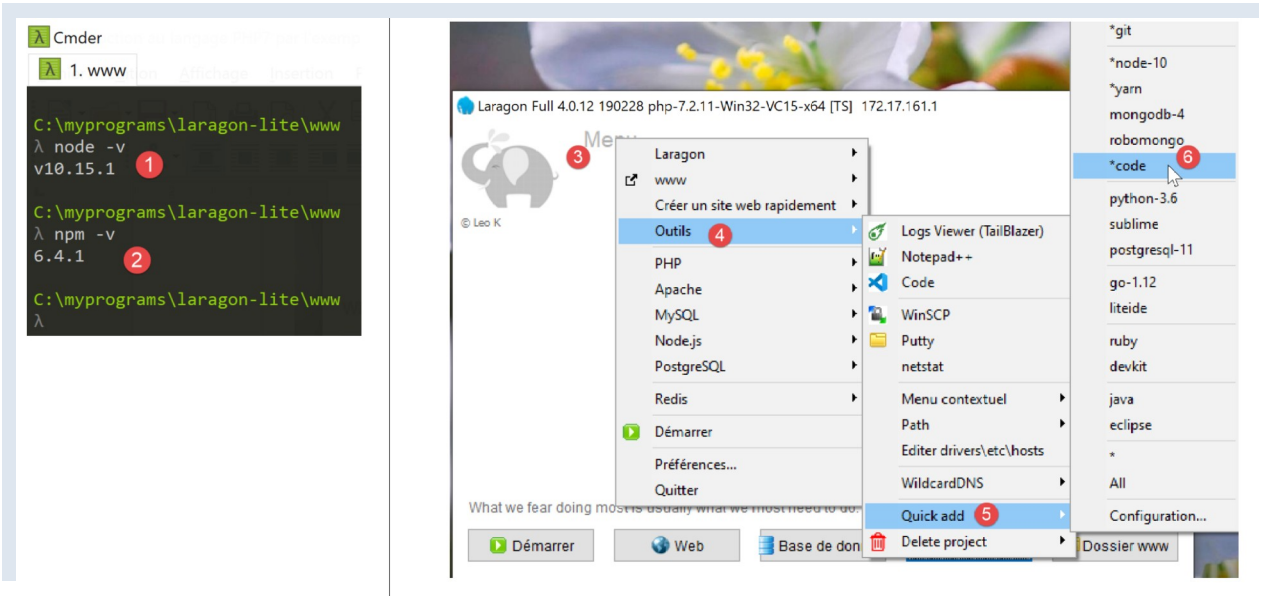
Maintenant que l'environnement de travail a été installé, nous pouvons aborder les bases de PHP.

2.2 Environnement de travail pour JavaScript

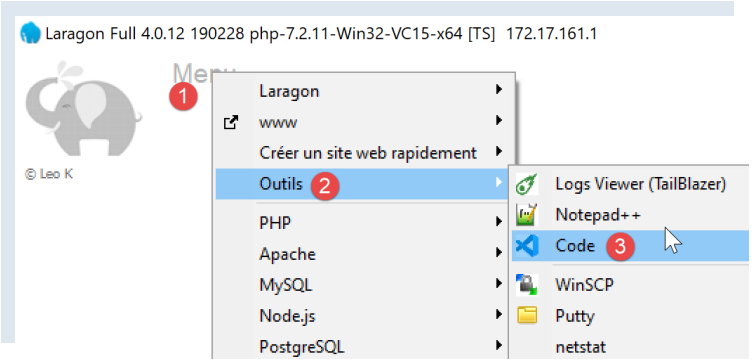
Ces outils peuvent être installés à partir de Laragon (cf. paragraphe lien) :

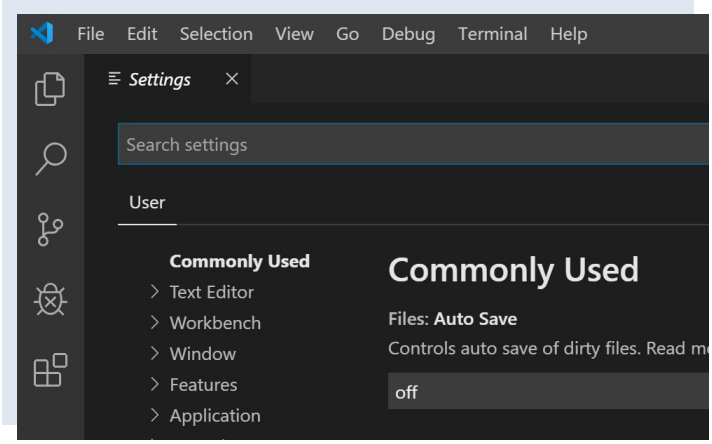


En **[4]**, on installe **[node.js]**. Une fois l'installation terminée, on ouvre un terminal Laragon (cf. paragraphe lien) et on demande la version de **[node.js]** installée (1) ainsi que celle de **[npm]** (2) :



Ensuite nous installons Visual Studio Code appelé fréquemment **[code]** ou **[VSCode]** **[3-6]**. Ceci fait, nous pouvons lancer cet outil de développement :

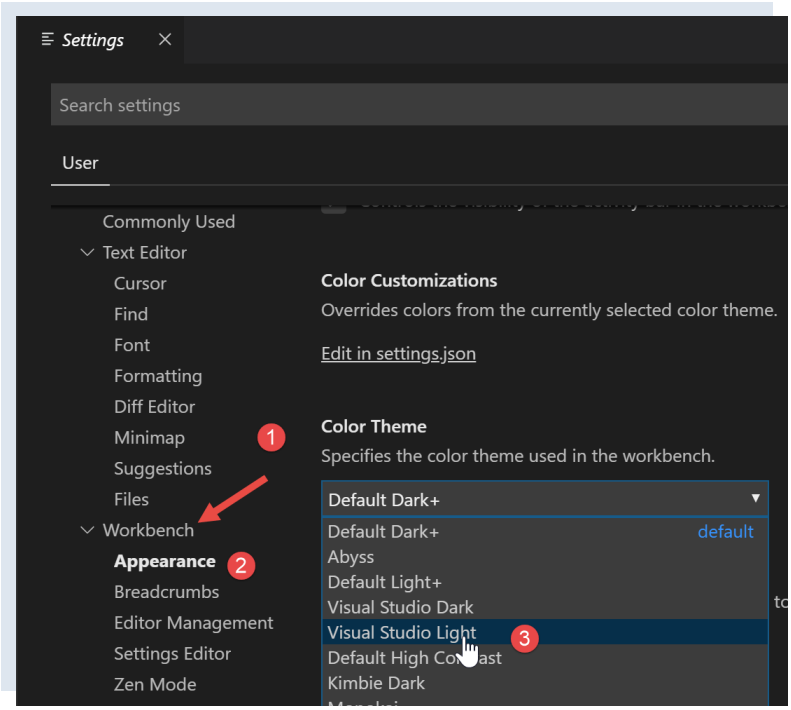




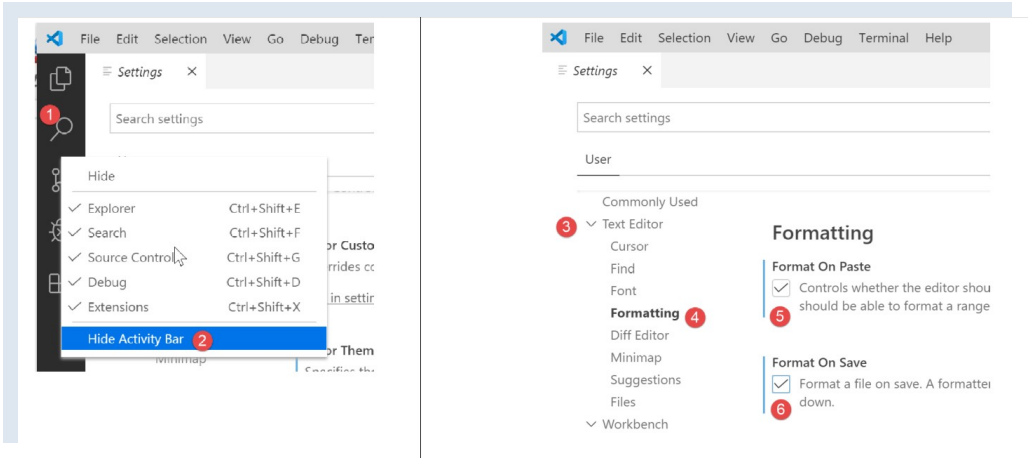
2.2.1 Configuration de Visual Studio Code

Nous montrons maintenant comment nous avons configuré **[VSCode]** afin que le lecteur comprenne les copie d'écran qui apparaîtront de temps en temps. Le lecteur est lui libre de configurer **[VSCode]** comme il l'entend. Il peut même installer son environnement de travail favori. Celui-ci importe peu pour ce que nous allons faire par la suite.

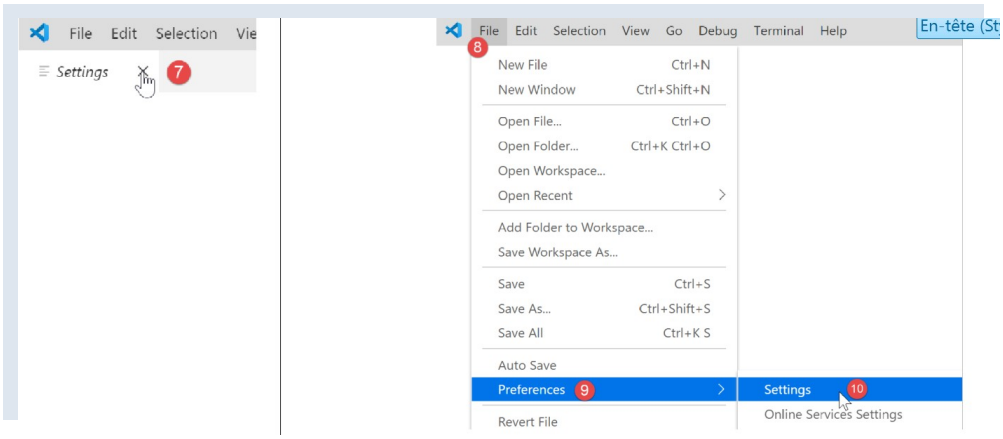
Tout d'abord, nous changeons l'apparence de la fenêtre **[VSCode]** pour avoir un fond clair plutôt que noir :



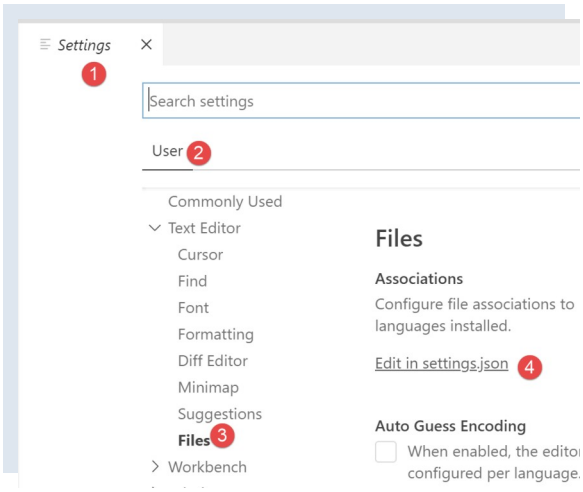
Puis nous cachons la barre de gauche **[1-2]** dont les éléments sont également disponibles dans le menu. En **[3-6]**, nous demandons un formatage du code à chaque sauvegarde du fichier et à chaque copier / coller.



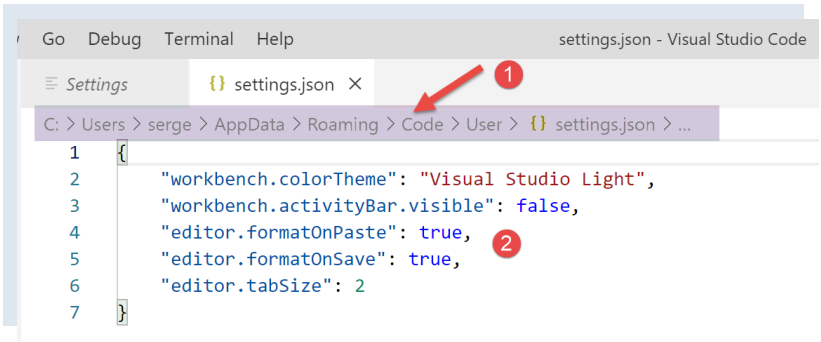
Après avoir sauvegardé la configuration **[Ctrl-S]**, on peut fermer la fenêtre **[Settings] [7]**. On peut revenir à tout moment à la configuration de **[VSCode] [8-10]** :



Ces configurations sont sauvegardées dans un fichier **[settings.json]** que l'on peut éditer directement. Ouvrons la fenêtre de configuration **[Settings]** comme il a été vu :

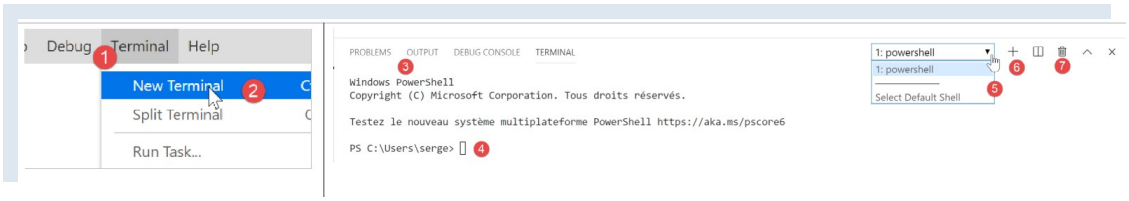


En **[4]**, on peut éditer directement le fichier **[settings.json]** :



- en **[1]**, le chemin du fichier **[settings.json]**. Une façon de revenir à la configuration par défaut est de supprimer ce fichier ;
- en **[2]**, les configurations que nous venons de faire ;

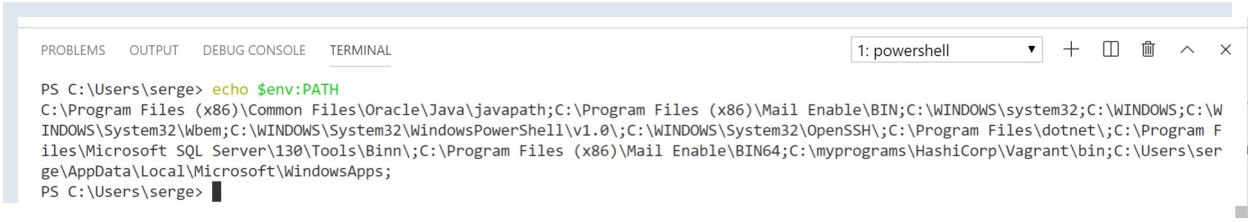
Maintenant, ouvrons un terminal à l'intérieur de **[VSCode] [1-2]** :



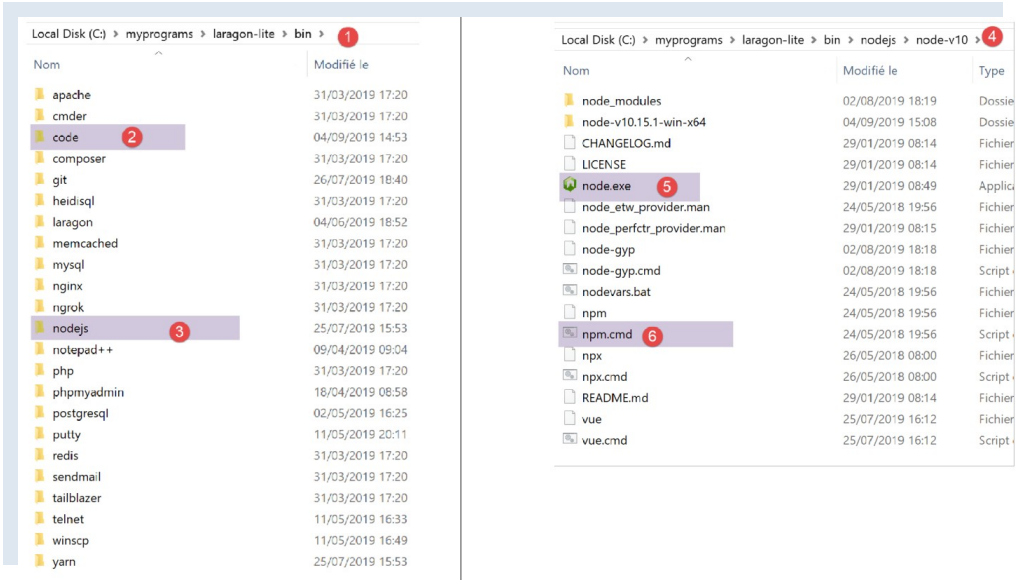
- en **[3]**, le type de terminal ouvert, ici PowerShell ;
- en **[4]**, on peut taper des commandes Windows ;
- en **[6]**, on peut ouvrir d'autres terminaux ;
- en **[5]**, la liste des terminaux ouverts ;
- en **[7]**, supprime le terminal actif ;

Nous utiliserons le terminal de **[VSCode]** pour installer des packages (bibliothèques) Javascript avec l'outil **[npm]** (Node Package Manager). Demandons, comme nous l'avons fait précédemment dans un terminal Laragon, la version de **[npm]** installée :

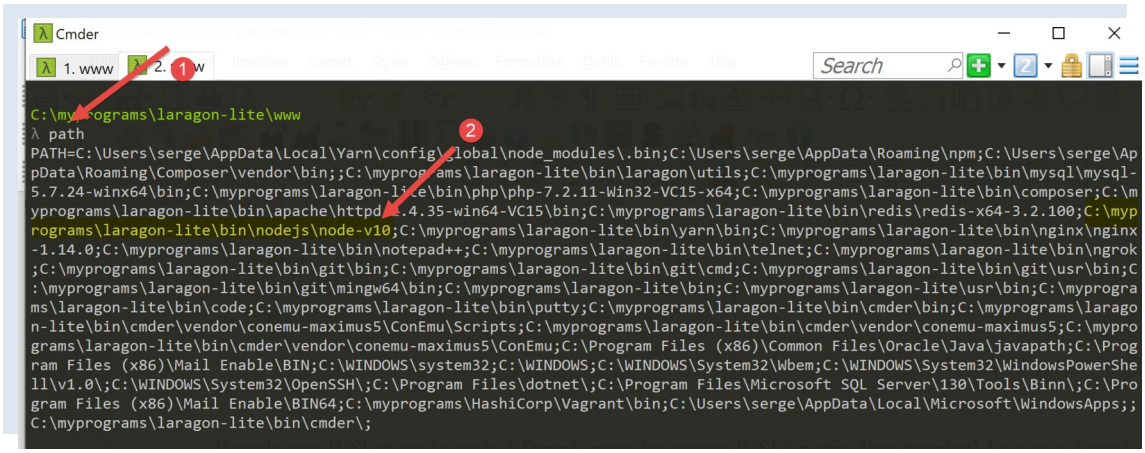
On voit que la commande **[npm]** n'a pas été reconnue. Cela signifie qu'elle n'appartient pas au PATH (liste des dossiers à explorer pour chercher un exécutable, ici **[npm]**) du terminal. On peut connaître le PATH utilisé par le terminal :



L'exécutable **[npm]** ne se trouve pas parmi ces dossiers. Comme les autres outils installés par Laragon, il se trouve dans le dossier **[<laragon>\bin]** de Laragon et plus précisément dans le dossier de **[nodejs] [4-6]**.

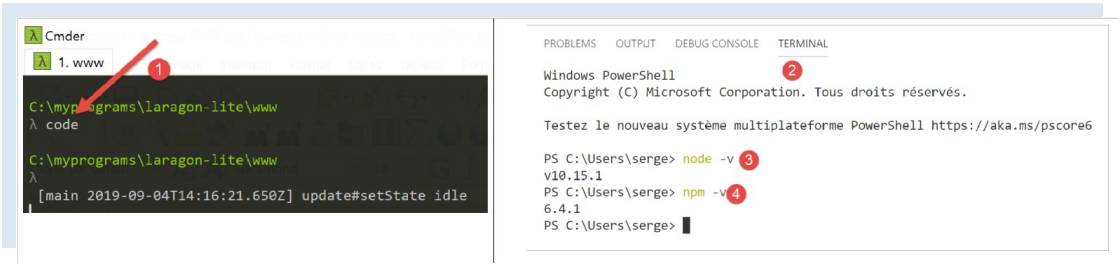


Pour lancer **[VSCode]** et avoir accès à **[npm]**, nous lancerons **[VSCode]** à partir d'un terminal Laragon. Lancé de cette façon, **[VSCode]** va hériter du PATH du terminal Laragon qui lui, contient le dossier des exécutables **[node]** et **[npm]** :



- en **[1]** : on tape la commande **[path]** ;
- en **[2]** : la liste des dossiers du PATH. On y voit le dossier **[node-v10] [2]**, ce qui nous garantit que les exécutables **[node]** et **[npm]** seront trouvés ;

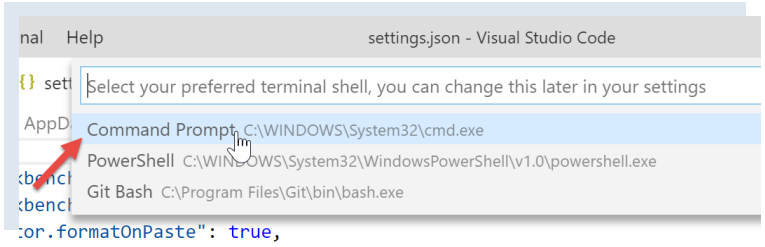
[VSCode] est lancé à partir d'un terminal Laragon avec la commande **[code]** :



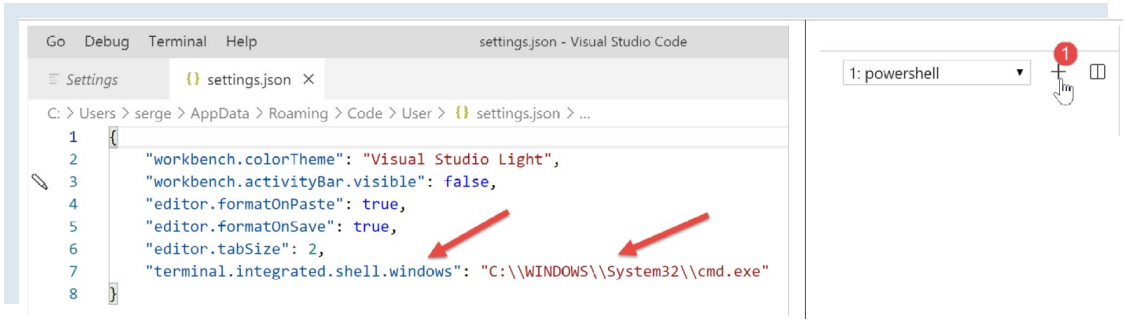
- en **[2]**, on ouvre un terminal PowerShell dans **[VSCode]** ;
- en **[3-4]**, on voit que les exécutables **[node]** et **[npm]** sont accessibles ;

Note : il ne faut pas fermer le terminal Laragon qui a lancé l'environnement de développement **[VSCode]**, sinon VSCode lui-même se ferme.

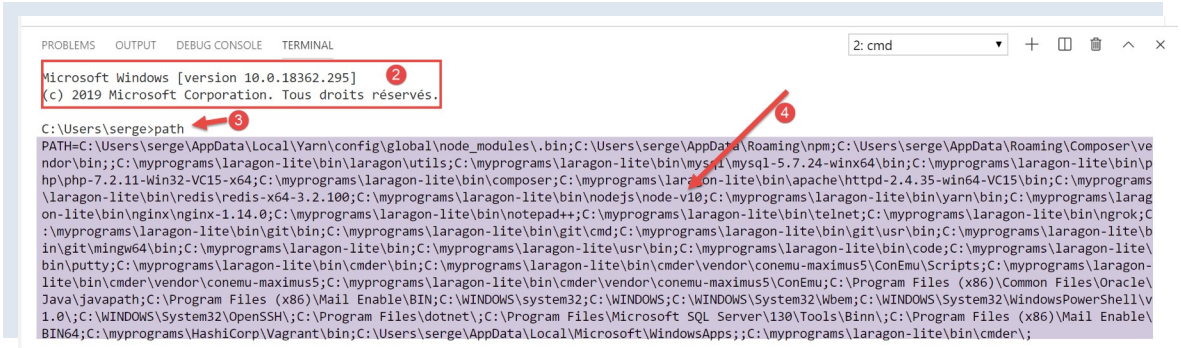
Nous allons faire une dernière configuration : nous allons changer le terminal par défaut de **[VSCode]** :



Le fichier **[settings.json]** se met aussitôt à jour :



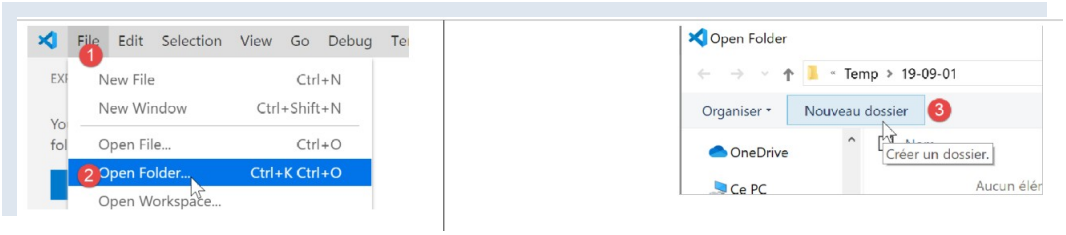
Maintenant, ouvrons un nouveau terminal **[VSCode] [1]** :

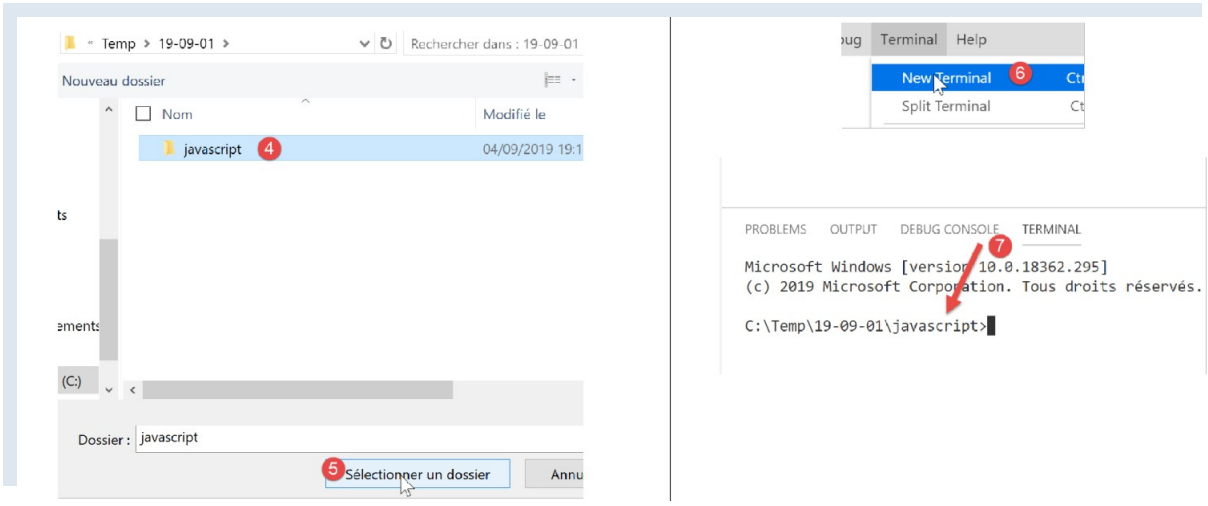


- en **[2]**, un terminal **[cmd]** (pas PowerShell) ;
- en **[3]**, la commande **[path]** donne le PATH du terminal ;
- en **[4]**, on y voit bien le dossier des exécutables **[node]** et **[npm]**

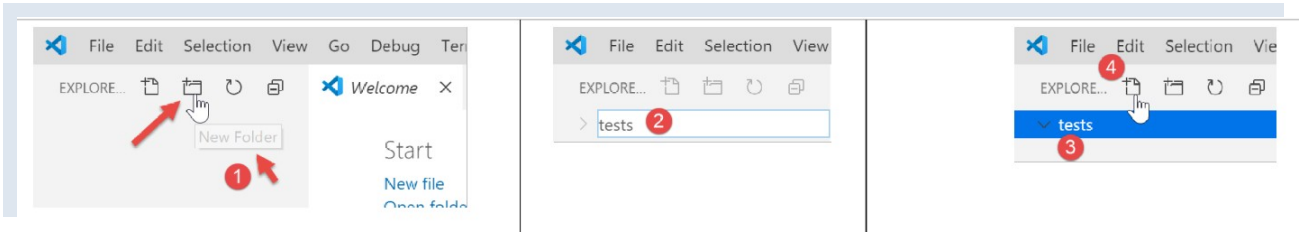
2.2.2 Ajout d'extensions à Visual Studio Code

Créons un fichier Javascript avec **[VSCode]** :

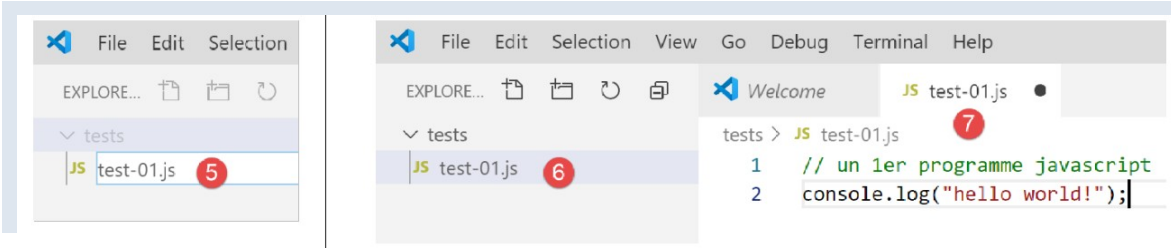




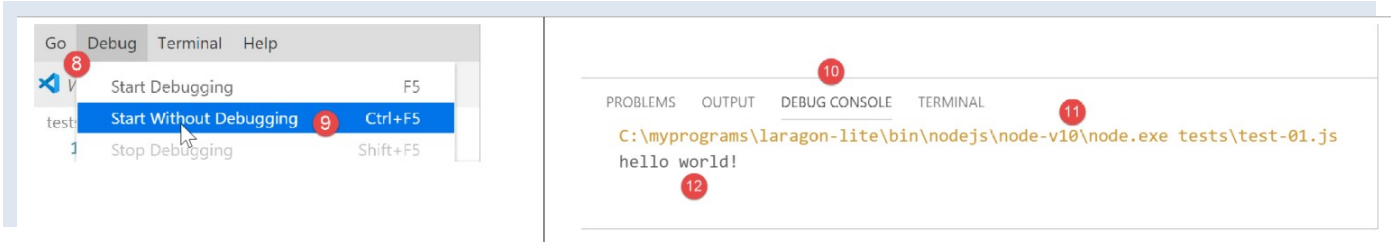
- en **[3-4]**, on crée un dossier ;
- en **[5]**, on en fait le dossier courant de **[VSCode]** ;
- en **[6]**, on ouvre un terminal ;
- en **[7]**, on voit qu'on est positionné sur le dossier choisi. Les opérations à suivre vont se faire dans celui-ci ;



- en **[1-3]** : on crée un nouveau dossier ;
- en **[4]** : on ajoute un fichier dans ce dossier ;

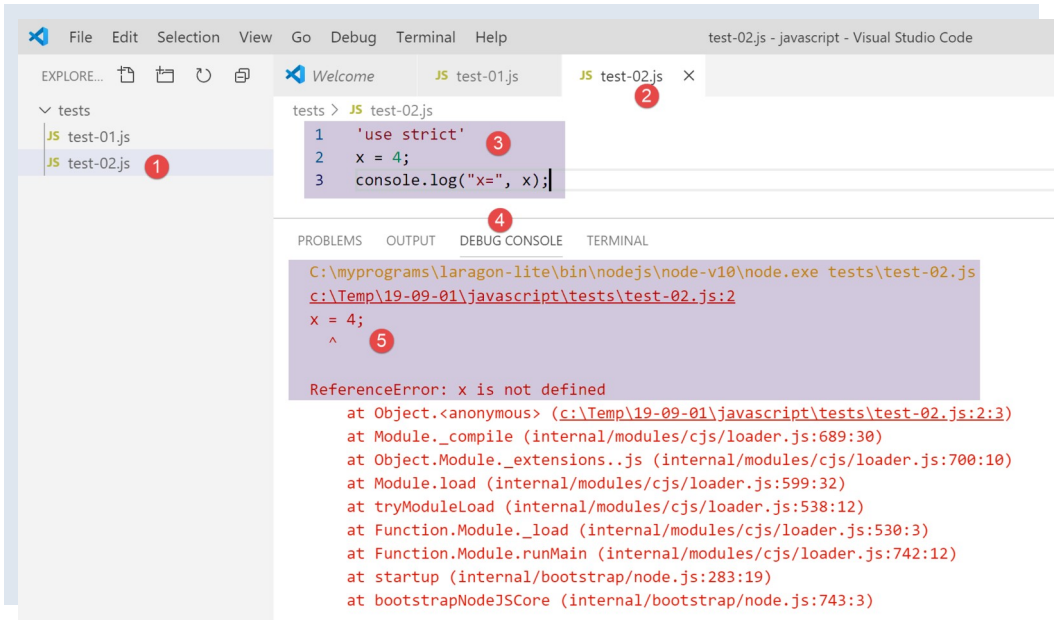


- en **[5-7]** : on crée notre 1er programme Javascript ;



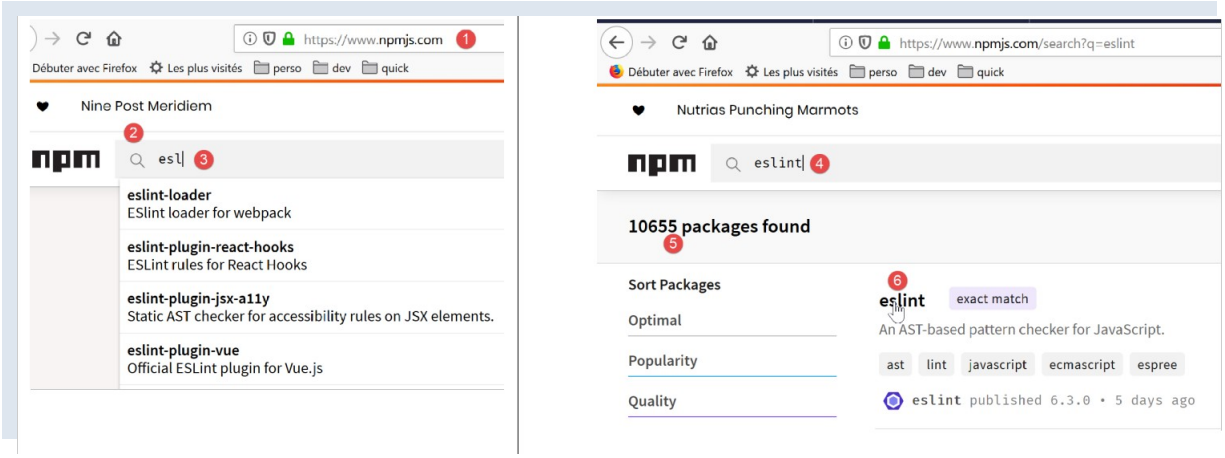
- en **[8-9]** : on exécute le programme Javascript ;
- le résultat apparaît dans la console d'exécution **[10]**. On voit en **[11]** la commande qui a été exécutée : c'est l'application **[node]** qui a exécuté le script **[test-01.js]**. C'est parce que cet exécutable est dans le PATH de **[VSCode]** que cela a pu être possible, sinon on aurait eu une erreur indiquant que la commande **[node]** n'était pas connue ;

Procédons de la même façon pour exécuter un second script **[test-02.js]** :

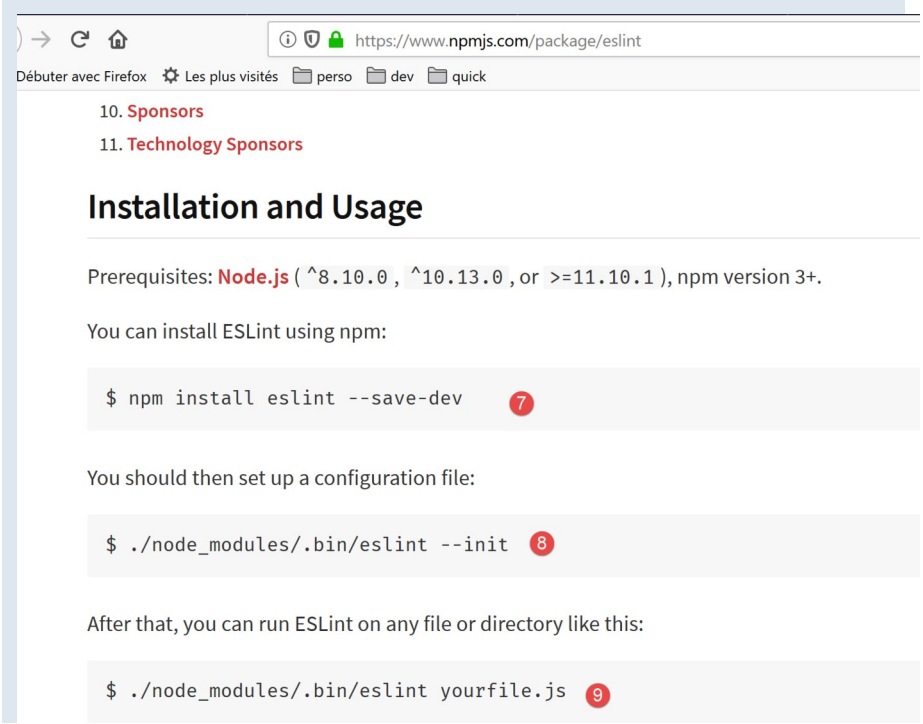


- en **[1-3]**, on définit le nouveau script. L'instruction **[use strict]** de la ligne 1 demande une vérification stricte de la syntaxe. Dans ce contexte, toute variable doit être déclarée avec l'un des mots clés **[let, const, var]**. Ce n'est pas le cas de la variable **[x]** de la ligne 2 ;
- lorsqu'on exécute ce code par **[Ctrl-F5]**, on obtient l'erreur **[5]**. Il est possible d'être averti de ce type d'erreur avant l'exécution. C'est préférable. Nous allons faire deux choses :
  - installer avec **[npm]** une bibliothèque appelée **[eslint]** qui vérifie que la syntaxe du script est conforme à la norme ECMAScript 6 ;
  - installer une extension à Visual Studio Code, appelée elle-aussi ESLINT qui facilite l'utilisation de la bibliothèque **[eslint]** au sein de **[VSCode]** ;

Installons tout d'abord la bibliothèque Javascript **[eslint]** à l'aide de l'outil **[npm]**. Pour installer une bibliothèque (on dit un package) **[npm]**, il faut connaître son nom exact. Si ce n'est pas le cas, on peut aller sur le site de **[npm]** à l'URL (2019) **[https://www.npmjs.com/]** :

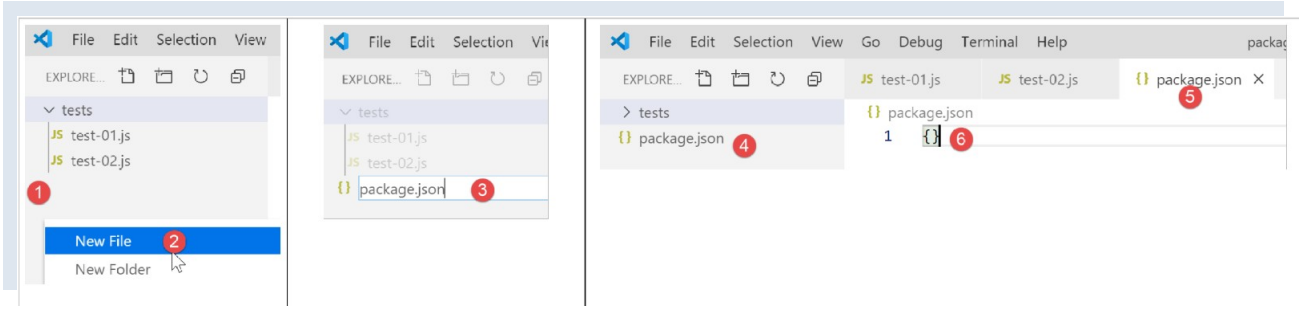


- en **[3]**, les packages commençant par **[esl]** ;
- en **[4-6]**, on trouve le package **[eslint]** ;



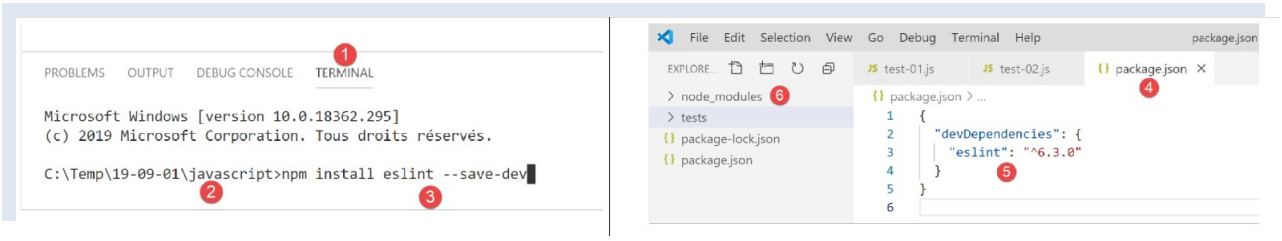
- en **[7]**, la commande **[npm]** pour installer le package **[eslint]** ;
- en **[8]**, la configuration du package ;
- en **[9]**, son utilisation pour vérifier la syntaxe d'un script Javascript ;

Nous installons le package **[eslint]** dans une fenêtre **[Terminal]** de **[VSCode]**. Tout d'abord, il nous faut créer un fichier **[package.json]** à la racine du dossier de travail de **[VSCode]**. Ce fichier contiendra la liste des packages jS utilisés par le projet **[VSCode]** :

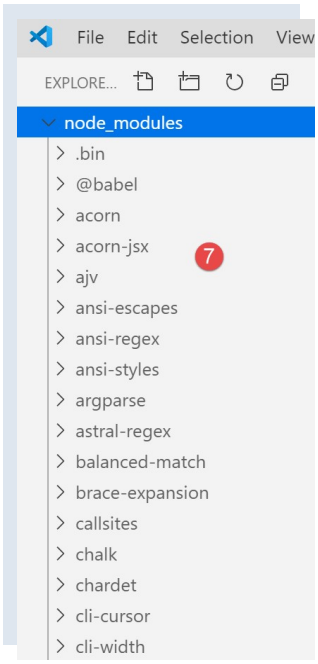


- en **[1]**, cliquer droit dans l'explorateur de projet (pas sur le dossier tests) ;
- en **[3-4]**, on crée le fichier **[package.json]** à la racine du projet **[javascript]**, au même niveau que le dossier **[tests]** (mais pas dans **[tests]**) ;
- en **[4-6]**, on met dans le fichier **[package.json]** un objet jSON vide ;

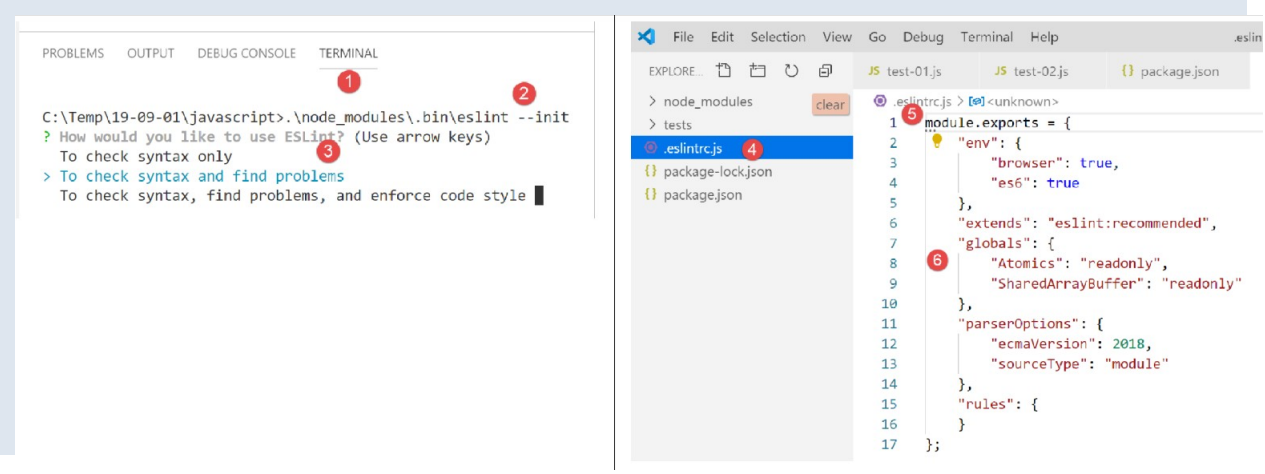
Puis on ouvre un terminal **[VSCode]** pour installer **[eslint]** :



- en **[2]**, on est à la racine du projet **[javascript]** ;
- en **[3]**, la commande qui installe le package **[eslint]** ;
- après exécution,
  - en **[4-5]**, le fichier **[package.json]** a été modifié. Ligne 3, on trouve la version de **[eslint]** installée. Ligne 2, **[devDependencies]** correspond à l'argument **[--save-dev]** de l'installation. Cet argument signifie que la dépendance installée doit être inscrite dans le fichier **[package.json]** comme élément de la propriété **[devDependencies]**. Cette propriété liste les dépendances du projet dont on a besoin en mode développement mais pas en mode production. En effet, on a besoin de la dépendance **[eslint]** uniquement en développement pour vérifier que le code écrit respecte la norme ECMAScript ;
  - en **[6]**, un dossier **[node_modules]** est apparu dans le projet. C'est le dossier où sont installées les dépendances du projet ;



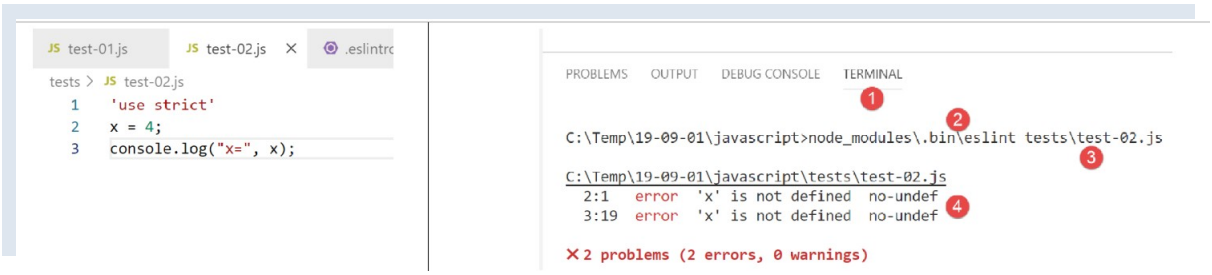
- en **[7]**, une partie des packages installés. Ceux-ci sont très nombreux. En effet, non seulement le package **[eslint]** a été installé mais également tous les packages sur lesquels celui-ci s'appuie ;



- **[1-2]**, dans un terminal **[VSCode]** on émet la commande de configuration du package **[eslint]**. Celle-ci va poser diverses questions **[3]** pour savoir comment on souhaite utiliser **[eslint]**. Dans le doute, laissez les options proposées par défaut. Pour sélectionner une option, utilisez les flèches haute et basse du clavier pour choisir l'option puis validez celle-ci ;
- en **[4]**, un fichier **[.eslintrc.js]** a été créé à la racine du projet ;
- en **[6]**, le contenu du fichier. Vous pouvez en copier le contenu dans votre propre fichier ;

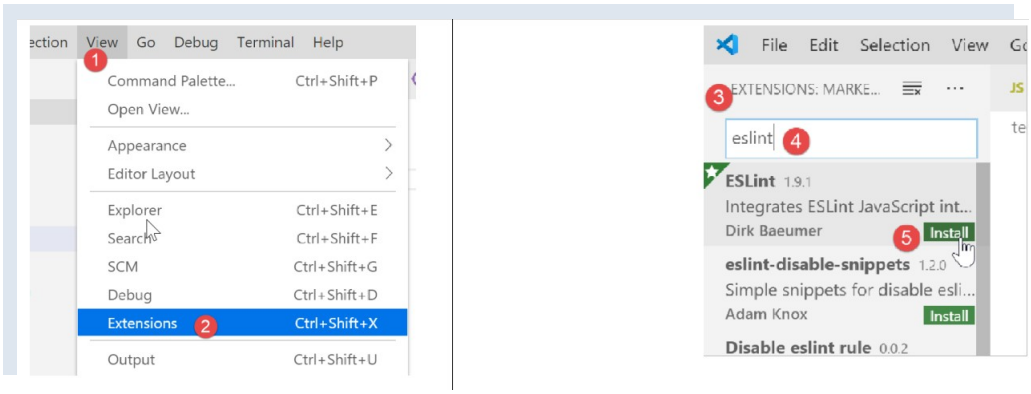
```
1.  module.exports = {
2.    "env": {
3.      "browser": true,
4.      "es6": true
5.    },
6.    "extends": "eslint:recommended",
7.    "globals": {
8.      "Atomics": "readonly",
9.      "SharedArrayBuffer": "readonly"
10.   },
11.   "parserOptions": {
12.     "ecmaVersion": 2018,
13.     "sourceType": "module"
14.   },
15.   "rules": {
16.   }
17. };
```

Tout ceci n'est pas suffisant pour signaler les erreurs du fichier **[test-02.js]** :

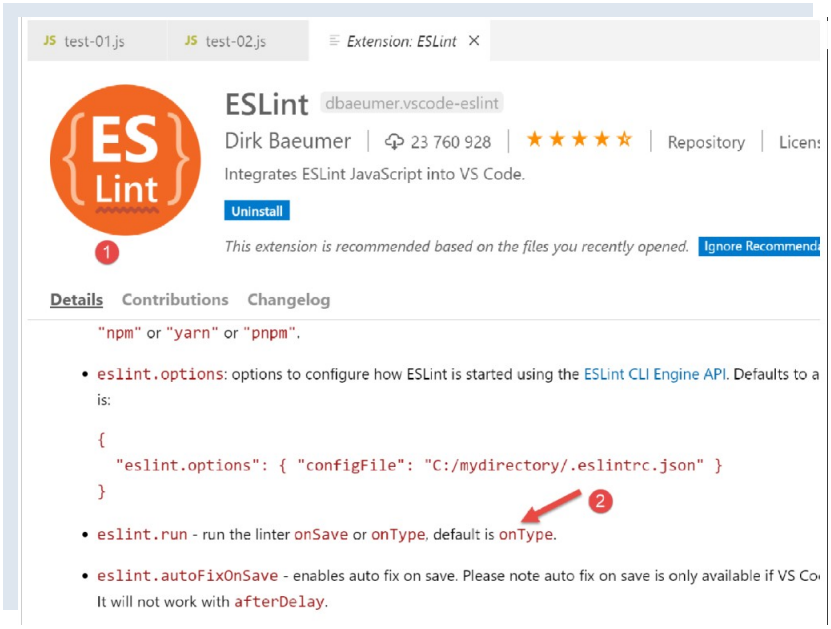


- il faut taper la commande **[2-3]** pour que le fichier **[tests/test-02.js]** soit analysé ;
- en **[4]**, l'erreur sur la variable non déclarée est détectée ;

Nous allons ajouter à **[VSCode]** une extension qui va permettre de voir les erreurs Javascript en temps réel. Cette extension s'appuie sur le package **[eslint]** que nous avons installé :



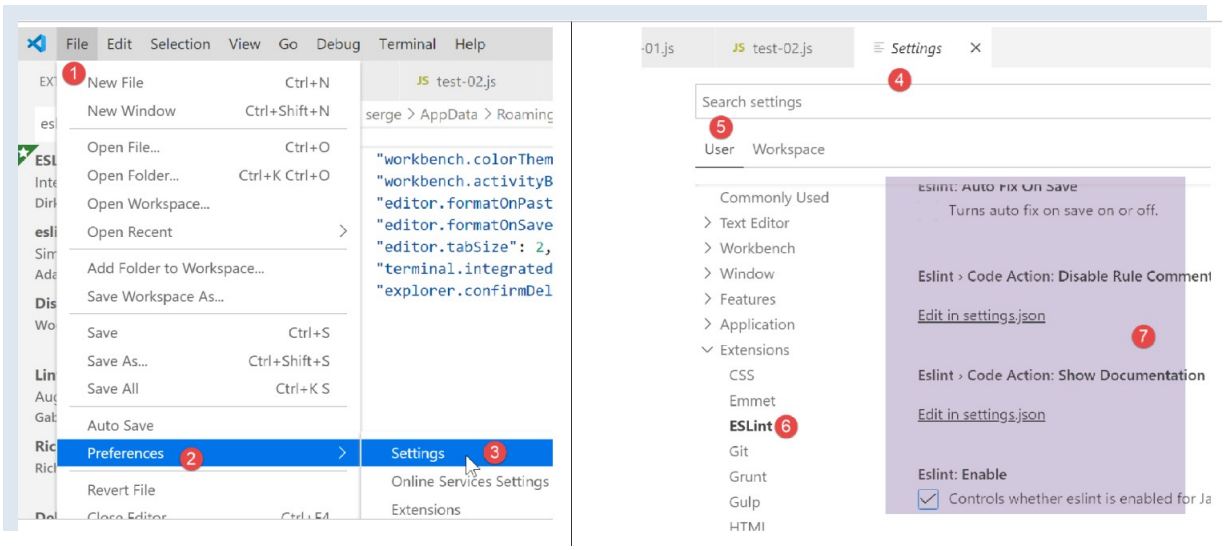
- en **[3-5]**, nous installons l'extension appelée **[ESLint]** ;



- en **[1]**, une page d'informations sur l'extension nouvellement installée ;

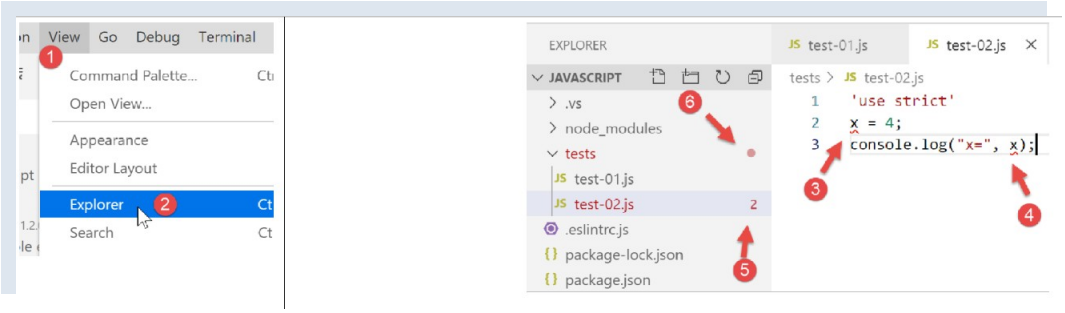
en **[2]**, on voit que le mode de vérification de **[ESLint]** est **[type]**, ce qui signifie que la syntaxe des scripts jS sera vérifiée en même temps que la frappe du texte ;

ESLint peut être configuré via le fichier de configuration général de **[VSCode]** :



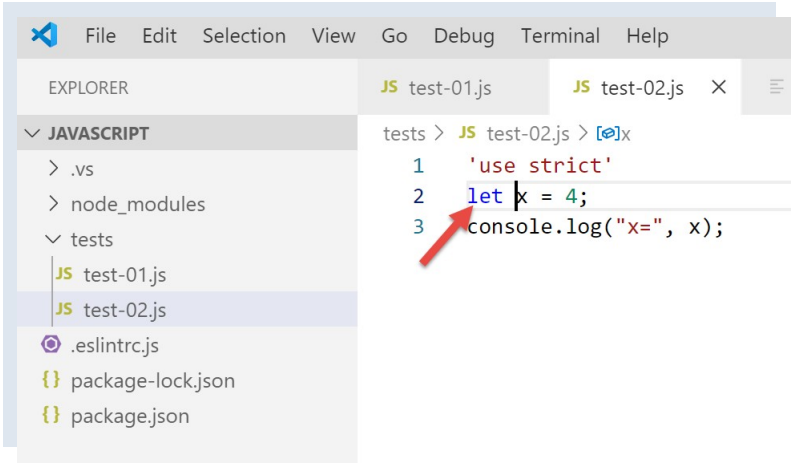
- en **[6-7]**, la configuration de **[ESLint]**. C'est ici que vous pourrez la modifier ;

Revenons maintenant au fichier **[test-02.js]** :

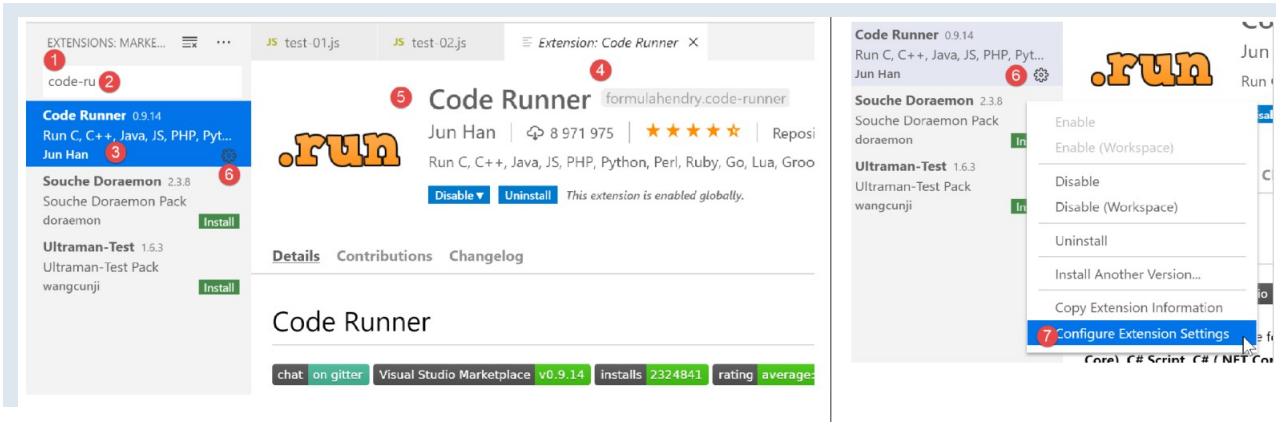


- en **[3-4]**, les erreurs sur la variable **[x]** sont désormais signalées ;
- en **[5]** : le nombre d'erreurs ESLint dans le fichier ;
- en **[6]**, indique que dans le dossier **[tests]**, il y a des fichiers erronés ;

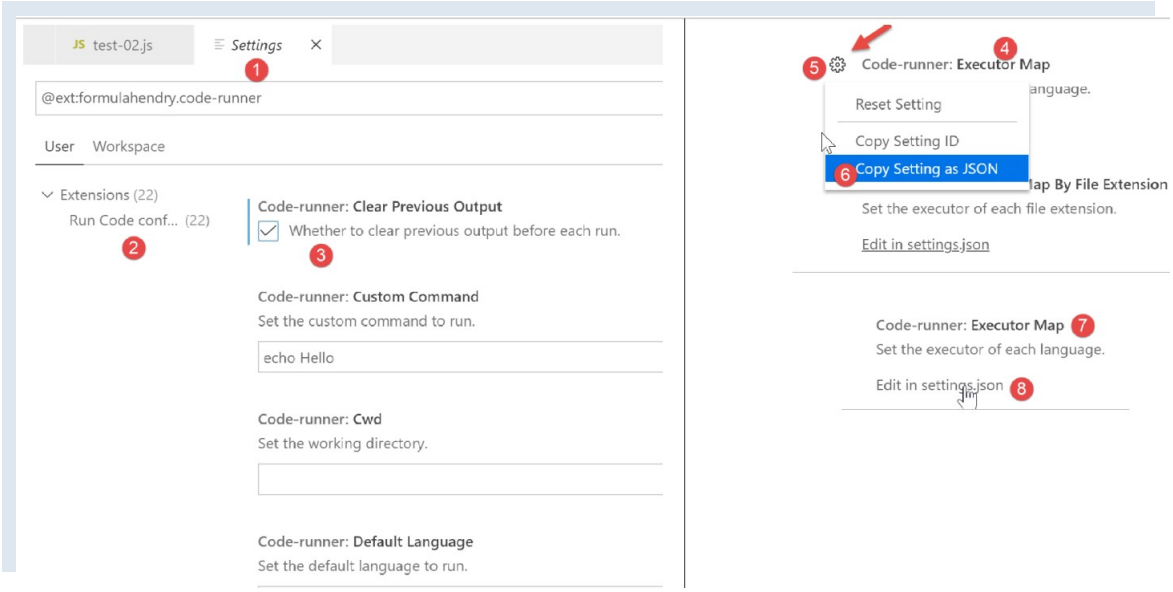
Si on corrige l'erreur, les avertissements d'ESLint disparaissent :



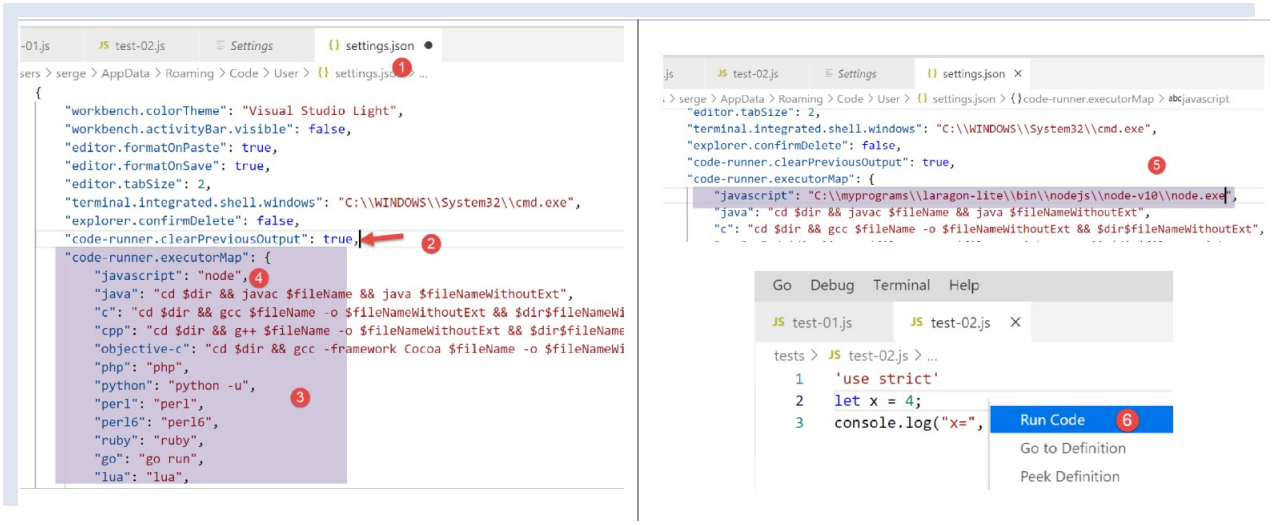
Installons maintenant une extension appelée **[Code Runner]** :



- une fois installée l'extension **[Code-Runner] [1-5]**, on peut la configurer avec **[6-7]** (ci-dessus) ;

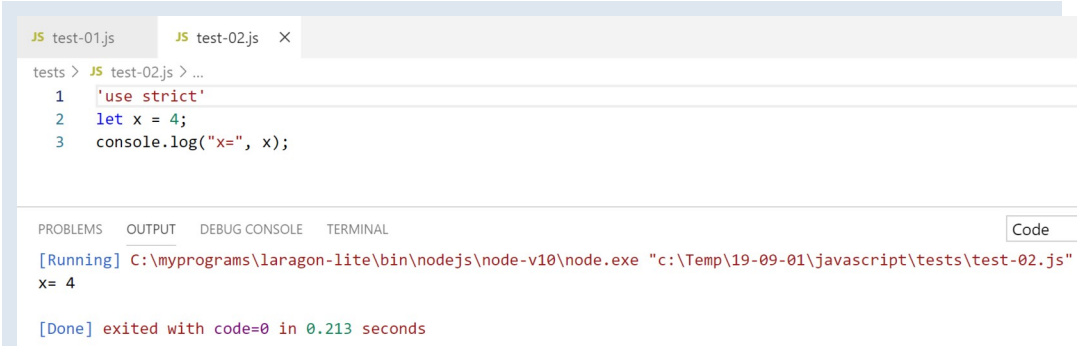


- en **[1-2]**, les éléments de configuration de **[Code-Runner]** ;
- en **[3]**, on demande à ce que le terminal de sortie soit nettoyé avant chaque exécution ;
- en **[4]**, on localise l'élément **[Executor Map]** qui liste les outils d'exécution de différents langages ;
- en **[5-6]**, on copie la configuration dans le presse-papiers ;
- en **[7-8]**, on modifie le fichier **[settings.json]** ;



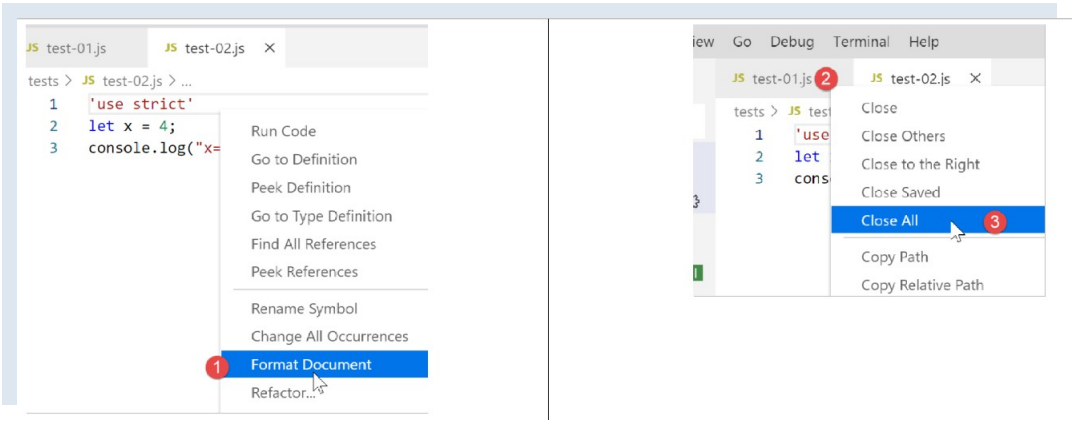
- en **[2]**, on ajoute la virgule derrière le dernier élément du fichier **[settings.json] [1]** ;
- en **[3]**, on colle ce qu'on a copié en **[5-6]** précédemment : c'est la liste des commandes permettant d'exécuter les différents langages supportés par **[VSCode]** ;
- en **[4]**, la commande permettant d'exécuter les fichiers Javascript. Celle-ci ne fonctionne que si **[node]** est dans le PATH de **[VSCode]**. Si ce n'est pas le cas, on peut mettre le chemin complet de l'exécutable **[5]** ;

Ceci fait sauvegardons la configuration (Ctrl-S). Avec l'extension **[Code Runner]**, les fichiers Javascript peuvent être exécutés avec un clic droit sur le code **[6]** (ci-dessus) :

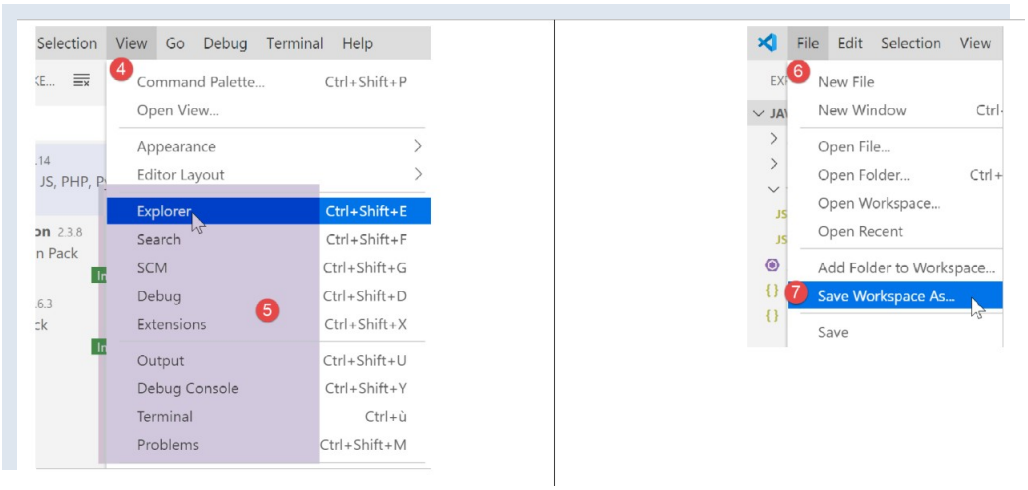


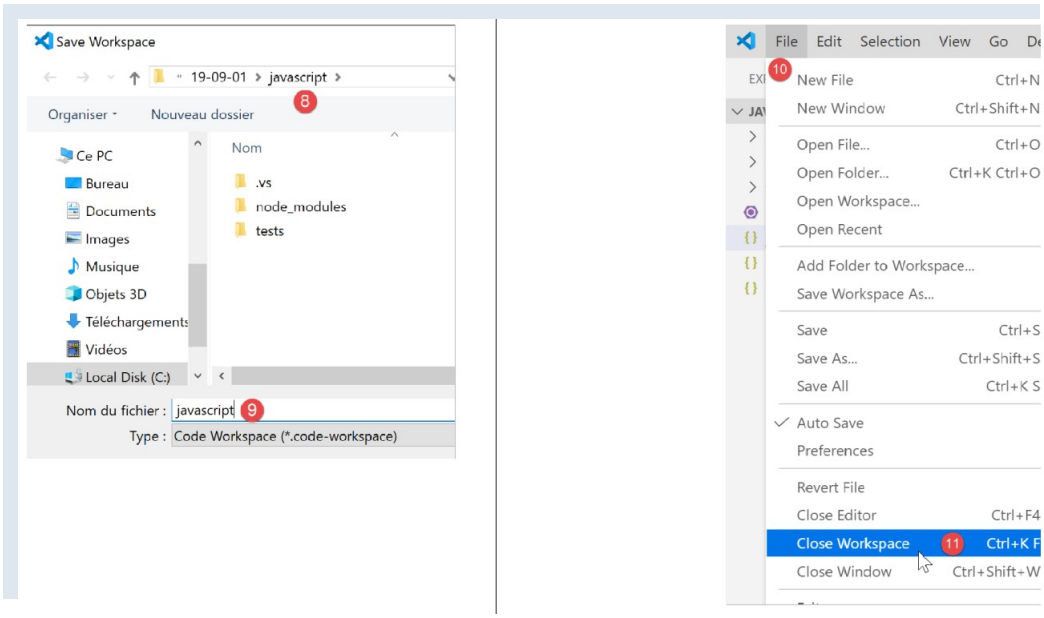
2.2.3 Quelques commandes [vscode] utiles

- pour formater votre code, cliquez droit dessus **[1]** ;
- pour fermer les fenêtres ouvertes, cliquez droit sur leurs titres **[2-3]** ;

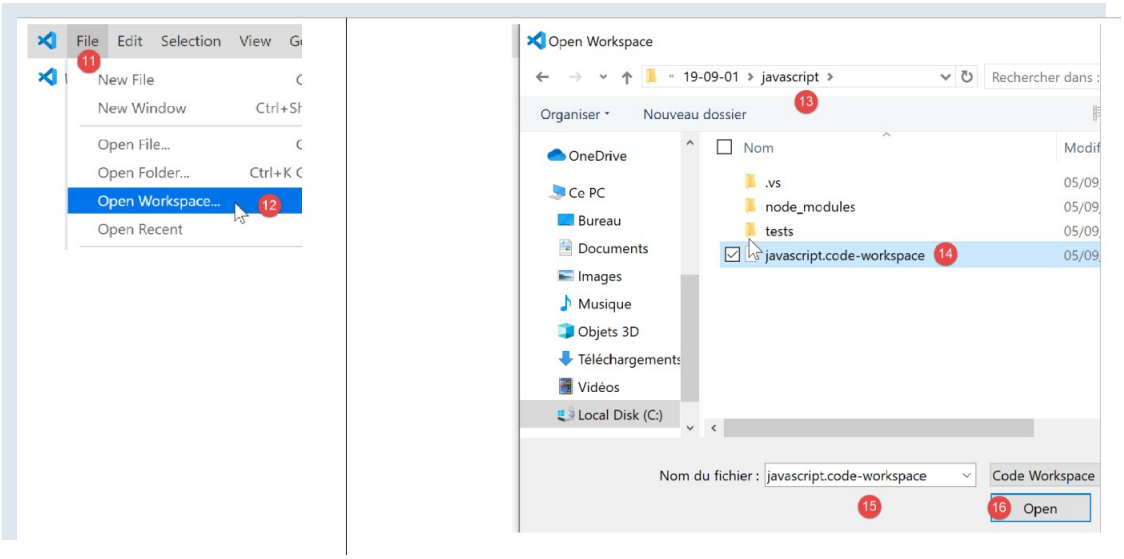


- pour afficher une fenêtre particulière **[4-5]** ;
- pour sauvegarder votre projet (Workspace) **[6-9]** ;
- pour sauvegarder un projet **[10-11]** ;

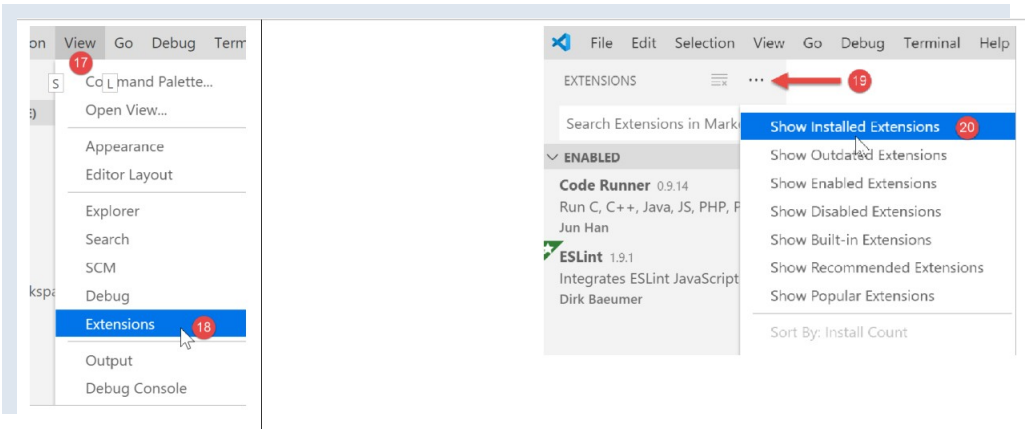




- pour ouvrir un projet **[11-16]** :



- voir les extensions installées **[19-20]** :

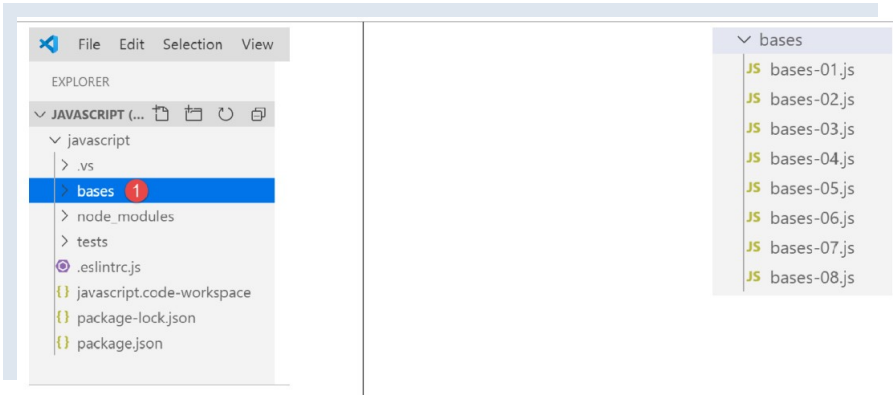


Nous avons désormais de bons outils pour développer en Javascript. Nous allons maintenant présenter ce langage à l'aide de courts extraits de code. Comme cette présentation se fait à la suite d'un cours PHP, nous ferons parfois des comparaisons entre ces deux langages pour signaler des différences entre eux.

3 Les bases de Javascript

Note : dans la suite, le terme **[Javascript]** désignera toujours la norme ECMAScript 6.

A l'intérieur du projet Javascript précédent, créez un dossier **[bases]**. Nous y mettrons les exemples de cette section :



3.1 script [bases-01]

Pour introduire les bases de PHP7, nous avions utilisé le code suivant(cf. paragraphe lien) :

```
1.  <?php
2.
3.  // ceci est un commentaire
4.  // variable utilisée sans avoir été déclarée
5.  $nom = "dupont";
6.  // un affichage écran
7.  print "nom=$nom\n";
8.  // un tableau avec des éléments de type différent
9.  $tableau = array("un", "deux", 3, 4);
10. // son nombre d'éléments
11. $n = count($tableau);
12. // une boucle
13. for ($i = 0; $i < $n; $i++) {
14.   print "tableau[$i]=$tableau[$i]\n";
15. }
16. // initialisation de 2 variables avec le contenu d'un tableau
17. list($chaine1, $chaine2) = array("chaine1", "chaine2");
18. // concaténation des 2 chaînes
19. $chaine3 = $chaine1 . $chaine2;
20. // affichage résultat
21. print "[$chaine1,$chaine2,$chaine3]\n";
22. // utilisation fonction
23. affiche($chaine1);
24. // le type d'une variable peut être connu
25. afficheType("n", $n);
26. afficheType("chaine1", $chaine1);
27. afficheType("tableau", $tableau);
28. // le type d'une variable peut changer en cours d'exécution
29. $n = "a changé";
30. afficheType("n", $n);
31. // une fonction peut rendre un résultat
32. $res1 = f1(4);
33. print "res1=$res1\n";
34. // une fonction peut rendre un tableau de valeurs
35. list($res1, $res2, $res3) = f2();
36. print "(res1,res2,res3)=[$res1,$res2,$res3]\n";
37. // on aurait pu récupérer ces valeurs dans un tableau
38. $t = f2();
39. for ($i = 0; $i < count($t); $i++) {
40.   print "t[$i]=$t[$i]\n";
41. }
```

```
// des tests
for ($i = 0; $i < count($t); $i++) {
  // n'affiche que les chaînes
  if (getType($t[$i]) === "string") {
    print "t[$i]=$t[$i]\n";
  }
}
// opérateurs de comparaison == et ===
if("2"==2){
  print "avec l'opérateur ==, la chaîne 2 est égale à l'entier 2\n";
}else{
  print "avec l'opérateur ==, la chaîne 2 n'est pas égale à l'entier 2\n";
}
if("2"===2){
  print "avec l'opérateur ===, la chaîne 2 est égale à l'entier 2\n";
}
else{
  print "avec l'opérateur ===, la chaîne 2 n'est pas égale à l'entier 2\n";
}
// d'autres tests
for ($i = 0; $i < count($t); $i++) {
  // n'affiche que les entiers >10
  if (getType($t[$i]) === "integer" and $t[$i] > 10) {
    print "t[$i]=$t[$i]\n";
  }
}
// une boucle while
$t = [8, 5, 0, -2, 3, 4];
$i = 0;
$somme = 0;
while ($i < count($t) and $t[$i] > 0) {
  print "t[$i]=$t[$i]\n";
  $somme += $t[$i];   //$somme=$somme+$t[$i]
  $i++;               //$i=$i+1
}//while
print "somme=$somme\n";

// fin programme
exit;

//---------------------------------
function affiche($chaine) {
  // affiche $chaine
  print "chaine=$chaine\n";
}

//affiche
//---------------------------------
function afficheType($name, $variable) {
  // affiche le type de $variable
  print "type[variable $" . $name . "]=" . getType($variable) . "\n";
}

//afficheType
//---------------------------------
function f1($param) {
  // ajoute 10 à $param
  return $param + 10;
}

//---------------------------------
function f2() {
  // rend 3 valeurs
  return array("un", 0, 100);
}
?>
```

Traduit en Javascript, cela donne le code suivant :

```
'use strict';
// ceci est un commentaire
// constante
const nom = "dupont";
// un affichage écran
```

```
6.  console.log("nom : ", nom);
7.  // un tableau avec des éléments de type différent
8.  const tableau = ["un", "deux", 3, 4];
9.  // son nombre d'éléments
10. let n = tableau.length;
11. // une boucle
12. for (let i = 0; i < n; i++) {
13.   console.log("tableau[", i, "] = ", tableau[i]);
14. }
15. // initialisation de 2 variables avec le contenu d'un tableau
16. let [chaine1, chaine2] = ["chaine1", "chaine2"];
17. // concaténation des 2 chaînes
18. const chaine3 = chaine1 + chaine2;
19. // affichage résultat
20. console.log([chaine1, chaine2, chaine3]);
21. // utilisation fonction
22. affiche(chaine1);
23. // le type d'une variable peut être connu
24. afficheType("n", n);
25. afficheType("chaine1", chaine1);
26. afficheType("tableau", tableau);
27. // le type d'une variable peut changer en cours d'exécution
28. n = "a changé";
29. afficheType("n", n);
30. // une fonction peut rendre un résultat
31. let res1 = f1(4);
32. console.log("res1=", res1);
33. // une fonction peut rendre un tableau de valeurs
34. let res2, res3;
35. [res1, res2, res3] = f2();
36. console.log("(res1,res2,res3)=", [res1, res2, res3]);
37. // on aurait pu récupérer ces valeurs dans un tableau
38. let t = f2();
39. for (let i = 0; i < t.length; i++) {
40.   console.log("t[i]=", t[i]);
41. }
42. // des tests
43. for (let i = 0; i < t.length; i++) {
44.   // n'affiche que les chaînes
45.   if (typeof (t[i]) === "string") {
46.     console.log("t[i]=", t[i]);
47.   }
48. }
49. // opérateurs de comparaison == et ===
50. if ("2" == 2) {
51.   console.log("avec l'opérateur ==, la chaîne 2 est égale à l'entier 2");
52. } else {
53.   console.log("avec l'opérateur ==, la chaîne 2 n'est pas égale à l'entier 2");
54. }
55. if ("2" === 2) {
56.   console.log("avec l'opérateur ===, la chaîne 2 est égale à l'entier 2");
57. } else {
58.   console.log("avec l'opérateur ===, la chaîne 2 n'est pas égale à l'entier 2");
59. }
60. // d'autres tests
61. for (let i = 0; i < t.length; i++) {
62.   // n'affiche que les entiers >10
63.   if (typeof (t[i]) === "number" && Math.floor(t[i]) === t[i] && t[i] > 10) {
64.     console.log("t[i]=", t[i]);
65.   }
66. }
67. // une boucle while
68. t = [8, 5, 0, -2, 3, 4];
69. let i = 0;
70. let somme = 0;
71. while (i < t.length && t[i] > 0) {
72.   console.log("t[i]=", t[i]);
73.   somme += t[i];
74.   i++;
75. }
76. console.log("somme=", somme);
77.
78. // arrêt du programme car il n'y a plus de code exécutable
79.
```

```
80. //affiche
81. //---------------------------------
82. function affiche(chaine) {
83.   // affiche chaine
84.   console.log("chaine=", chaine);
85. }
86.
87. //afficheType
88. //---------------------------------
89. function afficheType(name, variable) {
90.   // affiche le type de variable
91.   console.log("type[variable ", name, "]=", typeof (variable));
92. }
93.
94. //---------------------------------
95. function f1(param) {
96.   // ajoute 10 à param
97.   return param + 10;
98. }
99.
100.//---------------------------------
101.function f2() {
102.  // rend 3 valeurs
103.  return ["un", 0, 100];
104.}
```

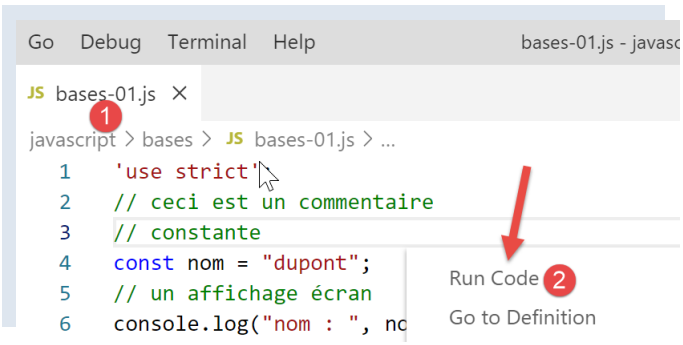
Commentons les différences entre les codes PHP et ECMAScript 6 avec la déclaration **`[use strict]`** (ligne 1) :

- la 1ère différence est qu'en ECMAScript **on déclare les variables** avec les mots clés suivants :
  - **`[let]`** pour déclarer une variable dont la valeur **peut changer** au cours de l'exécution du code ;
  - **`[const]`** pour déclarer une variable dont la valeur **ne va pas changer** (une constante donc) au cours de l'exécution du code ;
  - on peut également utiliser le mot clé **`[var]`** à la place de **`[let]`**. C'était le mot clé utilisé avec ECMAScript 5. Nous ne l'utiliserons pas dans ce cours ;
- ligne 6 : la méthode d'affichage **`[console.log]`** peut afficher toutes sortes de données : chaînes, nombres, booléens, tableaux, objets. La méthode PHP **`[print]`** ne sait pas afficher nativement des tableaux et objets. Dans l'expression **`[console.log]`**, **`[console]`** est un objet et **`[log]`** une méthode de cet objet ;
- ligne 8 : les tableaux Javascript sont des **objets référencés par un pointeur**. Lorsqu'on écrit :

```
const tableau = ["un", "deux", 3, 4];
```

  la variable **`[tableau]`** est un **pointeur** sur le tableau littéral **`["un", "deux", 3, 4]`**. Modifier le contenu du tableau ne modifie pas son pointeur. Aussi un tableau sera-t-il le plus souvent déclaré avec le mot clé **`[const]`**. En PHP, un tableau n'est pas référencé par un pointeur. C'est une donnée littérale ;
- ligne 12 : la variable de boucle **`[i]`** est déclarée (let) dans la boucle. Le mot clé **`[let]`** respecte la portée de bloc (code entre accolades). Ainsi la variable **`[i]`** de la ligne 12 n'est-elle connue que dans la boucle ;
- ligne 18 : l'opérateur de concaténation de chaîne est l'opérateur + en Javascript, . en PHP. Une particularité de cet opérateur est qu'il a précédence sur l'opérateur + d'addition. Ainsi :
  - en PHP, '1' +2 donne le nombre 3 ;
  - en Javascript '1'+2 donne la chaîne '12' ;
- ligne 20 : **`[console.log]`** sait afficher des tableaux ;
- ligne 82 : en Javascript, il n'est pas possible d'indiquer le type des paramètres d'une fonction ;
- ligne 91 : l'opérateur **`[typeof]`** permet de connaître le type d'une donnée. Il y en a quatre : nombre, chaîne de caractères, booléen et objet. On notera qu'en Javascript on n'a pas de type **`[integer]`** ni de type **`[tableau]`**. Comme il a été dit, les tableaux sont manipulés via des pointeurs et tombe dans la catégorie des objets ;
- lignes 50-59 : comme en PHP, Javascript a deux opérateurs de comparaison, == et '===' avec la même signification qu'en PHP. ESLint signale le plus souvent l'opérateur == comme une erreur possible. On utilisera systématiquement l'opérateur '===' ;
- ligne 79 : on aurait pu mettre l'instruction **`[return]`** mais ESLint émet l'avertissement que **`[return]`** ne doit s'utiliser que dans une fonction ;

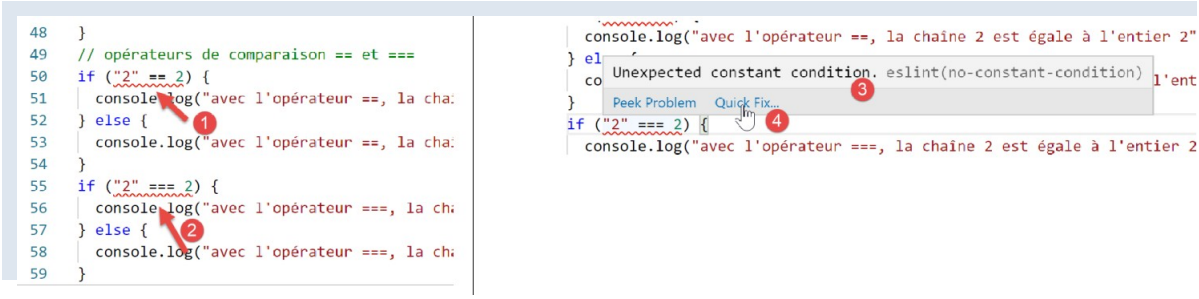
Exécutons ce code :



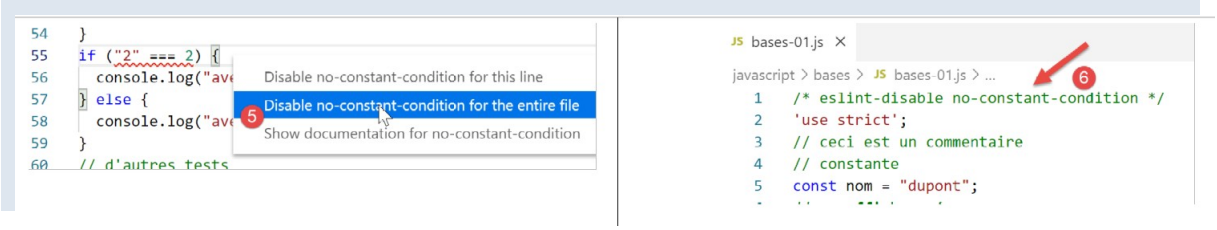
Les résultats de l'exécution :

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-
    01\javascript\bases\bases-01.js"
2.  nom : dupont
3.  tableau[ 0 ] = un
4.  tableau[ 1 ] = deux
5.  tableau[ 2 ] = 3
6.  tableau[ 3 ] = 4
7.  [ 'chaine1', 'chaine2', 'chaine1chaine2' ]
8.  chaine= chaine1
9.  type[variable n ]= number
10. type[variable chaine1 ]= string
11. type[variable tableau ]= object
12. type[variable n ]= string
13. res1= 14
14. (res1,res2,res3)= [ 'un', 0, 100 ]
15. t[ 0 ]= un
16. t[ 1 ]= 0
17. t[ 2 ]= 100
18. t[ 0 ]= un
19. avec l'opérateur ==, la chaîne 2 est égale à l'entier 2
20. avec l'opérateur ===, la chaîne 2 n'est pas égale à l'entier 2
21. t[ 2 ]= 100
22. t[ 0 ]= 8
23. t[ 1 ]= 5
24. somme= 13
25.
26. [Done] exited with code=0 in 0.316 seconds
```

Dans le code écrit, ESLINT signale deux erreurs :



- en passant le curseur sur la ligne rouge de l'avertissement, on a le message d'erreur **[3]**. Ici ESLint ne comprend pas qu'on compare deux constantes. L'un des deux opérandes devrait être une variable ;
- en **[4]**, une option **[Quick Fix]** permet de lever l'avertissement si on décide de ne pas corriger l'erreur ;



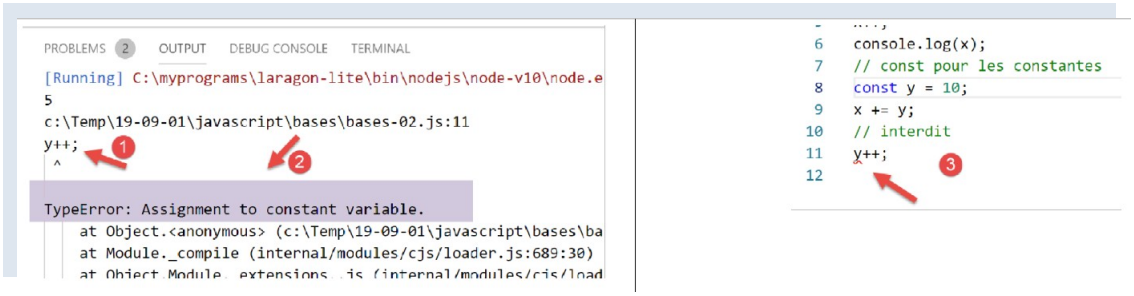
- en **[5]**, on a la possibilité de désactiver l'avertissement pour la ligne courante ou pour l'ensemble du fichier. C'est cette dernière option que nous choisissons ici . La ligne **[6]** est alors générée au début du fichier ;

3.2 script [bases-02]

Le script **[bases-02]** montre l'utilisation des mots clés **[let]** et **[const]** :

```
1.  'use strict';
2.  // pour initialiser une variable, on utilise let ou const
3.  // let pour les variables
4.  let x = 4;
5.  x++;
6.  console.log(x);
7.  // const pour les constantes
8.  const y = 10;
9.  x += y;
10. // interdit
11. y++;
```

- la ligne 11 provoque une erreur à l'exécution **[1-2]**. Elle est signalée par ESLint avant l'exécution **[3]** :



3.3 script [bases-03]

Le script **[bases-03]** examine la portée des variables en Javascript :

```
1.  'use strict';
2.  // portée des variables
3.  let count = 1;
4.  function doSomething() {
5.    // count est ici connu
6.    console.log("count=",count);
7.  }
8.  // appel
9.  doSomething();
```

- la variable **[count]** déclarée en-dehors de la fonction **[doSomething]** est pourtant connue dans cette fonction. C'est une différence fondamentale avec PHP ;

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-
    01\javascript\bases\bases-03.js"
```

```
2.  count= 1
3.
4.  [Done] exited with code=0 in 0.3 seconds
```

3.4 script [bases-04]

Une variable locale cache une variable globale de même nom :

```
1.  'use strict';
2.  // portée des variables
3.  const count = 1;
4.  function doSomething() {
5.    // la variable locale cache la variable globale
6.    const count = 2;
7.    console.log("count inside function=",count);
8.  }
9.  // variable globale
10. console.log("count outside function=",count);
11. // variable locale
12. doSomething();
```

Exécution

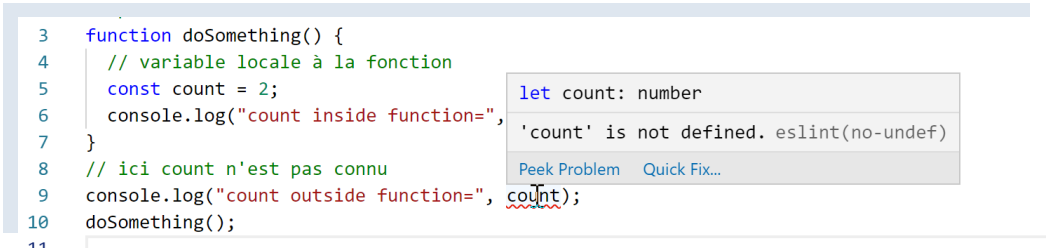
```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-
    01\javascript\bases\bases-04.js"
2.  count outside function= 1
3.  count inside function= 2
4.
5.  [Done] exited with code=0 in 0.246 seconds
```

3.5 script [bases-05]

Une variable définie dans une fonction n'est pas connue en-dehors de celle-ci :

```
1.  'use strict';
2.  // portée des variables
3.  function doSomething() {
4.    // variable locale à la fonction
5.    const count = 2;
6.    console.log("count inside function=", count);
7.  }
8.  // ici count n'est pas connu
9.  console.log("count outside function=", count);
10. doSomething();
```

ESLint déclare une erreur sur la ligne 9 :



3.6 script [bases-06]

Les mots clés **[let]** et **[const]** définissent des variables de portée **[bloc]** (code entre accolades) mais pas le mot clé **[var]** :

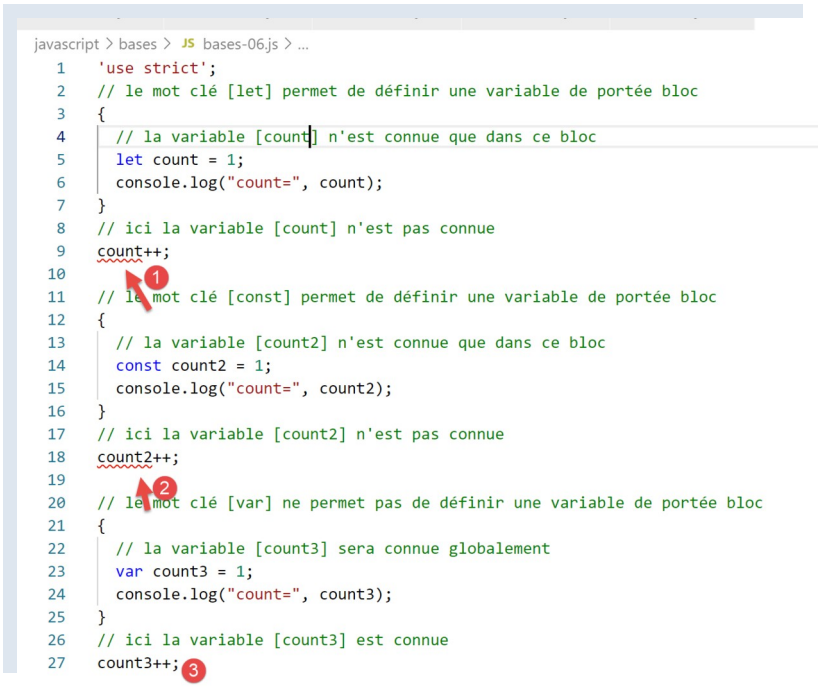
```
1.  'use strict';
2.  // le mot clé [let] permet de définir une variable de portée bloc
```

```
3.  {
4.    // la variable [count] n'est connue que dans ce bloc
5.    let count = 1;
6.    console.log("count=", count);
7.  }
8.  // ici la variable [count] n'est pas connue
9.  count++;
10.
11. // le mot clé [const] permet de définir une variable de portée bloc
12. {
13.   // la variable [count2] n'est connue que dans ce bloc
14.   const count2 = 1;
15.   console.log("count=", count2);
16. }
17. // ici la variable [count2] n'est pas connue
18. count2++;
19.
20. // le mot clé [var] ne permet pas de définir une variable de portée bloc
21. {
22.   // la variable [count3] sera connue globalement
23.   var count3 = 1;
24.   console.log("count=", count3);
25. }
26. // ici la variable [count3] est connue
27. count3++;
```

Commentaires

- ligne 5 : la variable **[count]** n'est connue que dans le bloc de code dans laquelle elle est déclarée (lignes 3-7) ;
- ligne 14 : la constante **[count2]** n'est connue que dans le bloc de code dans laquelle elle est déclarée (lignes 12-16) ;
- ligne 23 : la variable **[count3]** est connue en-dehors du bloc de code dans laquelle elle est déclarée (lignes 21-25) ;

ESLint déclare les erreurs suivantes :



Pour ces raisons de portée de bloc, nous n'utiliserons par la suite que les mots clés **[let]** et **[const]**.

3.7 script [bases-07]

Les types de données en Javascript :

```
1.  'use strict';
2.
3.  // type de données jS
4.  const var1 = 10;
5.  const var2 = "abc";
6.  const var3 = true;
7.  const var4 = [1, 2, 3];
8.  const var5 = {
9.    nom: 'axèle'
10. };
11. const var6 = function () {
12.   return +3;
13. }
14. // affichage des types
15. console.log("typeof(var1)=", typeof (var1));
16. console.log("typeof(var2)=", typeof (var2));
17. console.log("typeof(var3)=", typeof (var3));
18. console.log("typeof(var4)=", typeof (var4));
19. console.log("typeof(var5)=", typeof (var5));
20. console.log("typeof(var6)=", typeof (var6));
```

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-
    01\javascript\bases\bases-07.js"
2.  typeof(var1)= number
3.  typeof(var2)= string
4.  typeof(var3)= boolean
5.  typeof(var4)= object
6.  typeof(var5)= object
7.  typeof(var6)= function
8.
9.  [Done] exited with code=0 in 0.26 seconds
```

Commentaires

- ligne 7 (code) : un tableau est un objet. A ce titre **[var4]** est un pointeur vers le tableau, pas le tableau lui-même ;
- ligne 8 (code) : **[var5]** est un pointeur vers un objet littéral. On verra que les objets littéraux de Javascript ressemblent fort aux instances de classe de PHP. Eux-aussi sont référencés via des pointeurs ;
- ligne 11 (code) : une variable peut être de type **[fonction]** (ligne 7 des résultats) ;

3.8 script [bases-08]

Ce script montre les changements de type possibles en Javascript.

```
1.  'use strict';
2.
3.  // changements implicites de types
4.  // type -->bool
5.  console.log("---------------[Conversion implicite vers un
    booléen]-----------------------------");
6.  showBool("abcd");
7.  showBool("");
8.  showBool([1, 2, 3]);
9.  showBool([]);
10. showBool(null);
11. showBool(0.0);
12. showBool(0);
13. showBool(4.6);
14. showBool({});
15. showBool(undefined);
16.
17. function showBool(data) {
18.   // la conversion de data en booléen se fait automatiquement dans le test qui suit
```

```
19.   console.log("[data=", data, "], [type(data)]=", typeof (data), "[valeur booléenne(data)]=",
    data ? true : false);
20. }
21.
22. // changements implicites de type  vers un type numérique
23. console.log("---------------[Conversion implicite vers un
    nombre]------------------------------");
24. showNumber("12");
25. showNumber("45.67");
26. showNumber("abcd");
27.
28. function showNumber(data) {
29.   // data + 1 ne marche pas car alors jS fait une concaténation de chaînes plutôt qu'une
    addition
30.   const nombre = data * 1;
31.   console.log("[data=", data, "], [type(data)]=", typeof (data), "[nombre]=", nombre,
    "[type(nombre)]=", typeof (nombre));
32. }
33.
34. // changements explicites de types vers un booléen
35. console.log("---------------[Conversion explicite vers un
    booléen]------------------------------");
36. showBool2("abcd");
37. showBool2("");
38. showBool2([1, 2, 3]);
39. showBool2([]);
40. showBool2(null);
41. showBool2(0.0);
42. showBool2(0);
43. showBool2(4.6);
44. showBool2({});
45. showBool2(undefined);
46.
47. function showBool2(data) {
48.   // la conversion de data en booléen se fait explicitement dans le test qui suit
49.   console.log("[", data, "], [type(data)]=", typeof (data), "[valeur booléenne(data)]=",
    Boolean(data));
50. }
51. // changements explicites de type vers Number
52. console.log("---------------[Conversion explicite vers un
    nombre]------------------------------");
53. showNumber2("12.45");
54. showNumber2(67.8);
55. showNumber2(true);
56. showNumber2(null);
57.
58. function showNumber2(data) {
59.   const nombre = Number(data);
60.   console.log("[data=", data, "], [type(data)]=", typeof (data), "[nombre]=", nombre,
    "[type(nombre)]=", typeof (nombre));
61. }
62.
63. // vers String
64. console.log("---------------[Conversion explicite vers un
    string]------------------------------");
65. showString(5);
66. showString(6.7);
67. showString(false);
68. showString(null);
69.
70. function showString(data) {
71.   const chaîne = String(data);
72.   console.log("[data=", data, "], [type(data)]=", typeof (data), "[chaîne]=", chaîne,
    "[type(chaîne)]=", typeof (chaîne));
73. }
74.
75. // qqs conversions implicites inattendues
76. console.log("---------------[Autres cas]------------------------------");
77. const string1 = '1000.78';
78. // concaténation de chaînes par défaut
79. const data1 = string1 + 1.034;
80. console.log("data1=", data1, "type=", typeof (data1));
81. const data2 = 1.034 + string1;
82. console.log("data2=", data2, "type=", typeof (data2));
```

```
// conversion explicite vers nombre
const data3 = Number(string1) + 1.034;
console.log("data3=", data3, "type=", typeof (data3));
// true est converti en le nombre 1
const data4 = true * 1.18;
console.log("data4=", data4, "type=", typeof (data4));
// false est converti en le nombre 0
const data5 = false * 1.18;
console.log("data5=", data5, "type=", typeof (data5));
```

Exécution

```
[Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:
\Data\st-2019\dev\es6\javascript\bases\bases-08.js"
---------------[Conversion implicite vers un booléen]------------------------------
[data= abcd ], [type(data)]= string [valeur booléenne(data)]= true
[data= ], [type(data)]= string [valeur booléenne(data)]= false
[data= [ 1, 2, 3 ] ], [type(data)]= object [valeur booléenne(data)]= true
[data= [] ], [type(data)]= object [valeur booléenne(data)]= true
[data= null ], [type(data)]= object [valeur booléenne(data)]= false
[data= 0 ], [type(data)]= number [valeur booléenne(data)]= false
[data= 0 ], [type(data)]= number [valeur booléenne(data)]= false
[data= 4.6 ], [type(data)]= number [valeur booléenne(data)]= true
[data= {} ], [type(data)]= object [valeur booléenne(data)]= true
[data= undefined ], [type(data)]= undefined [valeur booléenne(data)]= false
---------------[Conversion implicite vers un nombre]------------------------------
[data= 12 ], [type(data)]= string [nombre]= 12 [type(nombre)]= number
[data= 45.67 ], [type(data)]= string [nombre]= 45.67 [type(nombre)]= number
[data= abcd ], [type(data)]= string [nombre]= NaN [type(nombre)]= number
---------------[Conversion explicite vers un booléen]------------------------------
[ abcd ], [type(data)]= string [valeur booléenne(data)]= true
[ ], [type(data)]= string [valeur booléenne(data)]= false
[ [ 1, 2, 3 ] ], [type(data)]= object [valeur booléenne(data)]= true
[ [] ], [type(data)]= object [valeur booléenne(data)]= true
[ null ], [type(data)]= object [valeur booléenne(data)]= false
[ 0 ], [type(data)]= number [valeur booléenne(data)]= false
[ 0 ], [type(data)]= number [valeur booléenne(data)]= false
[ 4.6 ], [type(data)]= number [valeur booléenne(data)]= true
[ {} ], [type(data)]= object [valeur booléenne(data)]= true
[ undefined ], [type(data)]= undefined [valeur booléenne(data)]= false
---------------[Conversion explicite vers un nombre]------------------------------
[data= 12.45 ], [type(data)]= string [nombre]= 12.45 [type(nombre)]= number
[data= 67.8 ], [type(data)]= number [nombre]= 67.8 [type(nombre)]= number
[data= true ], [type(data)]= boolean [nombre]= 1 [type(nombre)]= number
[data= null ], [type(data)]= object [nombre]= 0 [type(nombre)]= number
---------------[Conversion explicite vers un string]------------------------------
[data= 5 ], [type(data)]= number [chaîne]= 5 [type(chaîne)]= string
[data= 6.7 ], [type(data)]= number [chaîne]= 6.7 [type(chaîne)]= string
[data= false ], [type(data)]= boolean [chaîne]= false [type(chaîne)]= string
[data= null ], [type(data)]= object [chaîne]= null [type(chaîne)]= string
---------------[Autres cas]------------------------------
data1= 1000.781.034 type= string
data2= 1.0341000.78 type= string
data3= 1001.814 type= number
data4= 1.18 type= number
data5= 0 type= number
```

4 Les tableaux

4.1 script [tab-01]

Le script suivant illustre certaines caractéristiques des tableaux Javascript. Ceux-ci ressemblent aux tableaux PHP avec cependant une grande différence : ils sont manipulés via des pointeurs et sont considérés comme des objets.

```
1.  'use strict';
2.
3.  // un tableau est un objet manipulé via son adresse
4.  const tab1 = [1, 2, 3];
5.  // copie d'adresses
6.  const tab2 = tab1;
7.  // tab1 et tab2 pointent sur le même tableau
8.  console.log("tab1===tab2 :", tab1 === tab2);
9.  // on peut modifier le tableau en passant indifféremment par tab1 ou tab2
10. tab2[1] = 10;
11. console.log("tab1=", tab1);
12. console.log("tab2=", tab2);
```

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-
    01\javascript\tableaux\tab-01.js"
2.  tab1===tab2 : true
3.  tab1= [ 1, 10, 3 ]
4.  tab2= [ 1, 10, 3 ]
```

Commentaires

- la ligne 2 des résultats montre que **[tab1]** et **[tab2]** sont deux entités égales, en fait deux pointeurs égaux ;
- les lignes 4 et 5 des résultats montrent que ces deux pointeurs pointent sur le même tableau ;

4.2 script [tab-02]

Ce script montre que le tableau Javascript est différent des tableaux des langages compilés.

```
1.  'use strict';
2.
3.  // tableau
4.  const tab = [];
5.  console.log("tab=", tab, ", longueur=[", tab.length, "]");
6.  console.log("-------------------------------");
7.  // initialisation d'un élément
8.  tab[3] = 100;
9.  tab[1] = "huit";
10. // tableau
11. console.log("tab=", tab, ", longueur=[", tab.length, "]");
12. console.log("-------------------------------");
13. // toString
14. console.log("tab.toString=[", tab.toString(), "]");
15. console.log("-------------------------------");
16. // les clés du tableau sont ses indices
17. for (let key of tab.keys()) {
18.   console.log("clé=[", key, "], valeur=[", tab[key], "]");
19. }
20. console.log("-------------------------------");
```

```
21. // les valeurs du tableau
22. for (let value of tab.values()) {
23.   console.log("valeur=[", value, "]");
24. }
```

- ligne 4 : un tableau vide ;
- ligne 8 : un tableau n'a pas de taille fixe. C'est simplement une suite d'éléments indexée par un n°. On peut initialiser l'élément n° 3 même si les éléments **[0, 1, 2]** ne sont pas encore définis ;
- lignes 16-24 : un tableau Javascript a un comportement analogue à celui du tableau PHP ;

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-
    01\javascript\tableaux\tab-02.js"
2.  tab= [] , longueur=[ 0 ]
3.  ------------------------------
4.  tab= [ <1 empty item>, 'huit', <1 empty item>, 100 ] , longueur=[ 4 ]
5.  ------------------------------
6.  tab.toString=[ ,huit,,100 ]
7.  ------------------------------
8.  clé=[ 0 ], valeur=[ undefined ]
9.  clé=[ 1 ], valeur=[ huit ]
10. clé=[ 2 ], valeur=[ undefined ]
11. clé=[ 3 ], valeur=[ 100 ]
12. ------------------------------
13. valeur=[ undefined ]
14. valeur=[ huit ]
15. valeur=[ undefined ]
16. valeur=[ 100 ]
```

4.3 script [tab-03]

Ce script montre différentes méthodes de l'objet **[tableau]**.

```
1.  'use strict';
2.  // un tableau peut contenir différents types de données
3.  const tab = [1, 2, "un", "deux", true, [10, 20], { prop1: 10, prop2: "abc" }];
4.  // console.log sait afficher le contenu d'un tableau
5.  show(1);
6.  console.log("tab=", tab);
7.  show(2);
8.  // parcours du tableau avec foreach
9.  tab.forEach(element => {
10.   console.log("élément=", element, typeof (element));
11. });
12. show("2b");
13. // une autre écriture pour faire la même chose
14. tab.forEach(function (element) {
15.   console.log("élément=", element, typeof (element));
16. });
17. show(3);
18. // parcours du tableau avec for
19. for (let i = 0; i < tab.length; i++) {
20.   console.log("i=", i, "tab[i]=", tab[i]);
21. }
22. show(4);
23. // modification tab[i]
24. tab[5] = [];
25. // affichage
26. console.log("tab=", tab);
27. show(5);
28. // on enlève le dernier élément
29. let element = tab.pop(tab);
30. console.log("élément=", element, "tab=", tab);
31. show(6);
32. // on ajoute un élément à la fin du tableau
33. tab.push('xyz');
34. console.log("tab=", tab);
35. show(7);
36. // on ajoute un élément au début du tableau
37. tab.unshift(1000);
38. console.log("tab=", tab);
```

```
39. show(8);
40. // on enlève le 1er élément du tableau
41. element = tab.shift();
42. console.log("élément=", element, "tab=", tab);
43. show(9);
44. // on enlève l'élément n° 2 du tableau
45. element = tab.splice(2, 1);
46. console.log("élément=", element, "tab=", tab);
47. show(10);
48. // on enlève du tableau deux éléments à partir de l'élément n° 1
49. element = tab.splice(1, 2);
50. console.log("élément=", element, "tab=", tab);
51.
52. // fonction
53. function show(param) {
54.   console.log("[", param,
    "::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: ]");
55. }
```

Commentaires

- la différence entre tableaux PHP et tableaux Javascript est illustrée par les lignes 3 et 24 :
  - la ligne 3 déclare la variable **`[tab]`** comme une constante ;
  - la ligne 24 modifie l'élément tab**`[5]`** ;

  Ligne 3, c'est le pointeur qui pointe sur le tableau qui est déclaré constant, ce n'est pas le tableau lui-même. Celui-ci peut être modifié.
- lignes 14-16 : le tableau **`[tab]`** est parcouru à l'aide d'une méthode **`[forEach]`** du tableau **`[tab]`**. Cette méthode reçoit en paramètre la définition d'une fonction, qu'on pourrait appeler une fonction littérale. Cette fonction paramètre reçoit un paramètre : l'élément courant du tableau **`[tab]`**. La fonction est appelée pour chaque élément du tableau. Ce type d'écriture est courant en Javascript ;
- lignes 9-10 : on utilise une autre syntaxe pour définir la fonction paramètre. Au lieu d'écrire :
  - **function(p1, p2, …, pn){….}**

  on écrit :
  - **(p1,p2, ..,pn)=>{….}**. On appelle cela, la notation « flèche » ou « arrow » ;
- le reste du code est expliqué par les commentaires ;

De ce script, on notera que :

- un tableau est un objet référencé par un pointeur ;
- que cet objet a des méthodes **`[forEach, pop, push, shift, unshift]`** ;

4.4 script `[tab-04]`

Ce script présente d'autres méthodes des objets tableau.

```
1.  'use strict';
2.
3.  // méthode de manipulation de tableaux
4.
5.  // un tableau
6.  const tab = [];
7.  for (let i = 0; i < 10; i++) {
8.    tab[i] = i * 10;
9.  }
10. // affichage
11. console.log("tab=", tab);
12. // map
13. const tab2 = tab.map(element => {
14.   return { prop1: element, prop2: element * element }
15. });
16. // affichage
17. console.log("tab=", tab);
18. console.log("tab2=", tab2);
19. // reduce sans valeur initiale
20. const somme = tab.reduce((accumulator, currentValue) => accumulator + currentValue);
21. console.log("somme tab=", somme);
22. // reduce avec valeur initiale
23. const somme2 = tab.reduce((accumulator, currentValue) => accumulator + currentValue, 10);
24. console.log("somme2 tab=", somme2);
25. // filter
```

```
26. const tab4 = tab.filter((element) => {
27.   if (element > 50) {
28.     return element;
29.   }
30. });
31. console.log("tab4=", tab4);
32. // find
33. const element1 = tab.find((element) => (element > 20));
34. console.log("élément1=", element1);
35. // findIndex
36. const index1 = tab.findIndex((element) => (element === 20));
37. console.log("index1 20=", index1);
38. // indexOf
39. const index2 = tab.indexOf(30);
40. console.log("index2 30=", index2);
41. const index3 = tab.indexOf(31);
42. console.log("index3 31=", index3);
43. // lastIndexOf
44. const index4 = [4, 5, 4, 2].lastIndexOf(4);
45. console.log("index4 4=", index4);
46. // sort
47. const tab5 = [4, 5, 4, 2].sort();
48. console.log("tab5=", tab5);
49. // sort inverse
50. const tab6 = [4, 5, 4, 2].sort((e1, e2) => {
51.   if (e1 > e2) {
52.     return -1;
53.   }
54.   else if (e1 === e2) {
55.     return 0;
56.   } else {
57.     return +1;
58.   }
59. });
60. console.log("tab6=", tab6);
```

Commentaires

- lignes 13-15 : la méthode **[map]** admet une fonction de transformation comme paramètre. Celle-ci est appelée de façon répétée pour chaque élément du tableau. Elle est chargée de transformer celui-ci en autre chose, ici un objet avec les propriétés **[prop1, prop2]**. La méthode **[map]** rend un nouveau tableau. L'ancien n'est pas modifié :

```
tab= [ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90 ]
tab2= [ { prop1: 0, prop2: 0 },
{ prop1: 10, prop2: 100 },
{ prop1: 20, prop2: 400 },
{ prop1: 30, prop2: 900 },
{ prop1: 40, prop2: 1600 },
{ prop1: 50, prop2: 2500 },
{ prop1: 60, prop2: 3600 },
{ prop1: 70, prop2: 4900 },
{ prop1: 80, prop2: 6400 },
{ prop1: 90, prop2: 8100 } ]
```

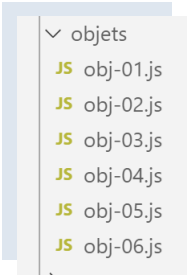
- ligne 20 : la méthode **[reduce]** admet pour paramètre une fonction à deux paramètres appelée de façon répétée pour chaque élément du tableau. Cette fonction admet deux paramètres :
  - **[currentValue]** est l'élément courant du tableau ;
  - **[accumulator]** est le dernier résultat obtenu par la fonction. Si aucune valeur initiale n'est prévue pour cet accumulateur, alors celle-ci sera 0 ;
  - la 1ère fois que la fonction d'accumulation est appelée, elle rend **[0+tab[0]]**. C'est cette valeur qui est affectée à l'accumulateur ;
  - la seconde fois, elle rend accumulateur+tab**[1]**, donc tab**[0]**+tab**[1]** ;
  - la troisème fois, elle rend accumulateur+tab**[2]**, donc tab**[0]**+tab**[1]**+tab**[2]** ;
  - etc. Au final, l'accumulateur représentera la somme de tous les éléments du tableau **[tab]** ;
- ligne 26 : la méthode **[filter]** a pour paramètre une fonction de filtrage. Celle-ci est appelée de façon répétée pour chaque élément du tableau et reçoit celui-ci en paramètre. Elle doit rendre :
  - **[true]** si l'élément doit être gardé ;
  - **[false]** sinon ;

- ligne 33 : la méthode `[find]` a pour paramètre une fonction de recherche. Celle-ci est appelée de façon répétée pour chaque élément du tableau et reçoit celui-ci en paramètre. Elle doit rendre `[true]` si l'élément reçu satisfait le critère de recherche. Celle-ci s'arrête alors. La méthode `[find]` rend donc 0 ou 1 élément ;
- ligne 36 : la méthode `[findIndex]` fonctionne comme la méthode `[find]` mais au lieu de rendre l'élément trouvé, elle rend son index dans le tableau ;
- lignes 39, 41, la méthode `[indexOf(valeur)]` recherche `[valeur]` dans le tableau et rend son index, ou -1 s'il n'est pas trouvé ;
- ligne 44 : la méthode `[lastIndexOf(valeur)]` fonctionne comme la méthode `[indexOf(valeur)]` mais commence sa recherche par la fin du tableau ;
- ligne 47 : la méthode `[sort]` sans paramètres rend un tableau trié dans l'ordre naturel (nombres, chaînes) ;
- ligne 50 : lorsque l'ordre naturel ne convient pas, il faut passer à la méthode `[sort]` une fonction à deux paramètres (e1,e2) qui rend :
  - +1 si e1 doit être classé après e2 ;
  - -1 si e1 doit être classé avant e2 ;
  - 0 si les deux éléments doivent avoir le même classement ;

  La fonction passée en paramètre à la méthode `[sort]` est appelée de façon répétée par celle-ci pour comparer deux éléments du tableau ;

5 Les objets littéraux

Nous appelons ici 'objets littéraux' des objets définis littéralement dans le code. Javascript a la notion de classe, et d'objet instance de classe. Ce n'est donc pas ce type d'objet dont nous parlons maintenant.



5.1 script [obj-01]

Nous présentons ici les premières propriétés des objets littéraux. La principale est que l'objet est manipulé via un pointeur.

```
1.  'use strict';
2.  // un objet vide
3.  const obj1={};
4.  // on peut créer dynamiquement les propriétés de l'objet
5.  obj1.prop1="abcd";
6.  console.log('obj1=',obj1);
7.  // autre propriété
8.  obj1.prop2=[1,2,3];
9.  console.log("obj1=",obj1);
10. // autre propriété avec une notation différente
11. obj1['prop3']=true;
12. console.log("obj1=",obj1);
13. // obj1 est une référence sur l'objet (pointeur), pas l'objet lui-même
14. const obj2=obj1;
15. // obj2 et obj1 pointent sur le même objet
16. obj2.prop1="xyzt";
17. console.log("obj1=",obj1);
18. console.log("obj2=",obj2);
19. // les propriété peuvent être des variables
20. const var1='prop1';
21. console.log('prop1=',obj1[var1]);
```

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-
    01\javascript\objets\obj-01.js"
2.  obj1=[object Object]
3.  obj1= { prop1: 'abcd', prop2: [ 1, 2, 3 ] }
4.  obj1= { prop1: 'abcd', prop2: [ 1, 2, 3 ], prop3: true }
5.  obj1= { prop1: 'xyzt', prop2: [ 1, 2, 3 ], prop3: true }
6.  obj2= { prop1: 'xyzt', prop2: [ 1, 2, 3 ], prop3: true }
7.  prop1= xyzt
```

Commentaires

- ligne 3 du code : un objet est manipulé via un pointeur. Donc `[obj1]` est un pointeur. Modifier l'objet pointé ne modifie pas le pointeur `[obj1]`. C'est pourquoi, comme pour les tableaux, une référence d'objet est déclarée avec le mot clé `[const]` ;
- ligne 6 du code : comme pour les tableaux, `[console.log]` sait afficher des objets ;
- ligne 11 du code : obj1.prop3 peut être réécrit obj1.`['prop3']`. Cette dernière notation est utile lorsque 'prop3' est en fait une variable (lignes 20-21) ;
- lignes 13-18 du code : montrent que l'instruction `[obj2=obj1]` est une copie de référence d'objet et non de l'objet lui-même ;

5.2 script [obj-02]

Ce script montre que les propriétés d'un objet peuvent avoir pour valeur un objet. On a alors des objets multi-niveaux. On introduit également l'objet global **[JSON]** qui permet de faire des conversions objet ↔ chaîne de caractères.

```
1.  'use strict';
2.  // un objet à plusieurs niveaux
3.  const personne = {
4.    prénom: "martin",
5.    âge: 12,
6.    père: {
7.      prénom: "paul",
8.      âge: 45
9.    },
10.   mère: {
11.     prénom: "micheline",
12.     âge: 42
13.   }
14. }
15. // accès aux propriétés
16. console.log("prénom personne=", personne.prénom);
17. console.log("prénom mère=", personne.mère.prénom);
18. personne.mère.âge = 40;
19. console.log("âge mère=", personne.mère.âge);
20. // console.log sait afficher des objets
21. console.log("personne=", personne);
22. console.log("mère=", personne.mère);
23. // on peut aussi afficher la chaîne jSON de l'objet
24. let json = JSON.stringify(personne);
25. console.log("jSON=", json);
26. // on peut relire le jSON
27. let personne2 = JSON.parse(json);
28. console.log("père=", personne2.père);
```

Commentaires

- ligne 24 : transformation d'un objet Javascript en chaîne jSON ;
- ligne 27 : transformation d'une chaîne jSON en objet Javascript ;

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-
    01\javascript\objets\obj-02.js"
2.  prénom personne= martin
3.  prénom mère= micheline
4.  âge mère= 40
5.  personne= { 'prénom': 'martin',
6.  'âge': 12,
7.  'père': { 'prénom': 'paul', 'âge': 45 },
8.  'mère': { 'prénom': 'micheline', 'âge': 40 } }
9.  mère= { 'prénom': 'micheline', 'âge': 40 }
10. jSON= {"prénom":"martin","âge":12,"père":{"prénom":"paul","âge":45},"mère":
    {"prénom":"micheline","âge":40}}
11. père= { 'prénom': 'paul', 'âge': 45 }
```

Commentaires

- ligne 10 : dans une chaîne jSON, les propriétés sont obligatoirement entourées de guillemets ainsi que les valeurs de type chaîne de caractères ;

5.3 script [obj-03]

Ce script introduit la notion de getter / setter d'une propriété d'un objet :

```
1.  'use strict';
2.  // getters et setters d'un objet
3.  const personne = {
4.    // getter
5.    get nom() {
```

```
6.      console.log("getter nom");
7.      return this._nom;
8.    },
9.    // setter
10.   set nom(unNom) {
11.     console.log("setter nom");
12.     this._nom = unNom;
13.   }
14. };
15. // setter
16. personne.nom = "Hercule";
17. // getter
18. console.log(personne.nom);
19. // l'objet lui-même
20. console.log("personne=", personne);
21. // ça n'empêche pas d'accéder à la propriété [_nom] directement
22. personne._nom = "xyz";
23. console.log("personne=", personne);
```

Commentaires

- lignes 5-7 : définition d'un **[getter]**, une fonction qui rend généralement la valeur d'une propriété de l'objet mais qui en fait peut rendre n'importe quoi. Le mot clé **[function]** est remplacé par le mot clé **[get]** ;
- ligne 7 : le getter rend la propriété **[_nom]**. On voit que celle-ci n'a pas besoin d'être déclarée ;
- lignes 10-13 : définition d'un **[setter]**, une fonction qui affecte généralement la valeur reçue à une propriété de l'objet mais qui en fait peut faire n'importe quoi. Le mot clé **[function]** est remplacé par le mot clé **[set]**. Le **[setter]** peut être utilisé pour vérifier la validité de la valeur passée en paramètre au **[setter]** ;
- ligne 16 : la fonction **[set nom]** va être appelée implicitement ;
- ligne 18 : la fonction **[get nom]** va être appelée implicitement ;
- ligne 22 : montre que l'utilisation des getter / setter dépend de la bonne volonté du développeur. Si celui-ci connaît le nom de la propriété gérée par ceux-ci, il peut y accéder directement ;

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-
    01\javascript\objets\obj-03.js"
2.  setter nom
3.  getter nom
4.  Hercule
5.  personne= { nom: [Getter/Setter], _nom: 'Hercule' }
6.  personne= { nom: [Getter/Setter], _nom: 'xyz' }
```

On notera lignes 5-6, que **[console.log]** affiche également les propriétés qui sont des fonctions.

5.4 script [obj-04]

Ce script montre trois façons d'écrire les noms des propriétés d'un objet et deux façons d'y accéder.

```
1.  'use strict';
2.  // les noms des propriétés d'un objet  peuvent être littéraux [nom], être entourés
    d'apostrophes ['nom']
3.  // ou de guillements ["nom"]
4.
5.  // littéraux
6.  const obj1 = {
7.    nom: "martin",
8.    prénom: "jean"
9.  };
10. console.log("prénom=", obj1.prénom);
11.
12. // entourés d'apostrophes
13. const obj2 = {
14.   'nom': "martin",
15.   'prénom': "jean"
16. };
17. console.log("nom=", obj2.nom);
18.
19. // entourés de guillemets
20. const obj3 = {
21.   "nom": "martin",
```

```
22.   "prénom": "jean"
23. };
24.
25. // deux syntaxes possibles pour accéder à la propriété [nom]
26. console.log("nom=", obj3.nom);
27. console.log("nom=", obj3['nom']);
```

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-
    01\javascript\objets\obj-04.js"
2.  prénom= jean
3.  nom= martin
4.  nom= martin
5.  nom= martin
```

5.5 script [obj-05]

Le script montre que les propriétés d'un objet littéral peuvent être des fonctions. On est alors très proche de l'objet instance de classe, où on a des propriétés et des méthodes.

```
1.  'use strict';
2.
3.  // un objet peut avoir des propriétés de type [function]
4.  const personne = {
5.    // propriétés
6.    prénom: "martin",
7.    âge: 12,
8.    père: {
9.      prénom: "paul",
10.     âge: 45
11.   },
12.   mère: {
13.     prénom: "micheline",
14.     âge: 42
15.   },
16.   // méthode
17.   toString: function () {
18.     return JSON.stringify(this);
19.   }
20. }
21.
22. // usage
23. console.log("personne=", personne);
24. console.log("personne.toString=", personne.toString());
```

- lignes 17-19 : une méthode interne à l'objet. Dans celle-ci, on accède aux propriétés de l'objet via le mot clé **[this]** (ligne 18). **[this]** désigne l'objet lui-même, **[this.prénom]**, la propriété **[prénom]** de celui-ci ;

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-
    01\javascript\objets\obj-05.js"
2.  personne= { 'prénom': 'martin',
3.  'âge': 12,
4.  'père': { 'prénom': 'paul', 'âge': 45 },
5.  'mère': { 'prénom': 'micheline', 'âge': 42 },
6.  toString: [Function: toString] }
7.  personne.toString= {"prénom":"martin","âge":12,"père":{"prénom":"paul","âge":45},"mère":
    {"prénom":"micheline","âge":42}}
```

5.6 script [obj-06]

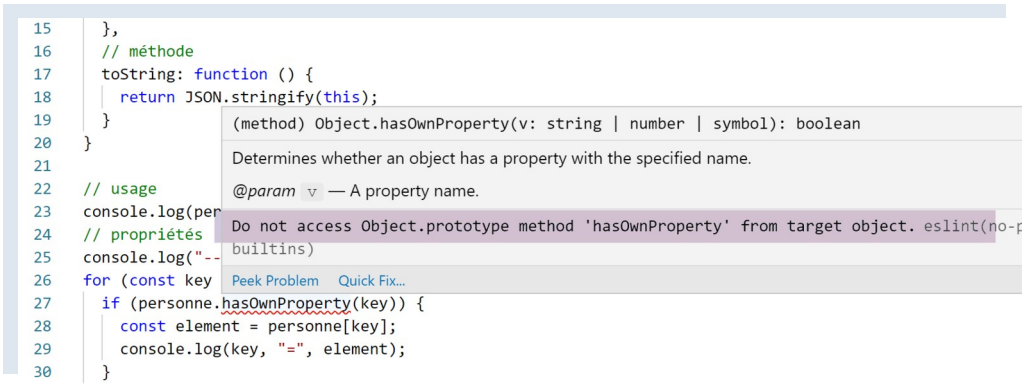
Ce script montre comment avoir accès aux propriétés d'un objet lorsqu'on ne connaît pas a priori le nom de celles-ci.

```
1.  'use strict';
2.
3.  // un objet peut avoir des propriétés de type [function]
4.  let personne = {
5.    // propriétés
```

```
6.    prénom: "martin",
7.    âge: 12,
8.    père: {
9.      prénom: "paul",
10.     âge: 45
11.   },
12.   mère: {
13.     prénom: "micheline",
14.     âge: 42
15.   },
16.   // méthode
17.   toString: function () {
18.     return JSON.stringify(this);
19.   }
20. }
21.
22. // usage
23. console.log(personne);
24. // propriétés
25. console.log("-----------------------");
26. for (const key in personne) {
27.   if (personne.hasOwnProperty(key)) {
28.     const element = personne[key];
29.     console.log(key, "=", element);
30.   }
31. }
32. // pour échapper à l'avertissement eslint (1)
33. console.log("-----------------------");
34. for (const key in personne) {
35.   if (Object.prototype.hasOwnProperty.call(personne, key)) {
36.     const element = personne[key];
37.     console.log(key, "=", element);
38.   }
39. }
40. // pour échapper à l'avertissement eslint (2)
41. console.log("-----------------------");
42. for (const key in personne) {
43.   // eslint-disable-next-line no-prototype-builtins
44.   if (personne.hasOwnProperty(key)) {
45.     const element = personne[key];
46.     console.log(key, "=", element);
47.   }
48. }
```

Commentaires

- lignes 26-31 : le code qui permet d'avoir la liste des propriétés, sans les méthodes, d'un objet. Ce code fait l'objet d'un avertissement d'ESLint :



- lignes 32-39 : le code qui permet d'échapper à l'avertissement d'ESLint. On passe par le prototype de la classe **`[Object]`** ;
- lignes 41-47 : ou bien on se contente de désactiver l'avertissement (ligne 43) ;

5.7 script [obj-07]

Le script [obj-07] montre la possibilité de déstructurer un objet :

```
1.  'use strict';
2.  // déstructuration
3.
4.  // littéraux
5.  const obj1 = {
6.    nom: "martin",
7.    prénom: "jean"
8.  };
9.
10. // déstructuration obj1 dans variables [n,p]
11. const { nom: n, prénom: p } = obj1;
12. console.log("n=", n, "p=", p);
13.
14. // déstructuration obj1 dans variables [n2,p2]
15. function f({ nom: n2, prénom: p2 }) {
16.   console.log("f-n2=", n2, "f-p2=", p2);
17. }
18. f(obj1);
19.
20. // déstructuration obj1 dans variables [nom,prénom]
21. function g({ nom: nom, prénom: prénom }) {
22.   console.log("g-nom=", nom, "g-prénom=", prénom);
23. }
24. g(obj1);
25.
26. // déstructuration obj1 dans variables [nom,prénom]
27. // avec notation raccourcie équivalente à h({nom:nom,prénom:prénom})
28. function h({ nom, prénom }) {
29.   console.log("h-nom=", nom, "h-prénom=", prénom);
30. }
31. h(obj1);
```

Commentaires

- ligne 11 : ce sont les accolades {} qui permettent la déstructuration. La syntaxe

  ```
  const { nom: n, prénom: p } = obj1
  ```

  crée deux variables [n] et [p] et est équivalente à :

  ```
  const n = obj1.nom
  const p = obj1.prénom
  ```

  La déclaration pourrait se lire de la façon suivante :

  ```
  const { nom => n, prénom => p } = obj1
  ```

  pour rappeler que les valeurs des attributs [nom, prénom] vont dans les variables [n, p] ;

- l'opération de déstructuration se répète aux lignes 15, 21 et 28. A chaque fois, c'est la présence des accolades {} qui indique qu'il va y avoir déstructuration d'un objet dans des variables ;
- la ligne 28 peut être déconcertante. C'est un raccourci pour la notation :

  ```
  function h({ nom : nom, prénom : prénom })
  ```

Les résultats de l'exécution sont les suivants :

```
1.  n= martin p= jean
2.  f-n2= martin f-p2= jean
3.  g-nom= martin g-prénom= jean
4.  h-nom= martin h-prénom= jean
```

5.8 script [obj-08]

Le script [obj-08] montre comment obtenir une copie d'un objet :

```
1.  'use strict'
2.
3.  // clônage d'objets
4.  const obj1 = {
5.    nom: "martin",
6.    prénom: "jean"
7.  };
8.
9.  // clône (copie) de obj1 avec l'opérateur de spread
10. const obj2 = { ...obj1 }
11.
12. // vérifications
13. // obj2 pointe sur une copie de obj1
14. console.log("obj2===obj1 :", obj1 === obj2)
15. console.log("obj2=", obj2)
```

- ligne 10 : l'opération de copie de l'objet [obj1]. L'opérateur ... est appelé opérateur de spread ;

Les résultats de l'exécution sont les suivants :

```
1.  obj2===obj1 : false
2.  obj2= { nom: 'martin', 'prénom': 'jean' }
```

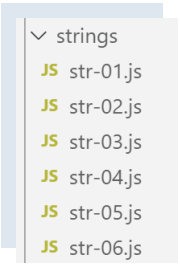
- ligne 1 : montre que les références [obj1] et [obj2] ne pointent pas sur le même objet ;
- ligne 2 : montre que l'objet pointé par [obj2] est une copie de l'objet pointé par [obj1] ;

5.9 Conclusion

Les scripts de cette section ont montré que l'objet littéral de Javascript est proche de l'objet instance de classe des langages à objets. On peut y définir propriétés, méthodes et getters / setters. C'est un objet dynamique dont on peut définir les propriétés à l'exécution. Il se comporte alors comme un dictionnaire dont les éléments peuvent être de tout type et notamment de type **`[fonction]`**.

6 Les chaînes de caractères

Les chaînes de caractères de Javascript sont très semblables à celles de PHP.



6.1 script [str-01]

La première chose à comprendre est qu'une fois une chaîne créée, elle n'est plus modifiable. On dispose de nombreuses méthodes pour produire une nouvelle chaîne à partir de la chaîne initiale mais celle-ci reste toujours inchangée. Par ailleurs, une chaîne de caractères peut être de deux types :

- **[string]** lorsqu'elle est initialisée avec une chaîne littérale ;
- **[object]** lorsqu'elle est créée comme instance de la classe **[String]** ;

```
1.  'use strict';
2.
3.  // les chaînes de caractères sont en lecture seule (on ne peut pas les modifier)
4.
5.  // une chaîne
6.  const chaîne1 = "abcd ";
7.  // type
8.  console.log("typeof(chaîne1)=", typeof (chaîne1));
9.  // caractère n° 2
10. console.log("chaîne1[2]=", chaîne1[2]);
11. // provoque une erreur
12. chaîne1[2] = "0";
```

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Temp\19-09-
    01\javascript\strings\str-01.js"
2.  typeof(chaîne1)= string
3.  chaîne1[2]= c
4.  c:\Temp\19-09-01\javascript\strings\str-01.js:1
5.  TypeError: Cannot assign to read only property '2' of string 'abcd '
6.  at Object.<anonymous> (c:\Temp\19-09-01\javascript\strings\str-01.js:12:12)
7.  at Generator.next (<anonymous>)
```

6.2 script [str-02]

Ce script montre qu'on peut construire une chaîne de caractères de deux façons.

```
1.  'use strict';
2.
3.  // les chaînes de caractères peuvent être de deux types
4.
5.  // une chaîne littérale
6.  const chaîne1 = "abcd ";
7.  // type
8.  console.log("typeof(chaîne1)=", typeof (chaîne1));
9.  // instance de String
10. const chaîne2 = new String("xyzt");
11. // type
12. console.log("typeof(chaîne2)=", typeof (chaîne2));
13. // autre écriture (sans new) - type [string] et non [object]
14. const chaîne3 = String("12 34");
15. // type
```

```
16. console.log("typeof(chaîne3)=", typeof (chaîne3));
17. // le type [string] et le type [object] offrent les mêmes méthodes, celles de la classe String
18. console.log("chaîne1.length=", chaîne1.length);
19. console.log("chaîne2.length=", chaîne2.length);
```

Commentaires

- ligne 6 : la méthode usuelle de définition d'une chaîne. **[chaîne1]** sera de type **[string]** ;
- ligne 10 : on peut construire une chaîne à l'aide du constructeur de la classe **[String]**. **[chaîne2]** sera de type **[object]** ;

Exécution

```
1. [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-
   01\javascript\strings\str-02.js"
2. typeof(chaîne1)= string
3. typeof(chaîne2)= object
4. typeof(chaîne3)= string
5. chaîne1.length= 5
6. chaîne2.length= 4
```

Le type **[string]** bénéficie des méthodes de la classe **[String]**.

6.3 script [str-03]

Ce script montre une chaîne particulière avec interpolation de variables.

```
1. 'use strict';
2.
3. // chaine
4. const chaîne = "Introduction à Javascript par l'exemple";
5. // chaîne avec interpolation de variables
6. const str = `[${chaîne}].substr(3, 2)=` + chaîne.substr(3, 2)
7. console.log(str);
```

Commentaires

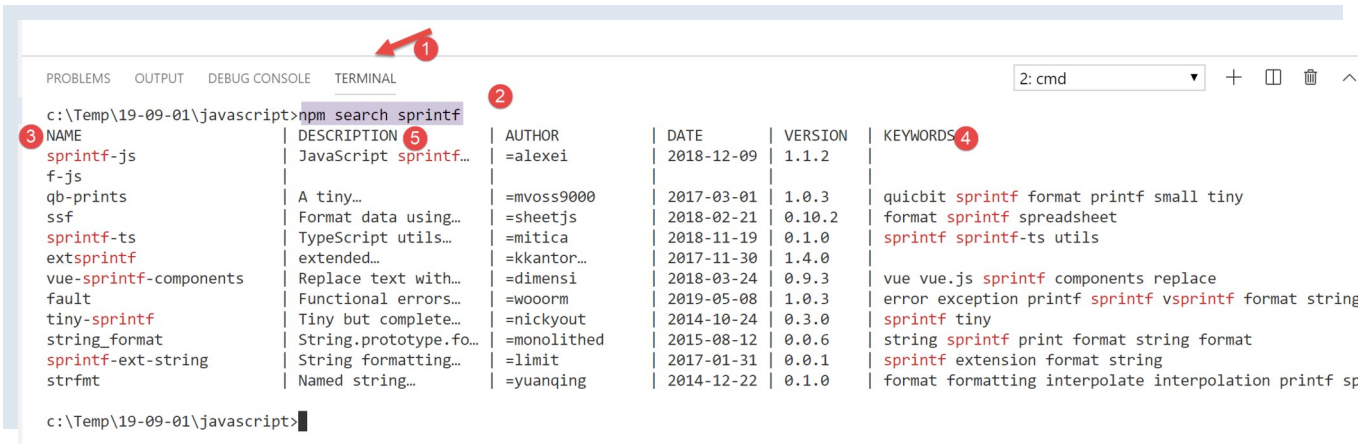
- ligne 6 : il est possible d'avoir des chaînes de caractères contenant des expression ${variable} qui sont remplacées par la valeur de la variable. On est là dans la même logique que les variables $ dans les chaînes de caractères PHP. On notera la notation d'une telle chaîne : elle est entourée de « backstick » ou apostrophe inverse (AltGr-7 sur un clavier français) ;

Exécution

```
1. [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Temp\19-09-
   01\javascript\strings\tempCodeRunnerFile.js"
2. [Introduction à Javascript par l'exemple].substr(3, 2)=ro
```

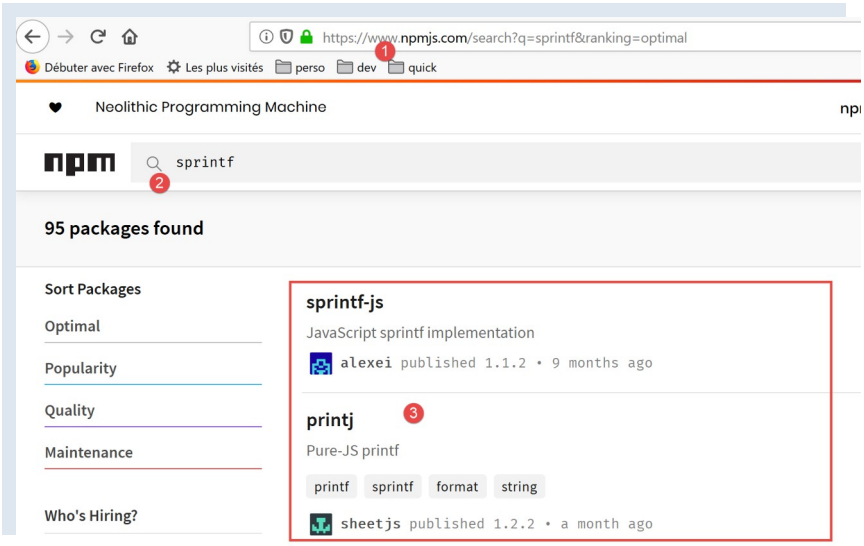
6.4 script [str-04]

La chaîne avec interpolation de variables reste insuffisante. Il n'est en effet pas possible de mettre une expression à la place de la variable dans l'expression ${variable}. Pour ceux qui ont programmé en C, il n'y a rien de tel que les fonctions **[printf, sprintf]** pour écrire ou construire des chaînes formatées. Des centaines de développeurs ont développé des milliers de packages Javascript formant un écosystème immense. Lorsqu'on a un besoin non satisfait nativement par Javascript, il est temps de chercher un package qui le satisfasse. Pour cela nous utilisons le gestionnaire de packages **[npm]**. Celui-ci dispose d'une option **[search]** qui permet de chercher une chaîne de caractères dans la description des packages. **[npm]** renvoie la liste des packages satisfaisant à la recherche. Nous allons donc chercher la chaîne **[sprintf]** dans la description des packages :



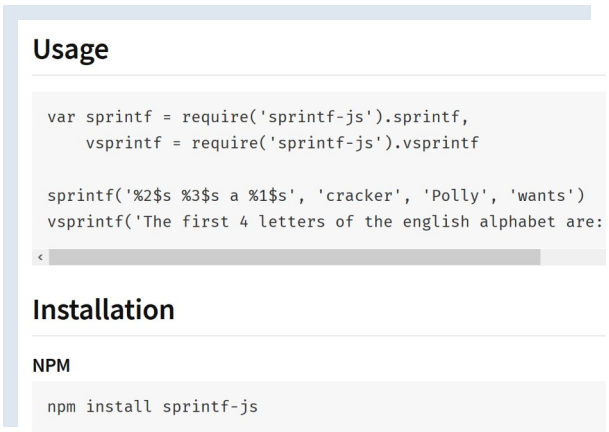
- dans la colonne **[4]**, les mots clés des packages de la colonne **[3]** ;
- dans la colonne **[5]**, la description des packages de la colonne **[3]** ;

L'étape suivante est d'aller sur le site de l'outil **[npm]**, **[https://www.npmjs.com/]** et de lire la description des packages :

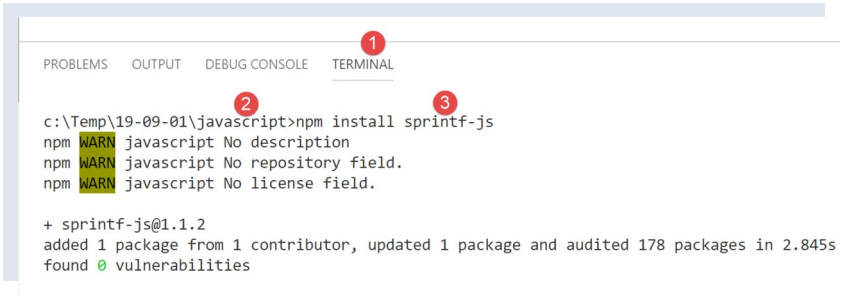


En **[3]**, on examine la liste des packages et on en choisit un.

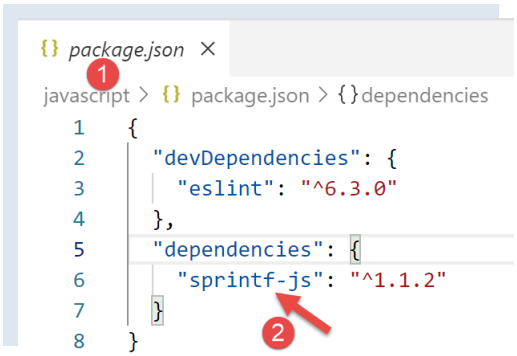
Dans la description du package, on trouve les informations pour l'installer et l'utiliser :



Nous installons le package **[sprintf-js]** dans un terminal de **[VSCode]** :



Cette installation va modifier le fichier **[package.json]** situé à la racine du dossier **[javascript] [2]** :



On voit ci-dessus que le package a été installé dans les **[dependencies]**, ç-à-d dans les packages nécessaires à l'exécution du projet. On rappelle qu'on met dans **[devDependencies]** les packages nécessaires uniquement pendant le développement du projet. Ils ne sont pas utilisés pendant l'exécution. Cette différence est importante lorsqu'il faut créer la version finale du projet pour sa mise en production. Il existe des outils pour :

- rassembler tous les fichiers jS nécessaires à l'exécution dans un seul fichier. Les packages des **[devDependencies]** ne sont donc pas inclus dans ce fichier final ;
- minifier celui-ci, ç-à-d rendre sa taille la plus petite possible. Pour cela par exemple, tous les commentaires sont éliminés ;
- « obscurcir » le code pour le rendre difficilement compréhensible. Par exemple, les variables taux, salaire, impôt vont être remplacées par des variables a, b, c ;
- faire d'autres optimisations ;

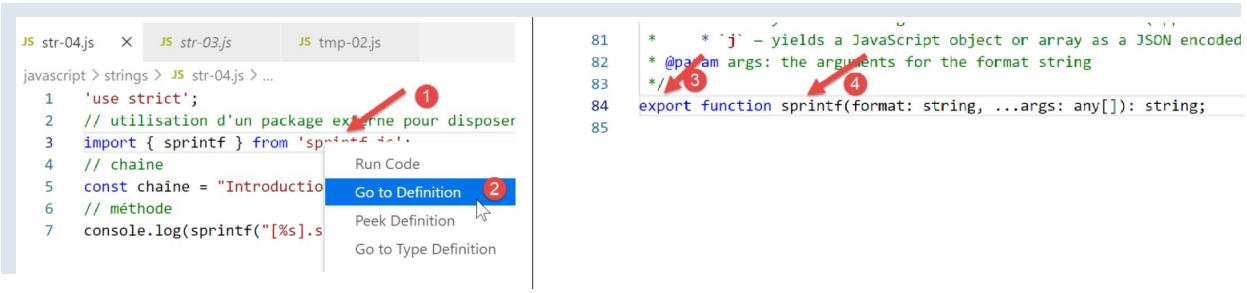
Cette optimisation du fichier final d'un projet jS est utilisée en programmation web. Une application web peut dépendre de très nombreux fichiers Javascript. Le chargement de ceux-ci par un navigateur peut ralentir l'affichage de la première

page de l'application. L'optimisation précédente vise à améliorer ce temps de chargement. Si le temps de chargement est jugé trop long par les utilisateurs, l'application ne sera pas utilisée.

Maintenant que nous disposons du package **[sprintf-js]**, il nous faut l'utiliser. C'est le script **[str-04]** :

```
1.  'use strict';
2.  // utilisation d'un package externe pour disposer de la fonction sprintf
3.  import { sprintf } from 'sprintf-js';
4.  // chaine
5.  const chaîne = "Introduction à Javascript par l'exemple";
6.  // méthode
7.  console.log(sprintf("[%s].substr(3,2)=[%s]", chaîne, chaîne.substr(3, 2)));
```

Avec ECMAScript 6, on utilise le mot clé **[import]** pour **importer** un objet exporté par un package. Pour savoir ce qu'exporte le package, on peut aller voir son code :



- en **[1]**, clic droit sur le package importé ;
- en **[2]**, on veut en voir la définition ;
- en **[3-4]**, on voit que le package exporte une fonction appelée **[sprintf]** ;

La fonction **[sprintf]** du package **[sprintf-js]** est importée avec l'instruction :

```
import { sprintf } from 'sprintf-js';
```

Le code complet :

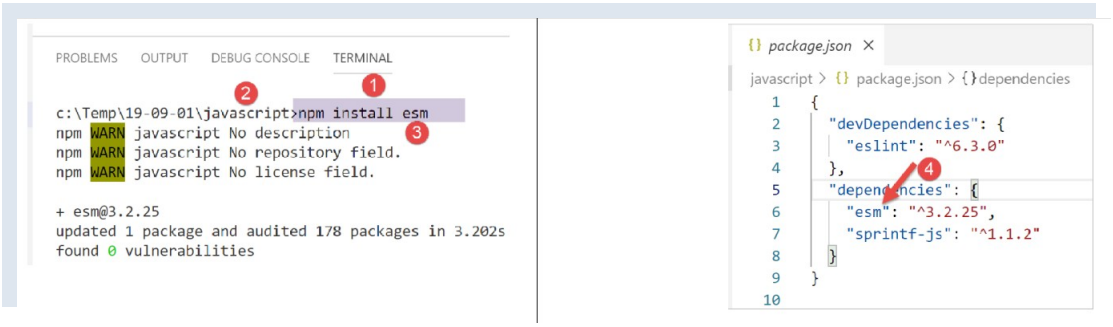
```
1.  'use strict';
2.  // utilisation d'un package externe pour disposer de la fonction sprintf
3.  import { sprintf } from 'sprintf-js';
4.  // chaine
5.  const chaîne = "Introduction à Javascript par l'exemple";
6.  // méthode
7.  console.log(sprintf("[%s].substr(3,2)=[%s]", chaîne, chaîne.substr(3, 2)));
```

produit les résultats suivants :

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-
    01\javascript\strings\str-04.js"
2.  c:\Temp\19-09-01\javascript\strings\str-04.js:3
3.  import { sprintf } from 'sprintf-js';
4.  ^
5.
6.  SyntaxError: Unexpected token {
7.  at new Script (vm.js:79:7)
8.  at createScript (vm.js:251:10)
9.  at Object.runInThisContext (vm.js:303:10)
10. at Module._compile (internal/modules/cjs/loader.js:657:28)
```

Ligne 3, l'instruction **[import]** n'est pas comprise. Cela vient du fait que la version 10.15.1 de **[node.js]** utilisée dans ce cours (sept 2019) n'observe pas encore la norme ECMAScript pour l'importation de packages appelés modules. En 2019, **[node.js]** observe une norme de modules appelée CommonJS. L'intégration des modules ECMAScript par **[node.js]** est prévue en 2020. Là encore des développeurs se sont mis à la tâche et ont produit des packages permettant l'utilisation de modules ES6 avec **[node.js]** dès maintenant (2019).

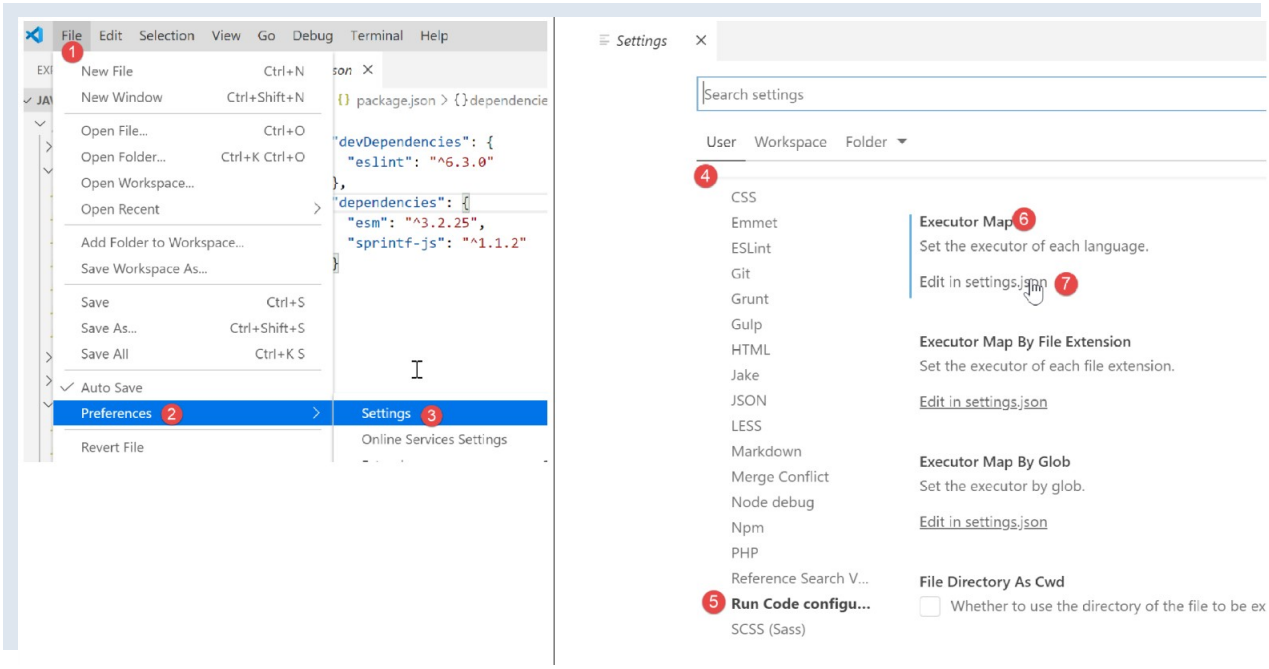
Nous allons utiliser un package appelé **[esm]** (**E**CMA**S**cript **M**odules). Nous l'installons dans un terminal du projet **[javascript]** :

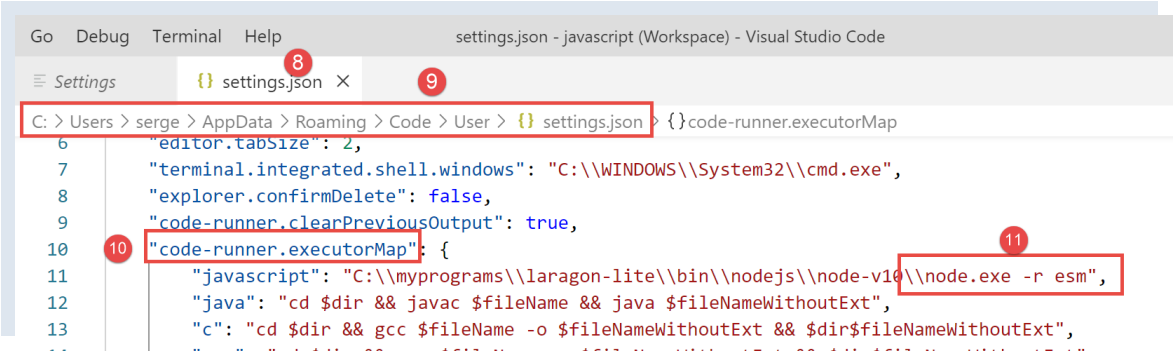


En **[4]**, on constate que l'installation du package **[esm]** **[1-3]** a modifié le fichier **[javascript/package.json]**.

Nous n'avons pas fini. Pour que le module **[esm]** soit utilisé par **[node.js]**, il faut lancer celui-ci avec l'argument **[-r esm]**.

Nous modifions dons la configuration de l'extension **[Code Runner]** de **[VSCode]** :

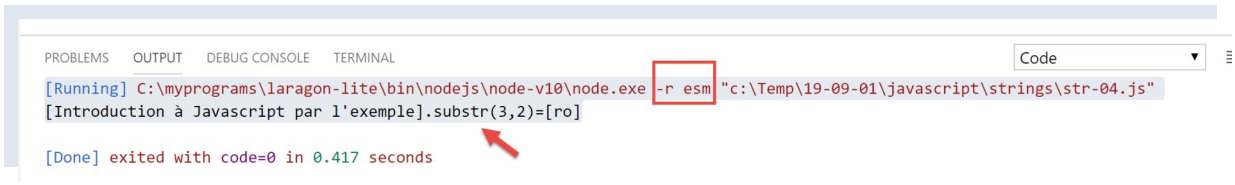




En **[11]**, on ajoute l'argument **[-r esm]** et on sauvegarde (Ctrl-S) la configuration.

Maintenant, nous pouvons exécuter le script **[str-04]** :

```
1.  'use strict';
2.  // utilisation d'un package externe pour disposer de la fonction sprintf
3.  import { sprintf } from 'sprintf-js';
4.  // chaine
5.  const chaîne = "Introduction à Javascript par l'exemple";
6.  // méthode substr
7.  console.log(sprintf("[%s].substr(3,2)=[%s]", chaîne, chaîne.substr(3, 2)));
```



6.5 script [str-05]

Voici ce que dit la documentation sur la fonction **[sprintf]** :

The placeholders in the format string are marked by `%` and are followed by one or more of these elements, in this order:

- *An optional number followed by a `$` sign that selects which argument index to use for the value. If not specified, arguments will be placed in the same order as the placeholders in the input string.*
- *An optional `+` sign that forces to preceed the result with a plus or minus sign on numeric values. By default, only the `-` sign is used on negative numbers.*
- *An optional padding specifier that says what character to use for padding (if specified). Possible values are `0` or any other character precedeed by a `'` (single quote). The default is to pad with spaces.*
- *An optional `-` sign, that causes `sprintf` to left-align the result of this placeholder. The default is to right-align the result.*
- *An optional number, that says how many characters the result should have. If the value to be returned is shorter than this number, the result will be padded. When used with the `j` (JSON) type specifier, the padding length specifies the tab size used for indentation.*
- *An optional precision modifier, consisting of a `.` (dot) followed by a number, that says how many digits should be displayed for floating point numbers. When used with the `g` type specifier, it specifies the number of significant digits. When used on a string, it causes the result to be truncated.*
- *A type specifier that can be any of:*
- `%` — *yields a literal `%` character*
- `b` — *yields an integer as a binary number*
- `c` — *yields an integer as the character with that ASCII value*
- `d` or `i` — *yields an integer as a signed decimal number*
- `e` — *yields a float using scientific notation*
- `u` — *yields an integer as an unsigned decimal number*
- `f` — *yields a float as is; see notes on precision above*
- `g` — *yields a float as is; see notes on precision above*
- `o` — *yields an integer as an octal number*
- `s` — *yields a string as is*
- `t` — *yields* `true` *or* `false`
- `T` — *yields the type of the argument*[1]
- `v` — *yields the primitive value of the specified argument*
- `x` — *yields an integer as a hexadecimal number (lower-case)*
- `X` — *yields an integer as a hexadecimal number (upper-case)*
- `j` — *yields a JavaScript object or array as a JSON encoded string*

Le script **[script-05]** met en œuvre quelques-uns de ces formats :

```
1.  'use strict';
2.  // utilisation d'un package externe pour disposer de la fonction sprintf
3.  import { sprintf } from 'sprintf-js';
4.  // chaine
5.  const chaîne = "Javascript";
```

```
// chaînes de caractères
console.log(sprintf("[%s, %%s]=>[%s]", chaîne, chaîne));
console.log(sprintf("[%s, %%20s]=>[%20s]", chaîne, chaîne));
console.log(sprintf("[%s, %%-20s]=>[%-20s]", chaîne, chaîne));
// entiers
console.log(sprintf("[%d, %%d]=>[%d]", 10, 10));
console.log(sprintf("[%d, %%4d]=>[%4d]", 10, 10));
console.log(sprintf("[%d, %%-4d]=>[%-4d]", 10, 10));
console.log(sprintf("[%d, %%04d]=>[%04d]", 10, 10));
// réels
console.log(sprintf("[%f, %%f]=>[%f]", -10.5, -10.5));
console.log(sprintf("[%f, %%10.2f]=>[%10.2f]", -10.5, -10.5));
console.log(sprintf("[%f, %%-10.2f]=>[%-10.2f]", -10.5, -10.5));
console.log(sprintf("[%f, %%010.3f]=>[%010.3f]", -10.5, -10.5));
// json
console.log(sprintf("personne (%%j)=%j", { nom: "mathieu", âge: 34 }));
// type
console.log(sprintf("type personne (%%T)=%T", { nom: "mathieu", âge: 34 }));
// booléen
console.log(sprintf("booléen (%%t)=%t", 4 === 4));
```

Exécution

```
[Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
\Data\st-2019\dev\es6\javascript\strings\str-05.js"
[Javascript, %s]=>[Javascript]
[Javascript, %20s]=>[ Javascript]
[Javascript, %-20s]=>[Javascript ]
[10, %d]=>[10]
[10, %4d]=>[ 10]
[10, %-4d]=>[10 ]
[10, %04d]=>[0010]
[-10.5, %f]=>[-10.5]
[-10.5, %10.2f]=>[ -10.50]
[-10.5, %-10.2f]=>[-10.50 ]
[-10.5, %010.3f]=>[-00010.500]
personne (%j)={"nom":"mathieu","âge":34}
type personne (%T)=object
booléen (%t)=true
```

6.6 script [str-06]

Le script **[str-06]** présente quelques méthodes de la classe **[String]** utilisables également sur le type **[string]** :

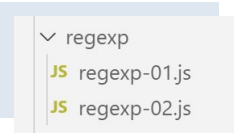
```
'use strict';
// utilisation d'un package externe pour disposer de la fonction sprintf
import { sprintf } from 'sprintf-js';
// chaine
const chaîne = "  Introduction à Javascript ";
// quelques méthodes
// substr(10,2) : 2 caractères à partir du n° 10
console.log(sprintf("[%s].substr(10,2)=[%s]", chaîne, chaîne.substr(10, 2)));
// trim : élimination des blancs de début et fin de chaîne (blanc=\b \t \r \n \f)
console.log(sprintf("[%s].trim()=[%s]", chaîne, chaîne.trim()));
// toLowerCase : transformation en minuscules
console.log(sprintf("[%s].toLowerCase=[%s]", chaîne, chaîne.toLowerCase()));
// toUpperCase : transformation en majuscules
console.log(sprintf("[%s].toUpperCase=[%s]", chaîne, chaîne.toUpperCase()));
// indexOf : position d'une chaîne  cherchée dans la chaîne, -1 si la sous-chaîne n'existe pas
console.log(sprintf("[%s].indexOf('Java')=[%s]", chaîne, chaîne.indexOf('Java')));
console.log(sprintf("[%s].trim().indexOf('abcd')=[%s]", chaîne, chaîne.indexOf('abcd')));
// includes : vrai si la chaîne cherchée est dans la chaîne
console.log(sprintf("[%s].includes('Java')=[%s]", chaîne, chaîne.includes('Java')));
// length : longueur de la chaîne - n'est pas une méthode mais une propriété
console.log(sprintf("[%s].length=[%s]", chaîne, chaîne.length));
// slice (7,10) : chaînes des caractères n° 7 à 9
console.log(sprintf("[%s].slice(7,10)=[%s]", chaîne, chaîne.slice(7, 10)));
// match : cherche une expression dans la chaîne - cette expression peut être une expression
régulière
// /intro/i : expression régulière désignant la chaîne [intro] en majuscules ou minuscules
// rend la chaîne trouvée
console.log(sprintf("[%s].match(/intro/i)=[%s]", chaîne, chaîne.match(/intro/i)));
// replace : remplace chaine1 par chaine2 dans chaîne
```

```
// remplace la 1ère occurrence de i par x
console.log(sprintf("[%s].replace('i','x')=[%s]", chaîne, chaîne.replace('i', 'x')));
// remplace toutes les occurrences de i par x
// /i/g est une expression régulière désignant toutes (g) les occurrences de i
console.log(sprintf("[%s].replace(/i/g,'x')=[%s]", chaîne, chaîne.replace(/i/g, 'x')));
// split : divise la chaîne en mots séparés par le paramètre de split
// rend le tableau de ces mots
// /\s*/ : mots séparés par 0 ou plusieurs espaces
console.log(sprintf("[%s].split(/\\s*/)=[%s]", chaîne, chaîne.split(/\s*/)));
// /\s+/ : mots séparés par un ou plusieurs espaces
console.log(sprintf("[%s].split(/\\s+/)=[%s]", chaîne, chaîne.split(/\s+/)));
```

Exécution

```
[Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
\Data\st-2019\dev\es6\javascript\strings\str-06.js"
[ Introduction à Javascript ].substr(10,2)=[ti]
[ Introduction à Javascript ].trim()=[Introduction à Javascript]
[ Introduction à Javascript ].toLowerCase=[ introduction à javascript ]
[ Introduction à Javascript ].toUpperCase=[ INTRODUCTION À JAVASCRIPT ]
[ Introduction à Javascript ].indexOf('Java')=[17]
[ Introduction à Javascript ].trim().indexOf('abcd')=[-1]
[ Introduction à Javascript ].includes('Java')=[true]
[ Introduction à Javascript ].length=[28]
[ Introduction à Javascript ].slice(7,10)=[duc]
[ Introduction à Javascript ].match(/intro/i)=[Intro]
[ Introduction à Javascript ].replace('i','x')=[ Introductxon à Javascript ]
[ Introduction à Javascript ].replace(/i/g,'x')=[ Introductxon à Javascrxpt ]
[ Introduction à Javascript ].split(/\s*/)=[,I,n,t,r,o,d,u,c,t,i,o,n,à,J,a,v,a,s,c,r,i,p,t,]
[ Introduction à Javascript ].split(/\s+/)=[,Introduction,à,Javascript,]
```

7 Expressions régulières



7.1 script [regex-01]

Dans le cours PHP, nous avions utilisé le code suivant pour illustrer les expressions régulières de PHP 7 :

```
1.  <?php
2.
3.  // type strict pour les paramètres de fonctions
4.  declare (strict_types=1);
5.
6.  // expressions régulières en php
7.  // récupérer les différents champs d'une chaîne
8.  // le modèle : une suite de chiffres entourée de caractères quelconques
9.  // on ne veut récupérer que la suite de chiffres
10. $modèle = "/(\d+)/";
11. // on confronte la chaîne au modèle
12. compareModele2Chaine($modèle, "xyz1234abcd");
13. compareModele2Chaine($modèle, "12 34");
14. compareModele2Chaine($modèle, "abcd");
15.
16. // le modèle : une suite de chiffres entourée de caractères quelconques
17. // on veut la suite de chiffres ainsi que les champs qui suivent et précèdent
18. $modèle = "/^(.*?)(\d+)(.*?)$/";
19. // on confronte la chaîne au modèle
20. compareModele2Chaine($modèle, "xyz1234abcd");
21. compareModele2Chaine($modèle, "12 34");
22. compareModele2Chaine($modèle, "abcd");
23.
24. // le modèle - une date au format jj/mm/aa
25. $modèle = "/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/";
26. compareModele2Chaine($modèle, "10/05/97");
27. compareModele2Chaine($modèle, "  04/04/01  ");
28. compareModele2Chaine($modèle, "5/1/01");
29.
30. // le modèle - un nombre décimal
31. $modèle = "/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/";
32. compareModele2Chaine($modèle, "187.8");
33. compareModele2Chaine($modèle, "-0.6");
34. compareModele2Chaine($modèle, "4");
35. compareModele2Chaine($modèle, ".6");
36. compareModele2Chaine($modèle, "4.");
37. compareModele2Chaine($modèle, " + 4");
38.
39. // fin
40. exit;
41.
42. // ----------------------------------------------------------------------------
43. function compareModele2Chaine(string $modèle, string $chaîne): void {
44.   // compare la chaîne $chaîne au modèle $modèle
45.   // on confronte la chaîne au modèle
46.   $champs = [];
47.   $correspond = preg_match($modèle, $chaîne, $champs);
48.   // affichage résultats
49.   print "\nRésultats($modèle,$chaîne)\n";
50.   if ($correspond) {
51.     for ($i = 0; $i < count($champs); $i++) {
52.       print "champs[$i]=$champs[$i]\n";
53.     }
54.   } else {
55.     print "La chaîne [$chaîne] ne correspond pas au modèle [$modèle]\n";
56.   }
```

```
57. }
```

Nous transposons ce code en Javascript de la façon suivante :

```
1.  'use strict';
2.
3.  /// expressions régulières en javascript
4.  // récupérer les différents champs d'une chaîne
5.  // le modèle : une suite de chiffres entourée de caractères quelconques
6.  // on ne veut récupérer que la suite de chiffres
7.  let modèle = /(\d+)/;
8.  // on confronte la chaîne au modèle
9.  compareModèleToChaîne(modèle, "xyz1234abcd");
10. compareModèleToChaîne(modèle, "12 34");
11. compareModèleToChaîne(modèle, "abcd");
12.
13. // le modèle : une suite de chiffres entourée de caractères quelconques
14. // on veut la suite de chiffres ainsi que les champs qui suivent et précèdent
15. modèle = /^(.*?)(\d+)(.*?)$/;
16. // on confronte la chaîne au modèle
17. compareModèleToChaîne(modèle, "xyz1234abcd");
18. compareModèleToChaîne(modèle, "12 34");
19. compareModèleToChaîne(modèle, "abcd");
20.
21. // le modèle - une date au format jj/mm/aa
22. modèle = /^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/;
23. compareModèleToChaîne(modèle, "10/05/97");
24. compareModèleToChaîne(modèle, "  04/04/01  ");
25. compareModèleToChaîne(modèle, "5/1/01");
26.
27. // le modèle - un nombre décimal
28. modèle = /^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/;
29. compareModèleToChaîne(modèle, "187.8");
30. compareModèleToChaîne(modèle, "-0.6");
31. compareModèleToChaîne(modèle, "4");
32. compareModèleToChaîne(modèle, ".6");
33. compareModèleToChaîne(modèle, "4.");
34. compareModèleToChaîne(modèle, " + 4");
35.
36. // ----------------------------------------------------------------------------
37. function compareModèleToChaîne(modèle, chaîne) {
38.   // compare la chaîne [chaîne] au modèle [modèle]
39.   console.log(`---------- chaîne=${chaîne}, modèle=${modèle}`)
40.   // on confronte la chaîne au modèle
41.   const result1 = modèle.exec(chaîne);
42.   console.log(`comparaison avec exec=`, result1);
43.   // une autre façon de faire
44.   const result2 = chaîne.match(modèle);
45.   console.log(`comparaison avec match=`, result2);
46. }
```

Commentaires

- les codes PHP et Javascript sont très proches l'un de l'autre ;
- ligne 7 : on notera qu'en Javascript l'expression régulière n'est pas une chaîne de caractères mais un objet. On ne met pas de guillemets ou d'apostrophes autour de l'expression ;
- lignes 41 et 44 : il y a deux méthodes pour obtenir le même résultat ;

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\regexp\regexp-01.js"
2.  type d'une expression régulière : object
3.  ---------- chaîne=xyz1234abcd, modèle=/(\d+)/
4.  comparaison avec exec= [ '1234',
5.  '1234',
6.  index: 3,
7.  input: 'xyz1234abcd',
8.  groups: undefined ]
9.  comparaison avec match= [ '1234',
10. '1234',
11. index: 3,
```

```
12. input: 'xyz1234abcd',
13. groups: undefined ]
14. ----------- chaîne=12 34, modèle=/(\d+)/
15. comparaison avec exec= [ '12', '12', index: 0, input: '12 34', groups: undefined ]
16. comparaison avec match= [ '12', '12', index: 0, input: '12 34', groups: undefined ]
17. ----------- chaîne=abcd, modèle=/(\d+)/
18. comparaison avec exec= null
19. comparaison avec match= null
20. ----------- chaîne=xyz1234abcd, modèle=/^(.*?)(\d+)(.*?)$/
21. comparaison avec exec= [ 'xyz1234abcd',
22. 'xyz',
23. '1234',
24. 'abcd',
25. index: 0,
26. input: 'xyz1234abcd',
27. groups: undefined ]
28. comparaison avec match= [ 'xyz1234abcd',
29. 'xyz',
30. '1234',
31. 'abcd',
32. index: 0,
33. input: 'xyz1234abcd',
34. groups: undefined ]
35. ----------- chaîne=12 34, modèle=/^(.*?)(\d+)(.*?)$/
36. comparaison avec exec= [ '12 34',
37. '',
38. '12',
39. ' 34',
40. index: 0,
41. input: '12 34',
42. groups: undefined ]
43. comparaison avec match= [ '12 34',
44. '',
45. '12',
46. ' 34',
47. index: 0,
48. input: '12 34',
49. groups: undefined ]
50. ----------- chaîne=abcd, modèle=/^(.*?)(\d+)(.*?)$/
51. comparaison avec exec= null
52. comparaison avec match= null
53. ----------- chaîne=10/05/97, modèle=/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/
54. comparaison avec exec= [ '10/05/97',
55. '10',
56. '05',
57. '97',
58. index: 0,
59. input: '10/05/97',
60. groups: undefined ]
61. comparaison avec match= [ '10/05/97',
62. '10',
63. '05',
64. '97',
65. index: 0,
66. input: '10/05/97',
67. groups: undefined ]
68. ----------- chaîne= 04/04/01 , modèle=/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/
69. comparaison avec exec= [ ' 04/04/01 ',
70. '04',
71. '04',
72. '01',
73. index: 0,
74. input: ' 04/04/01 ',
75. groups: undefined ]
76. comparaison avec match= [ ' 04/04/01 ',
77. '04',
78. '04',
79. '01',
80. index: 0,
81. input: ' 04/04/01 ',
82. groups: undefined ]
83. ----------- chaîne=5/1/01, modèle=/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/
84. comparaison avec exec= null
85. comparaison avec match= null
```

```
---------- chaîne=187.8, modèle=/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/
comparaison avec exec= [ '187.8',
'',
'187.8',
index: 0,
input: '187.8',
groups: undefined ]
comparaison avec match= [ '187.8',
'',
'187.8',
index: 0,
input: '187.8',
groups: undefined ]
---------- chaîne=-0.6, modèle=/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/
comparaison avec exec= [ '-0.6', '-', '0.6', index: 0, input: '-0.6', groups: undefined ]
comparaison avec match= [ '-0.6', '-', '0.6', index: 0, input: '-0.6', groups: undefined ]
---------- chaîne=4, modèle=/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/
comparaison avec exec= [ '4', '', '4', index: 0, input: '4', groups: undefined ]
comparaison avec match= [ '4', '', '4', index: 0, input: '4', groups: undefined ]
---------- chaîne=.6, modèle=/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/
comparaison avec exec= [ '.6', '', '.6', index: 0, input: '.6', groups: undefined ]
comparaison avec match= [ '.6', '', '.6', index: 0, input: '.6', groups: undefined ]
---------- chaîne=4., modèle=/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/
comparaison avec exec= [ '4.', '', '4.', index: 0, input: '4.', groups: undefined ]
comparaison avec match= [ '4.', '', '4.', index: 0, input: '4.', groups: undefined ]
---------- chaîne= + 4, modèle=/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/
comparaison avec exec= [ ' + 4', '+', '4', index: 0, input: ' + 4', groups: undefined ]
comparaison avec match= [ ' + 4', '+', '4', index: 0, input: ' + 4', groups: undefined ]
```

Les méthodes **[regexp.exec]** et **[string.match]** donnent les mêmes résultats :

- **[null]** s'il n'y a pas de correspondances entre la chaîne et son modèle ;
- un tableau t, s'il y a correspondance avec :
  - t**[0]** : la chaîne correspondant au modèle ;
  - t**[1]** : la chaîne correspondant à la 1ère parenthèse du modèle ;
  - t**[2]** : la chaîne correspondant à la 2ième parenthèse du modèle ;
  - …
  - t**[input]** : la chaîne entière dans laquelle on a cherché le modèle ;

7.2 script [regexp-02]

Parfois on ne souhaite pas récupérer des éléments de la chaîne testée mais seulement savoir si elle correspond au modèle :

```
1.  'use strict';
2.
3.  /// expressions régulières en javascript
4.  // récupérer les différents champs d'une chaîne
5.  // le modèle : une suite de chiffres entourée de caractères quelconques
6.  // on ne veut récupérer que la suite de chiffres
7.  let modèle = /\d+/;
8.  console.log("type d'une expression régulière : ", typeof (modèle));
9.  // on confronte la chaîne au modèle
10. compareModèleToChaîne(modèle, "xyz1234abcd");
11. compareModèleToChaîne(modèle, "12 34");
12. compareModèleToChaîne(modèle, "abcd");
13.
14. // le modèle : une suite de chiffres entourée de caractères quelconques
15. // on veut la suite de chiffres ainsi que les champs qui suivent et précèdent
16. modèle = /^.*?\d+.*?$/;
17. // on confronte la chaîne au modèle
18. compareModèleToChaîne(modèle, "xyz1234abcd");
19. compareModèleToChaîne(modèle, "12 34");
20. compareModèleToChaîne(modèle, "abcd");
21.
22. // le modèle - une date au format jj/mm/aa
23. modèle = /^\s*\d\d\/\d\d\/\d\d\s*$/;
24. compareModèleToChaîne(modèle, "10/05/97");
25. compareModèleToChaîne(modèle, "  04/04/01  ");
26. compareModèleToChaîne(modèle, "5/1/01");
27.
```

```
28. // le modèle - un nombre décimal
29. modèle = /^\s*[+|-]?\s*\d+\.\d*|\.\d+|\d+\s*$/;
30. compareModèleToChaîne(modèle, "187.8");
31. compareModèleToChaîne(modèle, "-0.6");
32. compareModèleToChaîne(modèle, "4");
33. compareModèleToChaîne(modèle, ".6");
34. compareModèleToChaîne(modèle, "4.");
35. compareModèleToChaîne(modèle, " + 4");
36.
37. // ------------------------------------------------------------------------
38. function compareModèleToChaîne(modèle, chaîne) {
39.   // test
40.   const correspond = modèle.test(chaîne);
41.   // compare la chaîne [chaîne] au modèle [modèle]
42.   console.log(`----------- chaîne=${chaîne}, modèle=${modèle}, correspond=${correspond}`);
43. }
```

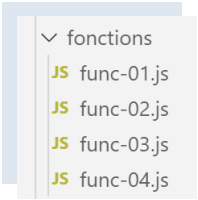
Commentaires

- **[regexp-02]** reprend le code de **[regexp-01]** avec les différences suivantes :
  - on ne souhaite pas récupérer des éléments de la chaîne testée. Aussi a-t-on enlevé les parenthèses dans les expression régulières utilisées ;
  - ligne 40 : on utilise la méthode **[Regexp.test]** pour savoir si une chaîne de caractères vérifie une expression régulière ;

Les résultats de l'exécution sont les suivants :

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\cours\regexp\regexp-02.js"
2.  type d'une expression régulière :  object
3.  ----------- chaîne=xyz1234abcd, modèle=/\d+/, correspond=true
4.  ----------- chaîne=12 34, modèle=/\d+/, correspond=true
5.  ----------- chaîne=abcd, modèle=/\d+/, correspond=false
6.  ----------- chaîne=xyz1234abcd, modèle=/^.*?\d+.*?$/, correspond=true
7.  ----------- chaîne=12 34, modèle=/^.*?\d+.*?$/, correspond=true
8.  ----------- chaîne=abcd, modèle=/^.*?\d+.*?$/, correspond=false
9.  ----------- chaîne=10/05/97, modèle=/^\s*\d\d\/\d\d\/\d\d\s*$/, correspond=true
10. ----------- chaîne=  04/04/01  , modèle=/^\s*\d\d\/\d\d\/\d\d\s*$/, correspond=true
11. ----------- chaîne=5/1/01, modèle=/^\s*\d\d\/\d\d\/\d\d\s*$/, correspond=false
12. ----------- chaîne=187.8, modèle=/^\s*[+|-]?\s*\d+\.\d*|\.\d+|\d+\s*$/, correspond=true
13. ----------- chaîne=-0.6, modèle=/^\s*[+|-]?\s*\d+\.\d*|\.\d+|\d+\s*$/, correspond=true
14. ----------- chaîne=4, modèle=/^\s*[+|-]?\s*\d+\.\d*|\.\d+|\d+\s*$/, correspond=true
15. ----------- chaîne=.6, modèle=/^\s*[+|-]?\s*\d+\.\d*|\.\d+|\d+\s*$/, correspond=true
16. ----------- chaîne=4., modèle=/^\s*[+|-]?\s*\d+\.\d*|\.\d+|\d+\s*$/, correspond=true
17. ----------- chaîne= + 4, modèle=/^\s*[+|-]?\s*\d+\.\d*|\.\d+|\d+\s*$/, correspond=true
18.
19. [Done] exited with code=0 in 0.269 seconds
```

8 Les fonctions



8.1 script [func-01]

Le script s'intéresse au mode de passage des paramètres d'une fonction :

- passage par valeur pour nombres, chaînes et booléens ;
- passage par référence pour les tableaux, objets littéraux et fonctions ;

```
1.  'use strict';
2.  // mode de passage des paramètres d'une fonction
3.  // ----------------------nombre - passage par valeur
4.  function doSomethingWithNumber(param) {
5.    param++;
6.    console.log("[param inside function]=", param, "[type]=", typeof (param), "[passage par
    référence]=", param === count);
7.  }
8.  // code d'appel
9.  let count = 10;
10. doSomethingWithNumber(count);
11. console.log("[count outside function]=", count);
12.
13. // --------------------- chaîne - passage par valeur
14. function doSomethingWithString(param) {
15.   param += " xyz"
16.   console.log("[param inside function]=", param, "[type]=", typeof (param), "[passage par
    référence]=", param === text);
17. }
18. // code d'appel
19. let text = "abcd";
20. doSomethingWithString(text);
21. console.log("[text outside function]=", text);
22.
23. // --------------------- booléen - passage par valeur
24. function doSomethingWithBoolean(param) {
25.   param = !param;
26.   console.log("[param inside function]=", param, "[type]=", typeof (param), "[passage par
    référence]=", param === bool);
27. }
28. // code d'appel
29. let bool = true;
30. doSomethingWithBoolean(bool);
31. console.log("bool [outside function]=", bool);
32.
33. // --------------------- tableau - passage par référence
34. function doSomethingWithArray(param) {
35.   param.push(1000);
36.   console.log("[param inside function]=", param, "[type]=", typeof (param), "[passage par
    référence]=", param === tab);
37. }
38. // code d'appel
39. const tab = [10, 20, 30];
40. doSomethingWithArray(tab);
41. console.log("[tab outside function]=", tab);
42.
43. // --------------------- objet - passage par référence
44. function doSomethingWithObject(param) {
45.   param.unePropriétéNouvelle = "xyz";
46.   console.log("[param inside function]=", param, "[type]=", typeof (param), "[passage par
    référence]=", param === obj);
```

```
47. }
48. // code d'appel
49. const obj = [10, 20, 30];
50. doSomethingWithObject(obj);
51. console.log("[obj outside function]=", obj);
52.
53. // --------------------- fonction - passage par référence
54. function doSomethingWithFunction(param) {
55.   // une chose plutôt bizarre qui marche pourtant
56.   param.unePropriétéNouvelle = "xyz";
57.   console.log("[param inside function]=", param, "[type]=", typeof (param), "[passage par
    référence]=", param === f);
58. }
59. // code d'appel
60. const f = x => x + 4;
61. doSomethingWithFunction(f);
62. console.log("[f outside function]=", f, f.unePropriétéNouvelle, typeof (f));
```

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\fonctions\func-01.js"
2.  [param inside function]= 11 [type]= number [passage par référence]= false
3.  [count outside function]= 10
4.  [param inside function]= abcd xyz [type]= string [passage par référence]= false
5.  [text outside function]= abcd
6.  [param inside function]= false [type]= boolean [passage par référence]= false
7.  bool [outside function]= true
8.  [param inside function]= [ 10, 20, 30, 1000 ] [type]= object [passage par référence]= true
9.  [tab outside function]= [ 10, 20, 30, 1000 ]
10. [param inside function]= [ 10, 20, 30, 'unePropriétéNouvelle': 'xyz' ] [type]= object [passage
    par référence]= true
11. [obj outside function]= [ 10, 20, 30, 'unePropriétéNouvelle': 'xyz' ]
12. [param inside function]= x => x + 4 [type]= function [passage par référence]= true
13. [f outside function]= x => x + 4 xyz function
```

8.2 script [func-02]

Le script suivant montre que le type **[function]** est un type de donnée comme un autre et qu'une variable peut avoir ce type. Il montre également deux façons de définir une fonction :

- l'une avec le mot clé **[function]** ;
- l'autre avec la notation « flèche » => ;

```
1.  'use strict';
2.  // on peut affecter une fonction à une variable
3.  const variable1 = function (a, b) {
4.  return a + b;
5.  };
6.  console.log("typeof(variable1)=", typeof (variable1));
7.  // la variable peut ensuite s'utiliser comme une fonction
8.  console.log("variable1(10,12)=", variable1(10, 12));
9.  // la définition de la fonction peut se faire avec la notation =>
10. const variable2 = (a, b, c) => {
11. return a - b + c;
12. };
13. console.log("variable2(10,12,14)=", variable2(10, 12, 14));
14. // on peut ne pas mettre les accolades s'il n'y a qu'une expression dans le code de la
    fonction
15. // cette expression est alors la valeur de retour de la fonction
16. const variable3 = (a, b, c) => a + b + c;
17. console.log("variable3(10,12,14)=", variable3(10, 12, 14));
```

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\fonctions\func-02.js"
2.  typeof(variable1)= function
3.  variable1(10,12)= 22
4.  variable2(10,12,14)= 12
5.  variable3(10,12,14)= 36
```

8.3 script [func-03]

Ce script revient sur la possibilté de passer une fonction en paramètre à une autre fonction. Ce procédé est abondamment utilisé dans les frameworks Javascript.

```
1.  'use strict';
2.  // les paramètres d'une fonction peuvent être de type [fonction]
3.
4.  // fonction f1
5.  function f1(param1, param2) {
6.    return param1 + param2 + 10;
7.  }
8.  // fonction f2
9.  function f2(param1, param2) {
10.   return param1 + param2 + 20;
11. }
12. // fonction g avec une fonction f en paramètre
13. function g(param1, param2, f) {
14.   return f(param1, param2) + 100;
15. }
16. // utilisations de g
17. console.log(g(0, 10, f1));
18. console.log(g(0, 10, f2));
19. // le paramètre effectif de type fonction peut être passé en direct - forme 1
20. console.log(g(0, 10, (param1, param2) => {
21.   return param1 + param2 + 30;
22. }));
23. // le paramètre effectif de type fonction peut être passé en direct - forme 2
24. console.log(g(0, 10, function (param1, param2) {
25.   return param1 + param2 + 40;
26. }));
```

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\fonctions\func-03.js"
2.  120
3.  130
4.  140
5.  150
```

8.4 script [func-04]

Le script suivant montre qu'une fonction Javascript peut se comporter comme une classe :

```
1.  'use strict';
2.  // une fonction peut être utilisée comme un objet
3.
4.  // une coquille vide
5.  function f() {
6.
7.  }
8.  // à qui on attribue des propriétés de l'extérieur
9.  f.prop1 = "val1";
10. f.show = function () {
11.   console.log(this.prop1);
12. }
13. // utilisation de f
14. f.show();
15.
16. // une fonction g fonctionnant comme une classe
17. function g() {
18.   this.prop2 = "val2";
19.   this.show = function () {
20.     console.log(this.prop2);
21.   }
22. }
23. // instanciation de la fonction avec [new]
24. new g().show();
```

Commentaires

- lignes 5-7 : le corps de la fonction f ne définit aucune propriété ;
- lignes 9-12 : on donne de l'extérieur des propriétés à la fonction f ;
- ligne 14 : utilisation de la function (objet) f. Notez qu'on n'écrit pas `[f()]` mais simplement `[f]`. On a là la notation d'un objet ;
- lignes 17-22 : on définit une fonction `[g]` comme si c'était une classe avec propriétés et méthodes ;
- ligne 24 : la fonction `[g]` est instanciée par `[new g()]` ;

Résultats de l'exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\classes\class-00.js"
2.  val1
3.  val2
```

ES6 a introduit la notion de classe qui nous permet désormais d'éviter de passer par des fonctions pour avoir des classes.

8.5 script [func-05]

Le script [func-05] montre l'usage d'un opérateur appelé [rest operator] :

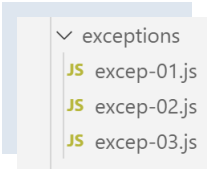
```
1.  'use strict';
2.  // rest operator
3.  function f(arg1, ...otherArgs) {
4.    // 1er argument
5.    console.log("arg1=", arg1);
6.    // les autres arguments
7.    let i = 0;
8.    otherArgs.forEach(element => {
9.      console.log("otherArguments[", i, "]=", element);
10.     i++;
11.   });
12. }
13.
14. // appel
15. f(1, "deux", "trois", { x: 2, y: 3 })
```

- ligne 3 : la notation [...otherArgs] fait qu'avec un appel de type f(param1, param2, param3) on aura ligne 3 :
  - arg1=param1
  - otherArgs=[param2, param3]. [otherArgs] est donc un tableau qui rassemble tous les paramètres effectifs passés derrière [param1] ;

Les résultats de l'application sont les suivants :

```
1.  arg1= 1
2.  otherArguments[ 0 ]= deux
3.  otherArguments[ 1 ]= trois
4.  otherArguments[ 2 ]= { x: 2, y: 3 }
```

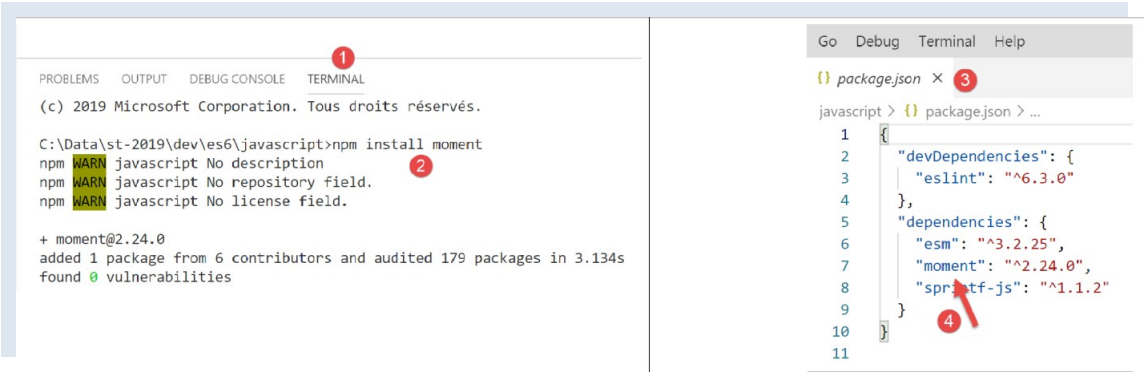
9 Les erreurs et exceptions



Javascript n'a pas un système d'exceptions très évolué. Il offre néanmoins l'instruction **[throw]** qui permet de signaler une erreur ainsi que la structure try / catch / finally qui permet d'intercepter ces erreurs.

9.1 script [excep-01]

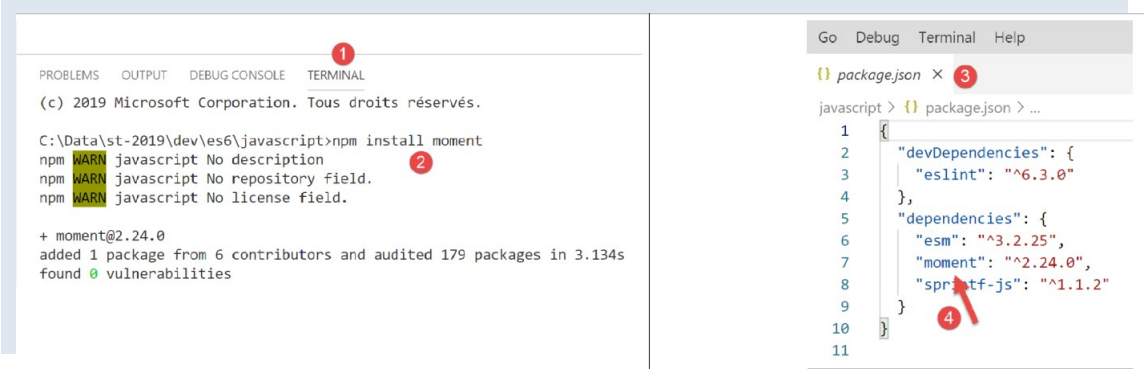
Dans le script qui suit, nous allons afficher la date du moment sous la forme **[heures:minutes:secondes:millisecondes]**. Pour ce faire, nous allons utiliser une bibliothèques jS appelée **[moment.js]**. Nous l'installons, comme d'habitude, avec l'outil **[npm]** :



Le code du script est le suivant :

```
1.  'use strict';
2.
3.  // package moment
4.  import moment from 'moment';
5.
6.  ...
```

Pour savoir comment écrire la ligne **[import]** de la ligne 4, on peut regarder la définition du module **[moment]** :



- en **[1-2]**, on va à la définition du module ;
- en **[3]**, on a un fichier Typescript, pas Javascript. A l'exécution, ce fichier Typescript est compilé en fichier Javascript avant d'être utilisé ;
- en **[4]**, on cherche les instructions **[export]** (Ctrl-F) ;
- en **[5]**, l'instruction exporte l'objet **[moment]**. Celui-ci peut être importé en ES6 de la façon suivante :

```
import moment from 'moment';
```

On peut utiliser n'importe quel nom pour importer l'objet **[moment]**, par exemple :

```
import m from 'moment';
```

Revenons au code du script :

```
1.  'use strict';
2.
3.  // package moment
4.  import moment from 'moment';
5.
6.  // principe du try / catch / finally
7.  for (let i = 0; i < 10; i++) {
8.    // date - heure du moment courant
9.    const now = Date.now();
10.   // formatage heure pour avoir les millisecondes
11.   const time = moment(now).format("HH:mm:ss:SSS");
12.   // les millisecondes
13.   const milli = Number(time.substr(time.length - 3));
14.   // affichage
15.   console.log("--------------------itération n° ", i, "à", time);
16.   try {
17.     // nbre variant selon l'heure du moment
18.     const nbre = milli % 2;
19.     if (nbre === 0) {
20.       // lancer un msg d'erreur
21.       throw "erreur";
22.     }
23.     // si on arrive ici c'est qu'il n'y a pas eu d'erreur
24.     console.log("pas d'erreur");
25.   } catch (error) {
26.     // si on arrive ici, c'est qu'il y a eu erreur
27.     console.log("erreur1=", error);
28.   } finally {
29.     // exécuté dans tous les cas erreur ou pas
30.     console.log("finally")
31.   }
32. }
```

Commentaires

- ligne 4 : import de la bibliothèque **[moment]** ;
- le principe du script est de boucler 10 fois (ligne 7). A chaque tour de boucle, on récupère l'heure du moment sous la forme **[heures:minutes:secondes:millisecondes]** (lignes 8-13) ;
- si le nombre de millisecondes est pair, on lance un message d'erreur (lignes 19-22) ;
- il s'agit ici de comprendre le fonctionnement du try / catch / finally

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\exceptions\excep-01.js"
2.  --------------------itération n° 0 à 17:57:04:440
3.  erreur1= erreur
4.  finally
5.  --------------------itération n° 1 à 17:57:04:447
6.  pas d'erreur
7.  finally
8.  --------------------itération n° 2 à 17:57:04:448
9.  erreur1= erreur
10. finally
11. --------------------itération n° 3 à 17:57:04:448
12. erreur1= erreur
```

```
13. finally
14. --------------------itération n° 4 à 17:57:04:448
15. erreur1= erreur
16. finally
17. --------------------itération n° 5 à 17:57:04:448
18. erreur1= erreur
19. finally
20. --------------------itération n° 6 à 17:57:04:448
21. erreur1= erreur
22. finally
23. --------------------itération n° 7 à 17:57:04:448
24. erreur1= erreur
25. finally
26. --------------------itération n° 8 à 17:57:04:449
27. pas d'erreur
28. finally
29. --------------------itération n° 9 à 17:57:04:449
30. pas d'erreur
31. finally
```

On voit que la clause **[finally]** est toujours exécutée qu'il y ait erreur ou pas.

9.2 script [excep-02]

Ce script que l'instruction **[throw]** peut lancer n'importe quel type de donnée et que cette donnée est récupérée intégralement par la clause **[catch]**.

```
1.  'use strict';
2.
3.  // on peut "lancer" (throw) à peu près n'importe quoi pour signaler une erreur
4.  let i = 0;
5.  console.log("--------------------essai n° ", i);
6.  // lancer une chaîne de caractères
7.  try {
8.    throw "msg d'erreur";
9.  } catch (error) {
10.   // il y a eu erreur
11.   console.log("erreur=[", error, "], type=", typeof (error));
12. }
13. // lancer un tableau
14. i++;
15. console.log("--------------------essai n° ", i);
16. try {
17.   throw [1, 2, 3]
18. } catch (error) {
19.   // il y a eu erreur
20.   console.log("erreur=[", error, "], type=", typeof (error));
21. }
22. // lancer un objet littéral
23. i++;
24. console.log("--------------------essai n° ", i);
25. try {
26.   throw { nom: "hercule", pays: "grèce antique" }
27. } catch (error) {
28.   // il y a eu erreur
29.   console.log("erreur=[", error, "], type=", typeof (error));
30. }
31. // lancer un type Error
32. i++;
33. console.log("--------------------essai n° ", i);
34. try {
35.   throw new Error("erreur de connexion au réseau");
36. } catch (error) {
37.   // il y a eu erreur
38.   console.log("erreur=[", error, "], type=", typeof (error));
39. }
40. // lancer un type Error
41. i++;
42. console.log("--------------------essai n° ", i);
43. try {
44.   throw new Error("erreur de connexion au réseau");
45. } catch (error) {
46.   // il y a eu erreur - le message est dans [error.message]
```

```
47.   console.log("erreur.message=[", error.message, "], type(error)=", typeof (error));
48. }
```

Commentaires

- lignes 35, 44 : `[Error]` est une classe Javascript dont le constructeur admet comme 1er paramètre facultatif un message d'erreur. Ce message peut être récupéré dans la propriété `[Error.message]` (ligne 47) ;
- il existe d'autres classes que `[Error]` pour signaler une erreur : [EvalError, InternalError, RangeError, ReferenceError, SyntaxError, TypeError, URIError) ;

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\exceptions\excep-02.js"
2.  --------------------essai n° 0
3.  erreur=[ msg d'erreur ], type= string
4.  --------------------essai n° 1
5.  erreur=[ [ 1, 2, 3 ] ], type= object
6.  --------------------essai n° 2
7.  erreur=[ { nom: 'hercule', pays: 'grèce antique' } ], type= object
8.  --------------------essai n° 3
9.  erreur=[ Error: erreur de connexion au réseau
10. at Object.<anonymous> (c:\Data\st-2019\dev\es6\javascript\exceptions\excep-02.js:35:9)
11. at Object.<anonymous> (c:\Temp\19-09-01\javascript\node_modules\esm\esm.js:1:251206)
12. at c:\Temp\19-09-01\javascript\node_modules\esm\esm.js:1:245054
13. at Generator.next (<anonymous>)
14. at bl (c:\Temp\19-09-01\javascript\node_modules\esm\esm.js:1:245412)
15. at kl (c:\Temp\19-09-01\javascript\node_modules\esm\esm.js:1:247659)
16. at Object.u (c:\Temp\19-09-01\javascript\node_modules\esm\esm.js:1:287740)
17. at Object.o (c:\Temp\19-09-01\javascript\node_modules\esm\esm.js:1:287137)
18. at Object.<anonymous> (c:\Temp\19-09-01\javascript\node_modules\esm\esm.js:1:284879)
19. at Object.apply (c:\Temp\19-09-01\javascript\node_modules\esm\esm.js:1:199341) ], type= object
20. --------------------essai n° 4
21. erreur.message=[ erreur de connexion au réseau ], type(error)= object
```

9.3 script [excep-03]

Ce script montre qu'on peut, dans un `[catch]`, différentier le type d'instance `[Error]` interceptée par le `[catch]` :

```
1.  'use strict';
2.
3.  // package moment
4.  import moment from 'moment';
5.
6.  // différencier l'instance d'Error reçue dans un [catch]
7.  for (let i = 0; i < 10; i++) {
8.    // date - heure du moment courant
9.    const now = Date.now();
10.   // formatage heure pour avoir les millisecondes
11.   const time = moment(now).format("HH:mm:ss:SSS");
12.   // les millisecondes
13.   const milli = Number(time.substr(time.length - 3));
14.   console.log("--------------------itération n° ", i);
15.   try {
16.     // nbre [0, 1, 2]
17.     const nbre = milli % 3;
18.     switch (nbre) {
19.       case 0:
20.         throw new ReferenceError("erreur 1");
21.       case 1:
22.         throw new RangeError("erreur 2");
23.       default:
24.         throw new EvalError("erreur 3");
25.     }
26.   } catch (error) {
27.     // il y a eu erreur
28.     if (error instanceof ReferenceError) {
29.       console.log("ReferenceError :", error.message);
30.     } else {
31.       if (error instanceof RangeError) {
32.         console.log("RangeError :", error.message);
```

```
      }
      else {
        if (error instanceof EvalError) {
          console.log("EvalError :", error.message);
        }
      }
    }
  }
}
```

Exécution

```
[Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
\Data\st-2019\dev\es6\javascript\exceptions\excep-03.js"
--------------------itération n° 0
ReferenceError : erreur 1
--------------------itération n° 1
RangeError : erreur 2
--------------------itération n° 2
RangeError : erreur 2
--------------------itération n° 3
RangeError : erreur 2
--------------------itération n° 4
RangeError : erreur 2
--------------------itération n° 5
RangeError : erreur 2
--------------------itération n° 6
EvalError : erreur 3
--------------------itération n° 7
EvalError : erreur 3
--------------------itération n° 8
EvalError : erreur 3
--------------------itération n° 9
EvalError : erreur 3
```

10 Les modules ES6

Les modules ES6 permettent de construire des applications Javascript structurées en modules indépendants et réutilisables.

10.1 scripts [import-01, export-01]

Le script **[import-01]** va utiliser le module **[export-01]**. Celui-ci est défini de la façon suivante :

```
1.  // export par défaut d'un objet non nommé
2.  export default {
3.    data: 2,
4.    do() {
5.      console.log(this.data);
6.    }
7.  };
```

Commentaires

- les lignes **[2-7]** définissent un objet non nommé avec les propriétés **[data, do]** ;
- ligne 2 : l'instruction **[export default]** exporte cet objet. Celui-ci pourra donc être importé ;

Le module **[export-01]** est utilisé par le script **[import-01]** de la façon suivante :

```
1.  'use strict';
2.  // import d'un objet exporté par défaut
3.  import export01 from './export-01';
4.  // utilisation de cet objet
5.  export01.do();
6.  // on peut importer un export par défaut sous n'importe quel nom
7.  import data from './export-01';
8.  console.log(data.data);
```

Commentaires

- les lignes 3 et 7 importent l'objet exporté par défaut du module **[export-01]** sous deux noms différents ;
- une fois un objet importé, on peut l'utiliser comme s'il avait été défini localement dans le script ;

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\modules\import-01.js"
2.  2
3.  2
```

10.2 scripts [import-02, export-02]

Ces scripts montrent un export d'un objet nommé.

Le script **[export-02]** est le suivant :

```
1.  // export par défaut d'un objet nommé
```

```
2.  const data = {
3.    data: 2,
4.    do() {
5.      console.log(this.data);
6.    }
7.  };
8.  // export
9.  export default data;
```

- ligne 9 : on exporte l'objet **[data]** ;

Que l'objet exporté soit nommé ou non ne change rien à l'opération d'import. Le script **[import-02]** est le suivant :

```
1.  'use strict';
2.  // import d'un objet exporté par défaut
3.  import module1 from './export-02';
4.  // utilisation de cet objet
5.  module1.do();
6.  // on peut importer un export par défaut sous n'importe quel nom
7.  import module2 from './export-02';
8.  console.log(module2.data);
```

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\modules\import-02.js"
2.  2
3.  2
```

10.3 scripts [import-03, export-03]

Un module peut exporter plusieurs éléments.

Le script **[export-03]** est le suivant :

```
1.  // multi-exports
2.  // export objet
3.  const data = {
4.    data: 2,
5.    do() {
6.      console.log(this.data);
7.    }
8.  };
9.  // export fonction
10. export { data };
11. function doSomething() {
12.   console.log("doSomething");
13. }
14. export { doSomething };
```

Commentaires

- lignes 10 et 14 : export de deux éléments. L'export d'un élément se fait avec la syntaxe **[export {élément}]** ;

Le script **[import-03]** utilise le module **[export-03]** de la façon suivante :

```
1.  'use strict';
2.  // import d'un module [export03]
3.  import {data, doSomething} from './export-03';
4.  // utilisation des imports
5.  data.do();
6.  doSomething();
7.  // autre écriture
8.  import * as module from './export-03';
9.  // utilisation de l'import
10. console.log(module.data);
11. module.doSomething();
```

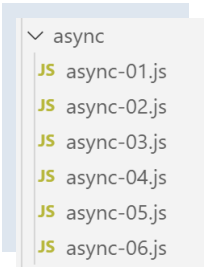
- ligne 3 : les **[imports]** se font avec les noms exacts des objets exportés ;

- ligne 8 : on peut importer tous les objets exportés dans un objet nommé (ici **`[module]`**) ;

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\modules\import-03.js"
2.  2
3.  doSomething
4.  { data: 2, do: [Function: do] }
5.  doSomething
```

11 Programmation événementielle et fonctions asynchrones



Une fonction asynchrone est une fonction dont l'exécution est lancée mais dont on n'attend pas le résultat. Lorsque l'exécution est terminée, la fonction asynchrone émet un événement et transmet son résultat via celui-ci.

Ce mode de fonctionnement est bien adapté à l'exécution au sein d'un navigateur web. En effet, une application exécutée au sein d'un navigateur est une application à événements : l'application réagit à des événements, principalement provoqués par l'utilisateur (clics, déplacements de souris, frappe de texte, …). Les applications Javascript exécutées au sein d'un navigateur sont amenées à dialoguer avec des services externes via le protocole HTTP. Les fonctions natives HTTP de Javascript sont asynchrones : elles sont lancées et l'obtention de la réponse du service externe sollicité est signalé par un événement qui s'ajoute à l'ensemble des événements gérés par l'application.

Les scripts suivants vont être exécutés par `[node.js]` et non par un navigateur. `[node.js]` a lui également un mode d'exécution par événement :

- `[node.js]`, comme un navigateur, utilise une boucle d'événements pour exécuter un script ;
- l'exécution du code principal du script est le 1er événement exécuté ;
- si ce code principal a lancé des tâches asynchrones alors l'exécution se poursuit tant que ces tâches asynchrones ne sont pas terminées. Celles-ci émettent un événement lorsqu'elles sont terminées. Ces événements sont mis en file d'attente dans la boucle d'événements ;
- le script principal doit s'abonner à ces événements s'il veut récupérer les résultats des actions asynchrones ;
- le script n'est terminé que lorsque tous les événements émis par celui-ci ont été traités ;

11.1 script [async-01]

Le script suivant montre le comportement d'un script comportant une action asynchrone.

```
1.  'use strict';
2.
3.  // imports
4.  import moment from 'moment';
5.  import { sprintf } from 'sprintf-js';
6.
7.  // début
8.  const débutScript = moment(Date.now());
9.  console.log("[début du script],", heure());
10.
11. // setTimeout arme un timer de 1000 ms (2ième paramètre) et retourne immédiatement le n° de ce
    timer
12. // lorsque le timer a épuisé les 1000 ms il émet un événement qui est mis en file d'attente du
    runtime
13. // lorsque l'événement est traité par le runtime, la fonction (1er paramètre) est exécutée
14. setTimeout(function () {
15.   // ce code sera exécuté lorsque le timer aura atteint la valeur 0
16.   console.log("[fin de l'action asynchrone setTimeout],", heure(débutScript));
17. }, 1000)
18.
19. // s'affichera avant le msg de la fonction interne au timer
20. console.log("[fin du code principal du script],", heure(débutScript));
21.
22. // utilitaire d'affichage heure et durée
23. function heure(début) {
24.   // heure du moment courant
25.   const now = moment(Date.now());
26.   // formatage heure
```

```
27.   let result = "heure=" + now.format("HH:mm:ss:SSS");
28.   // faut-il calculer une durée ?
29.   if (début) {
30.     const durée = now - début;
31.     const milliseconds = durée % 1000;
32.     const seconds = Math.floor(durée / 1000);
33.     // formatage heure + durée
34.     result = result + sprintf(", durée= %s seconde(s) et %s millisecondes", seconds,
    milliseconds);
35.   }
36.   // résultat
37.   return result;
38. }
```

- ligne 4 : on importe la bibliothèque **[moment]** pour pouvoir formater les dates (ligne 27) ;
- ligne 5 : on importe la bibliothèque **[sprintf-js]** pour pouvoir formater les durées (ligne 34) ;
- ligne 8 : on note l'heure de début du script ;
- ligne 9 : on l'affiche à l'aide de la méthode **[heure]** des lignes 20-34 ;
- lignes 14-17 : la fonction **[setTimeout]** a deux paramètres (f, durée) : **f** est une fonction qui est exécutée lorsque **[durée]** millisecondes se sont écoulées ;
- ligne 14 : à l'exécution du script, la fonction **[setTimeout]** est exécutée :
  - ligne 17 : un minuteur de 1000 ms est armé et le décompte commence jusqu'à atteindre ultérieurement 0. La fonction **[setTimeout]** est terminée dès que le minuteur est initialisé et le décompte commencé. Elle **n'attend pas** la fin de ce décompte. Elle rend un n° identifiant le minuteur utilisé et l'exécution passe à l'instruction suivante, ligne 20. Ici, le résultat de **[setTimeout]** n'est pas utilisé ;
- ligne 16 : ce message va s'afficher à la fin du délai de 1000 ms de la fonction **[setTimeout]** ;
- lignes 15-16 : la fonction f, 1er paramètre de la fonction **[setTimeout]**, va être exécutée à la fin du délai des 1000 ms. Le message de la ligne 16 va alors s'afficher ;
- ligne 20 : ce message va s'afficher avant celui de la ligne 16 ;

Fonction **heure** :

- ligne 23 : la fonction admet un paramètre facultatif **[heure]** qui est l'heure de début d'une opération dont elle doit afficher la durée ;
- lignes 25-27 : on calcule et formate l'heure du moment ;
- ligne 29 : si le paramètre **[début]** est présent alors il faut calculer une durée ;
- ligne 30 : la durée de l'opération. On obtient un nombre de millisecondes ;
- lignes 31-32 : ce nombre de millisecondes est décomposé en secondes et millisecondes ;
- ligne 34 : la durée est ajoutée à l'heure ;

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\async\async-01.js"
2.  [début du script], heure=09:26:40:238
3.  [fin du code principal du script], heure=09:26:40:246, durée= 0 seconde(s) et 11 millisecondes
4.  [fin de l'action asynchrone setTimeout], heure=09:26:41:249, durée= 1 seconde(s) et 14 millisec
    ondes
5.
6.  [Done] exited with code=0 in 1.672 seconds
```

- ligne 4, on voit que l'action asynchrone **[setTimeOut]** s'est terminée 1s environ après la fin du code principal du script ;
- ligne 6 : l'heure affichée ligne 3 est celle de la fin du code principal. Si celui-ci a lancé des tâches asynchrones, le script n'est terminé que lorsque toutes les tâches asynchrones ont été exécutées. La durée affichée ligne 6, est la durée totale d'exécution du script (code principal + tâches asynchrones) ;

La fonction **[setTimeout]** va nous permettre de simuler des tâches asynchrones dans un environnement **[node.js]**. En effet, la fonction **[setTimeout]** se comporte comme une tâche asynchrone :

- elle rend un résultat immédiatement, ici un n° de timer, par le mécanisme usuel des fonctions (return) ;
- elle peut rendre **ultérieurement** (ce n'est pas encore le cas ci-dessus) d'autres résultats **via des événements** qui sont alors traités par la boucle d'événements de **[node.js]** ;
- dans la plupart des cas qui vont suivre, ces événements seront au nombre de deux :
  - un événement qu'on pourrait appeler **[success]** qui sera émis par la tâche asynchrone qui a réussi ce qu'elle devait faire. Une donnée, le résultat de la tâche, est associée à l'événement émis ;

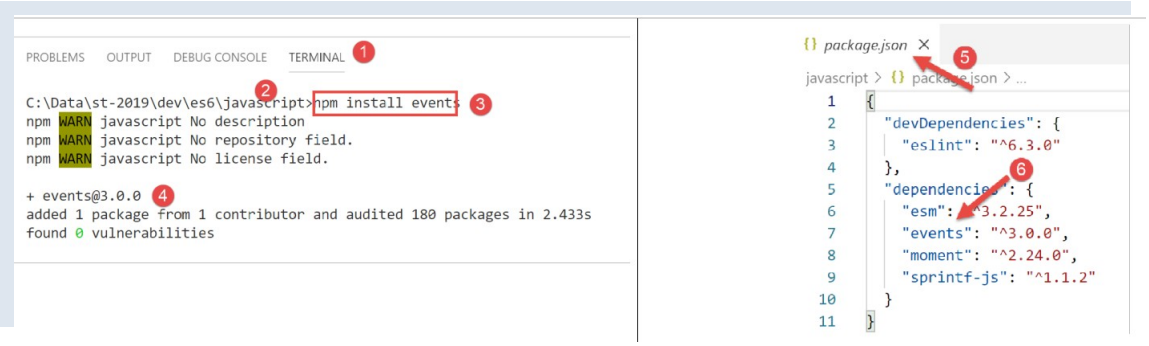
- un événement qu'on pourrait appeler **[failure]** qui sera émis par la tâche asynchrone qui a échoué à faire ce qu'elle devait faire. Une donnée, un objet décrivant l'erreur généralement, est associé à l'événement émis. Des erreurs possibles par exemple avec une tâche internet asynchrone seraient 'réseau indisponible', 'machine serveur inexistante', 'timeout dépassé', ...

- le code principal qui a lancé une tâche asynchrone peut s'abonner aux événements que cette tâche est susceptibe d'émettre. Lorsqu'un de ceux-ci est émis, le code principal en est averti et peut déclencher l'exécution d'une fonction particulière destinée à traiter l'événement. Cette fonction reçoit en paramètre, la donnée que la tâche asynchrone a associée à l'événement émis ;

11.2 script [async-02]

Dans ce script, la fonction asynchrone **[setTimeout]** va émettre des événements pour communiquer des données aux codes qui se seront abonnés à ceux-ci.

L'accès aux événements de **[node.js]** nécessite des bibliohèques supplémentaires. Nous choisissons la bibliothèque **[events]** que nous installons avec **[npm]** :



Le script **[async-02]** est le suivant :

```
1.  'use strict';
2.
3.  // les fonctions asynchrones peuvent rendre un résultat en émettant un événement
4.  // le code principal peut récupérer ces résultats en s'abonnant aux événements émis
5.
6.  // imports
7.  import moment from 'moment';
8.  import { sprintf } from 'sprintf-js';
9.  import EventEmitter from 'events';
10.
11. // début
12. const débutScript = moment(Date.now());
13. console.log("[début du script],", heure());
14. // un émetteur d'événements
15. const eventEmitter = new EventEmitter();
16.
17. // setTimeout arme un timer de 1000 ms (2ième paramètre) et retourne immédiatement le n° de ce
    timer
18. // lorsque le timer a épuisé les 1000 ms il émet un événement qui est mis en file d'attente du
    runtime
19. // lorsque l'événement est traité par le runtime, la fonction (1er paramètre) est exécutée
20. setTimeout(function () {
21.   // ce code sera exécuté lorsque le timer aura atteint la valeur 0
22.   console.log("[setTimeout, fin du timer d'1 s],", heure(débutScript));
23.   // on émet un événement pour dire qu'un résultat est disponible
24.   eventEmitter.emit("timer1Success", { success: 4 });
25.   // on émet un autre événement pour dire qu'un autre résultat est disponible
26.   eventEmitter.emit("timer1Failure", { failure: 6 });
27. }, 1000)
28.
29. // on s'abonne à l'évt [timer1Success]
30. eventEmitter.on('timer1Success', (result) => {
```

```
31.   console.log(sprintf("la fonction asynchrone du timer a rendu le résultat [%j], %s, via
    l'événement [timer1Success]", result, heure(débutScript)));
32. });
33.
34. // on s'abonne à l'évt [timer1Failure]
35. eventEmitter.on('timer1Failure', (result) => {
36.   console.log(sprintf("la fonction asynchrone du timer a rendu le résultat [%j], %s, via
    l'événement [timer1Failure]", result, heure(débutScript)));
37. });
38.
39. // s'affichera avant les msg des evts émis par la fonction associée à [timer1]
40. console.log("[fin du code principal du script],", heure(débutScript));
41.
42. // utilitaire d'affichage heure et durée
43. function heure(début) {
44.   // heure du moment courant
45.   const now = moment(Date.now());
46.   // formatage heure
47.   let result = "heure=" + now.format("HH:mm:ss:SSS");
48.   // faut-il calculer une durée ?
49.   if (début) {
50.     const durée = now - début;
51.     const milliseconds = durée % 1000;
52.     const seconds = Math.floor(durée / 1000);
53.     // formatage heure + durée
54.     result = result + sprintf(", durée= %s seconde(s) et %s millisecondes", seconds,
    milliseconds);
55.   }
56.   // résultat
57.   return result;
58. }
```

Commentaires

- ligne 9, on importe la classe `[EventEmitter]` de la bibliothèque `[events]`. C'est une nouveauté : nous n'avions jusqu'à maintenant importé que des objets littéraux et des fonctions ;
- ligne 15 : on crée un émetteur d'événements `[node.js]` en instanciant la classe `[EventEmitter]` avec le mot clé `[new]` ;
- lignes 20-27 : la fonction asynchrone `[setTimeout]`. Elle va émettre deux événements lors de son exécution :
  - ligne 24, l'événement `[timer1Success]` avec comme valeur associée l'objet {success : 4} ;
  - ligne 26, l'événement `[timer1Failure]` avec comme valeur associée l'objet {failure : 6} ;
  - une fonction asynchrone peut émettre autant d'événements qu'elle veut. On a dit précédemment que le plus souvent elle émettait l'un des deux événements `[success, failure]`, pas les deux comme on le fait ici ;
- ligne 20 : l'exécution de `[setTimeout]` est instantanée : un timer est armé et le n° de celui-ci rendu au code appelant. L'émission des événements se fera plus tard, ici 1 seconde plus tard ;
- l'émission d'événements est inutile s'il n'y a aucun code pour les exploiter lorsqu'ils surviendront. C'est pourquoi le code principal doit s'abonner aux deux événements `[timer1Success, timer1Failure]` s'il veut les gérer, notamment récupérer les données associées à ces événements ;
- lignes 30-32 : le code principal s'abonne à l'événement `[timer1Success]`. Lorsque la bouche d'événements de `[node.js]` traitera cet événement, il appellera la fonction qui est le second paramètre de la méthode `[eventEmitter.on]` en lui passant la donnée (ici appelée `[result]`) associée à l'événement `[timer1Success]` ;
- ligne 31 : la fonction de traitement de l'événement affichera le jSON de la donnée associée à l'événement ainsi que l'heure du moment ;
- lignes 35-37 : avec un code analogue, le code principal s'abonne à l'événement `[timer1Failure]` ;
- l'abonnement à un événement (1er paramètre) n'exécute pas immédiatement le code de la fonction `[callback]` (2ième paramètre). Celle-ci ne sera exécutée qu'après que l'événement ait eu lieu ;
- ligne 40 : le code principal du script est terminé mais pas le script lui-même puisque le code principal a lancé une tâche asynchrone. Le script global ne sera terminé qu'après la fin de cette tâche asynchrone ;

C'est ce que montrent les résultats obtenus :

```
1. [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
   \Data\st-2019\dev\es6\javascript\async\async-02.js"
2. [début du script], heure=09:34:58:909
3. [fin du code principal du script], heure=09:34:58:916, durée= 0 seconde(s) et 10 millisecondes
4. [setTimeout, fin du timer d'1 s], heure=09:34:59:929, durée= 1 seconde(s) et 23 millisecondes
```

```
5.  la fonction asynchrone du timer a rendu le résultat [{"success":4}], heure=09:34:59:931, durée=
    1 seconde(s) et 25 millisecondes, via l'événement [timer1Success]
6.  la fonction asynchrone du timer a rendu le résultat [{"failure":6}], heure=09:34:59:932, durée=
    1 seconde(s) et 26 millisecondes, via l'événement [timer1Failure]
7.
8.  [Done] exited with code=0 in 1.627 seconds
9.
```

- ligne 3 : fin du code principal 10 ms après le début du script ;
- ligne 4 : début de la fonction encapsulée dans le timer de 1000 ms, 1 seconde environ après le début du script ;
- ligne 5 : traitement de l'événement **['timer1Success']**, 2 ms plus tard ;
- ligne 6 : traitement de l'événement **['timer1Failure']**, 1 ms plus tard que l'événement **['timer1Success']** ;
- ligne 8 : fin du script global avec une durée totale de 1,627 seconde ;

11.3 script [async-03]

Le script suivant montre un autre aspect de la boucle événementielle de **[node.js]** :

- la boucle exécute les événements les uns après les autres, généralement dans leur ordre d'arrivée. Certains OS accordent des priorités aux événements qui sont alors traités par ordre de priorité et non par ordre d'arrivée ;
- la boucle **n'exécute qu'un événement à la fois**. Le suivant n'est traité que lorsque le traitement du précédent est terminé. Dans un système événementiel, il faut donc éviter d'écrire du code qui monopolise longtemps le processeur car alors les événements ne sont pas traités lorsqu'ils se produisent mais plus tard lorsque la boucle événementielle arrive à eux. On a alors une application peu « réactive » ;

Le script **[async-03]** montre un exemple de ce phénomène :

```
1.  'use strict';
2.
3.  // les fonctions asynchrones peuvent rendre un résultat en émettant un événement
4.  // le code principal peut récupérer ces résultats en s'abonnant aux événements émis
5.
6.  // imports
7.  import moment from 'moment';
8.  import { sprintf } from 'sprintf-js';
9.  import EventEmitter from 'events';
10.
11. // début
12. const débutScript = moment(Date.now());
13. console.log("[début du script],", heure());
14. // un émetteur d'événements
15. const eventEmitter = new EventEmitter();
16.
17. // setTimeout arme un timer de 1000 ms (2ième paramètre) et retourne immédiatement le n° de ce
    timer
18. // lorsque le timer a épuisé les 1000 ms il émet un événement qui est mis en file d'attente du
    runtime
19. // lorsque l'événement est traité par le runtime, la fonction (1er paramètre) est exécutée
20. setTimeout(function () {
21.   // ce code sera exécuté lorsque le timer aura atteint la valeur 0
22.   console.log("[setTimeout, fin du timer d'1 s],", heure(débutScript));
23.   // on émet un événement pour dire qu'un résultat est disponible
24.   eventEmitter.emit("timer1Success", { success: 4 });
25.   // on émet un autre événement pour dire qu'un autre résultat est disponible
26.   eventEmitter.emit("timer1Failure", { failure: 6 });
27. }, 1000)
28.
29. // on s'abonne à l'évt [timer1Success]
30. eventEmitter.on('timer1Success', (result) => {
31.   console.log(sprintf("la fonction asynchrone du timer a rendu le résultat [%j], %s, via
    l'événement [timer1Success]", result, heure(débutScript)));
32. });
33.
34. // on s'abonne à l'évt [timer1Failure]
35. eventEmitter.on('timer1Failure', (result) => {
36.   console.log(sprintf("la fonction asynchrone du timer a rendu le résultat [%j], %s, via
    l'événement [timer1Failure]", result, heure(débutScript)));
37. });
```

```
38.
39. // un code synchrone un peu intensif qui a empêcher le code principal de s'achever avant la
    fin de [timer1]
40. for (let i = 0; i < 1000000; i++) {
41.   for (let j = 0; j < 10000; j++) {
42.     i + i ^ 2 + i ^ 3;
43.   }
44. }
45.
46. // s'affichera avant les msg des evts émis par la fonction associée à [timer1]
47. console.log("[fin du script],", heure(débutScript));
48.
49. // utilitaire d'affichage heure et durée
50. function heure(début) {
51.  ...
52. }
```

Commentaires

- ce code est celui de l'exemple précédent **[async-02]** auquel on a ajouté les lignes 39-44 ;
- lignes 20-27 : la fonction **[setTimeout]** a été programmée pour exécuter une fonction asynchrone interne au bout d'un délai d'une seconde. Au bout de cette seconde, l'exécution de la fonction asynchrone du timer n'a pas lieu immédiatement : un événement est placé dans la boucle d'exécution pour demander celle-ci. Si la boucle d'exécution est occupée à traiter un autre événement, l'exécution de la fonction asynchrone du timer devra attendre ;
- lignes 20-27 : dès que la la fonction **[setTimeout]** a armé son timer d'un délai d'une seconde, elle lâche le processeur et rend la main au code appelant. Celui-ci continue avec les 30-37 qui sont des abonnements à des événements et qui ont un temps d'exécution négligeable ;
- le code principal continue avec les lignes 40-44 qui forment une boucle de 10^{10} itérations. Ce code sera en cours d'exécution lorsque le timer va émettre son événement de « fin du délai d'1 seconde ». Cet événement est alors mis dans la boucle événementielle mais devra attendre la fin d'exécution du code principal du script pour avoir une chance d'être traité ;
- ligne 47 : fin du code principal du script. C'est après ce dernier affichage que l'événement de fin de timer va pouvoir être traité et la fonction asynchrone interne à **[setTimeout]** va pouvoir être exécutée ;

Le script donne les résultats suivants :

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\async\async-03.js"
2.  [début du script], heure=08:55:02:665
3.  [fin du code principal du script], heure=08:55:11:789, durée= 9 seconde(s) et 131 millisecondes
4.  [setTimeout, fin du timer d'1 s], heure=08:55:11:794, durée= 9 seconde(s) et 136 millisecondes
5.  la fonction asynchrone du timer a rendu le résultat [{"success":4}], heure=08:55:11:794, durée=
    9 seconde(s) et 136 millisecondes, via l'événement [timer1Success]
6.  la fonction asynchrone du timer a rendu le résultat [{"failure":6}], heure=08:55:11:794, durée=
    9 seconde(s) et 136 millisecondes, via l'événement [timer1Failure]
7.
8.  [Done] exited with code=0 in 9.796 seconds
```

Commentaires

- ligne 3 : on voit que le code principal du script a mis 9 secondes à s'exécuter. Les événements qui ont pu se produire pendant ce temps ont été mis en attente dans la boucle événementielle ;
- ligne 4 : on voit que l'événement **[fin du timer]** a été traité 5 ms après la fin du code principal. Il a été émis environ 1 s après le début du script mais a du attendre 8s supplémentaires pour être finalement traité ;

On retiendra de cet exemple que dans un système événementiel, un code ne doit jamais occuper le processeur très longtemps. Si on a un code synchrone long à exécuter, on doit se « débrouiller » pour le décomposer en tâches asynchrones plus courtes qui signaleront leur fin avec un événement.

11.4 script [async-04]

Le script **[async-04]** montre un autre mécanisme, appelé **[Promise]**, une promesse de résultat. Ce mécanisme évite de gérer explicitement des événements **[node.js]**. C'est fait implicitement et le développeur peut alors ignorer l'existence de ces événements. Les comprendre lui permettra cependant de mieux appréhender le fonctionnement des **[Promise]** qui est de prime abord complexe.

Le type **[Promise]** est une classe Javascript. Son constructeur admet comme paramètre une fonction asynchrone à qui elle passe deux paramètres appelés traditionnellement **[resolve]** et **[reject]**. Ils pourraient porter un autre nom ;

```
6.  const promise=new Promise(function(resolve, reject){
7.          // une tâche asynchrone est lancée
8.          …
9.          // si réussite : appeler resolve(result) où [result] est le résultat de la tâche
    asynchrone ;
10.         // si échec : appeler reject(error) où [error] est un objet encapsulant l'erreur
    rencontrée ;
11. }
12. // s'abonner aux événements émis par la tâche asynchrone de la [Promise]
```

- le constructeur de **[Promise]** fait deux choses :
  - il crée un événement pour lancer l'exécution de la fonction **[function(resolve, reject)]** qu'on lui a passée en paramètre mais n'attend pas son résultat et rend immédiatement un objet **[Promise]** au code appelant. Celui-ci peut avoir quatre états :
    - **[pending]** : l'action asynchrone qui a rendu la **[Promise]** n'est pas encore terminée ;
    - **[fulfilled]** : l'action asynchrone qui a rendu la **[Promise]** s'est terminée avec succès ;
    - **[rejected]** : l'action asynchrone qui a rendu la **[Promise]** s'est terminée sur un échec ;
    - **[settled]** : l'action asynchrone qui a rendu la **[Promise]** est terminée ;

    Lorsque le constructeur rend son résultat, l'objet **[Promise]** créé est dans l'état **[pending]**, en attente des résultats de la fonction asynchrone ;
  - la tâche asynchrone des lignes 2-5 est **lancée immédiatement**. Les tâches asynchrones sont le plus souvent des tâches asynchrones d'entrée / sortie qui se décomposent de la façon suivante :
    1. exécution d'un code synchrone pour lancer l'opération d'E/S avec un autre organe, par exemple un serveur distant ;
    2. attente de la réponse de cet organe ;
    3. traitement de cette réponse ;

    C'est la phase 2 d'attente de l'organe extérieur au processeur qui est le plus coûteux. Plutôt que d'attendre :

    - la réception de la donnée demandée à l'organe extérieur va être signalée par un événement ;
    - dans le code synchrone qui va suivre la phase 1 (ligne 7 du code exemple), on va s'abonner à cet événement puis à un moment retourner dans la boucle d'événements de **[node.js]**. L'événement suivant dans la liste des événements en attente va alors être traité ;
    - pendant la phase 2, il y a parallélisme d'exécution mais sur des périphériques différents :
      - le processeur pour la boucle d'événements ;
      - un organe extérieur (disque, base de données, serveur distant) pour la recherche de la donnée demandée ;
    - à la fin de la phase 2, lorsque l'opération d'E/S a obtenu la donnée qu'elle demandait, un événement va être émis pour indiquer que le résultat de l'E/S est disponible. Cet événement va alors rejoindre les autres dans la liste d'attente des événements ;
    - lorsque son tour viendra, il sera traité. La fonction associée à cet événement (ligne 7 du code exemple) va alors être exécutée ;

    Ce mode de fonctionnement permet d'éviter les temps morts : celui où le processeur attend la réponse d'un périphérique plus lent que lui ;
  - lorsque la tâche asynchrone des lignes 2 et 5 a été lancée et a terminé son travail, elle a la possibilité de rendre un résultat au code appelant, grâce aux deux fonctions **[resolve, reject]** que le constructeur **[Promise]** lui a passé en paramètres. La convention est la suivante :
    - la tâche asynchrone signale un succès par **[resolve(result)]**. Cela revient à mettre dans la boucle d'événements de **[node.js]**, un événement qu'on pourrait appeler **[resolved]** avec **[result]** comme donnée associée ;
    - la tâche asynchrone signale un échec par **[reject(error)]**. Cela revient à mettre dans la boucle d'événements de **[node.js]**, un événement qu'on pourrait appeler **[rejected]** avec **[error]** comme donnée associée, en général un objet détaillant l'erreur qui s'est produite ;
    - il faut donc que le code appelant s'abonne à ces deux événements pour être prévenu de la disponibilité du résultat de la fonction asynchrone ;

Après l'exécution terminée de la tâche asynchrone encapsulée dans la **[Promise]**, l'état de l'objet **[promise]** rendu par le constructeur **[Promise(…)]** change :

- l'événement **[resolved]** le fait passer de l'état **[pending]** à **[resolved]** ;
- l'événement **[rejected]** le fait passer de l'état **[pending]** à **[rejected]** ;

L'abonnement aux événements **[resolved]** et **[rejected]** de la tâche asynchrone se fait avec des méthodes de la classe **[Promise]** avec la syntaxe suivante :

promise.then(f1).catch(f2).finally(f3) ;

où :

- **f1** est une fonction exécutée lorsque l'état de **[promise]** passe de **[pending]** à **[resolved]**, donc lorsque la tâche asynchrone a réussi son travail. Elle reçoit pour paramètre la valeur **[result]**, transmise par l'instruction **[resolve(result)]** de la tâche asynchrone ;
- **f2** est une fonction exécutée lorsque l'état de **[promise]** passe de **[pending]** à **[rejected]**, donc lorsque la tâche asynchrone a échoué à faire son travail. Elle reçoit pour paramètre la valeur **[error]**, transmise par l'instruction **[reject(errror)]** de la tâche asynchrone ;
- **f3** est une fonction exécutée après exécution des méthodes **[then]** ou **[catch]**, donc tout le temps exécutée. Elle ne reçoit aucun paramètre ;

Cette syntaxe cache complètement les événements auxquels on s'abonne. C'est pourtant un abonnement et comme celui de l'exemple précédent, il n'exécute pas immédiatement les fonctions **[f1, f2, f3]**. Celles-ci seront exécutées ou pas lorsque l'un des événements **[resolved, rejected]** auxquels on s'abonne va se produire.

Le script **[async-04]** montre cette mécanique :

```
1.  'use strict';
2.
3.  // il est possible d'obtenir les résultats (succes, failure) d'une fonction asynchrone
4.  // sans utiliser explicitement des événements grâce à la classe [Promise]
5.  // cette classe utilise implicitement des événements mais ceux-ci ne se voient pas dans le
    code
6.
7.  // imports
8.  import moment from 'moment';
9.  import { sprintf } from 'sprintf-js';
10.
11. // début
12. const débutScript = moment(Date.now());
13. console.log("[début du script],", heure(débutScript));
14.
15. // définition d'une tâche asynchrone à l'aide d'une promesse [Promise]
16. // la tâche asynchrone est le paramètre du constructeur [Promise]
17. const débutPromise1 = moment(Date.now());
18. const promise1 = new Promise(function (resolve) {
19.   // log
20.   console.log("[début fonction asynchrone de promise1],", heure(débutPromise1));
21.   // code asynchrone
22.   setTimeout(function () {
23.     console.log("[fin fonction asynchrone de promise1],", heure(débutPromise1));
24.     // la tâche asynchrone rend un résultat avec la fonction [resolve]
25.     // la promesse est alors réussie
26.     resolve('[réussite]');
27.   }, 1000)
28. });
29.
30. // on peut connaître le résultat de la promesse [promise1]
31. // lorsque celle-ci a été résolue (resolve) ou rejetée (reject)
32. // l'instruction qui suit est un abonnement à l'évt [resolved] via la méthode [then]
33. // et à l'évt [rejected] via la méthode [catch]
34. // la méthode [finally] est exécutée que ce soit après un then ou un catch
35. promise1.then(result => {
36.   // cas de réussite de la promesse  [evt resolved]
37.   console.log(sprintf("[promise1.then], %s, result=%s", heure(débutPromise2), result));
38. }).catch(result => {
39.   // cas d'erreur  [evt rejected]
40.   console.log(sprintf("[promise1.catch], %s, result=%s", heure(débutPromise2), result));
41. }).finally(() => {
42.   // exécuté dans tous les cas
```

```
43.   console.log("[promise1.finally]", heure(débutPromise1));
44. });
45.
46. // définition d'une tâche asynchrone à l'aide d'une promesse [Promise]
47. const débutPromise2 = moment(Date.now());
48. const promise2 = new Promise(function (resolve, reject) {
49.   // log
50.   console.log("[début fonction asynchrone de promise2],", heure(débutPromise1));
51.   // tâche asynchrone
52.   setTimeout(function () {
53.     console.log("[fin fonction asynchrone de promise2],", heure(débutPromise2));
54.     // la tâche asynchrone rend un résultat avec la fonction [reject]
55.     // la promesse est alors ratée
56.     reject('[échec]');
57.   }, 2000)
58. });
59.
60. // on peut connaître le résultat de la promesse [promise2]
61. // lorsque celle-ci a été résolue (resolve) ou rejetée (reject)
62. promise2.then(result => {
63.   // cas de réussite de la promesse [evt resolved]
64.   console.log(sprintf("[promise2.then], %s, result=%s", heure(débutPromise2), result));
65. }).catch(result => {
66.   // cas d'erreur [evt rejected]
67.   console.log(sprintf("[promise2.catch], %s, result=%s", heure(débutPromise2), result));
68. }).finally(() => {
69.   // exécuté dans tous les cas
70.   console.log(sprintf("[promise2.finally], %s", heure(débutPromise2)));
71. });
72.
73. // s'affichera avant les msg des fonctions asynchrones et ceux des évts associés
74. console.log("[fin du code principal du script],", heure(débutScript));
75.
76. // utilitaire
77. function heure(début) {
78.   // heure du moment courant
79.   const now = moment(Date.now());
80.   // formatage heure
81.   let result = "heure=" + now.format("HH:mm:ss:SSS");
82.   if (début) {
83.     const durée = now - début;
84.     const milliseconds = durée % 1000;
85.     const seconds = Math.floor(durée / 1000);
86.     // formatage durée
87.     result = result + sprintf(", durée= %s seconde(s) et %s millisecondes", seconds,
    milliseconds);
88.   }
89.   // résultat
90.   return result;
91. }
```

Commentaires

- lignes 18-28 : création d'une **[Promise promise1]**. Sa fonction asynchrone rend son résultat via un événement au bout d'une seconde. Une fois cet opération asynchrone lancée (armement d'un timer), on n'attend pas que celle-ci rend son résultat et on passe tout de suite au code de la ligne 35 ;
- lignes 35-44 : on s'abonne aux deux événements **[resolved, rejected]** que la fonction asynchrone interne à **[promise1]** peut émettre ;
- lignes 46-71 : on répète la même séquence de code que précédemment pour une seconde promesse **[promise2]** ;
- ligne 74 : le code principal du script est terminé mais pas le script dans son ensemble car deux actions asynchrones ont été lancées. On retourne dans la boucle événementielle où à un moment l'un des événements **[resolved, rejected]** des promesses **[promise1, promise2]** va se produire. Il sera alors traité ;
- puis il y aura retour à la boucle événementielle. Et là le second événement **[resolved, rejected]** des promesses **[promise1, promise2]** sera traité lorsqu'il se produira ;

Exécution

```
1. [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
   \Data\st-2019\dev\es6\javascript\async\async-04.js"
2. [début du script], heure=09:39:05:950, durée= 0 seconde(s) et 3 millisecondes
```

```
3.  [début fonction asynchrone de promise1], heure=09:39:05:958, durée= 0 seconde(s) et 0 milliseco
    ndes
4.  [début fonction asynchrone de promise2], heure=09:39:05:959, durée= 0 seconde(s) et 1 milliseco
    ndes
5.  [fin du code principal du script], heure=09:39:05:960, durée= 0 seconde(s) et 13 millisecondes
6.  [fin fonction asynchrone de promise1], heure=09:39:06:977, durée= 1 seconde(s) et 19 millisecon
    des
7.  [promise1.then], heure=09:39:06:980, durée= 1 seconde(s) et 21 millisecondes, result=[réussite]
8.  [promise1.finally] heure=09:39:06:982, durée= 1 seconde(s) et 24 millisecondes
9.  [fin fonction asynchrone de promise2], heure=09:39:07:976, durée= 2 seconde(s) et 17 millisecon
    des
10. [promise2.catch], heure=09:39:07:978, durée= 2 seconde(s) et 19 millisecondes, result=[échec]
11. [promise2.finally], heure=09:39:07:980, durée= 2 seconde(s) et 21 millisecondes
12.
13. [Done] exited with code=0 in 2.589 seconds
```

Commentaires

- ligne 3 : la fonction asynchrone de `[promise1]` est lancée mais on n'attend pas sa fin qui sera signalée par un événement ;
- ligne 4 : la fonction asynchrone de `[promise2]` est lancée mais on n'attend pas sa fin qui sera signalée par un événement ;
- ligne 5 : fin du code principal et retour à la boucle événementielle ;
- ligne 6 : traitement de l'événement `[fin fonction asynchrone de promise1]`. L'état de `[promise1]` va passer à `[resolved]`. Un événement le signale ;
- ligne 7 : `[promise2]` n'ayant toujours pas fini son travail, l'événement `[promise1 resolved]` qui vient d'être mis dans la boucle va être traité par la méthode `[promise1.then]` puis par la méthode `[promise.finally]` (ligne 8) ;
- lignes 9-11 : le même mécanisme se déroule lorsque `[promise2]` passe de l'état `[pending]` à `[resolved]` ;

11.5 script [async-05]

Revenons au code du constructeur d'un objet `[Promise]` :

```
13. const promise=new Promise(function(resolve, reject){
14.         // une tâche asynchrone est lancée
15.         …
16.         // si réussite : appeler resolve(result) où [result] est le résultat de la tâche
    asynchrone ;
17.         // si échec : appeler reject(error) où [error] est un objet encapsulant l'erreur
    rencontrée ;
18. }
19. // on s'abonne aux événements émis par la tâche asynchrone
```

Ligne 2, la tâche asynchrone de la `[Promise]` est lancée. Elle a souvent besoin de davantage de paramètres que les seuls paramètres `[resolve, reject]` que la fonction qui l'encapsule lui passe. Dans ce cas, on encapsule la création de la `[Promise]` dans une fonction qui va lui passer les paramètres dont sa fonction asynchrone a besoin :

```
20. // définition de la fonction asynchrone
21. function uneFonctionAsynchrone (p1, p2, …, pn){
22.  return new Promise(function(resolve, reject){
23.         // une tâche asynchrone est lancée avec les paramètres (P1, p2, …, pn)
24.         …
25.         // si réussite : appeler resolve(result) où [result] est le résultat de la tâche
    asynchrone ;
26.         // si échec : appeler reject(error) où [error] est un objet encapsulant l'erreur
    rencontrée ;
27. }
28. // on s'abonne aux évts [resolved, rejected] que va émettre la fonction asynchrone
    [uneFonctionAsynchrone]
29. …
30. // qq temps plus tard, la fonction asynchrone [uneFonctionAsynchrone] est appelée
31. uneFonctionAsynchrone(e1, e2, …, en) ;
```

Le script suivant :

- définit deux fonctions asynchrones rendant une `[Promise]` ;
- lance leur exécution en parallèle et attend que les deux soient terminées pour faire un certain travail ;

```
1.  'use strict';
2.
3.  // on peut définir des fonctions asynchrones qui rendent un type [Promise]
4.  // elles peuvent être alors taguées avec le mot clé [async]
5.
6.  // imports
7.  import moment from 'moment';
8.  import { sprintf } from 'sprintf-js';
9.
10. // début
11. const débutScript = moment(Date.now());
12. console.log("[début du script],", heure());
13.
14. // une fonction asynchrone qui rend une promesse [Promise]
15. function async01(p1) {
16.   return new Promise((resolve) => {
17.     console.log("[début de la tâche asynchrone async01]");
18.     // la tâche asynchrone
19.     const débutAsync01 = moment(Date.now());
20.     setTimeout(function () {
21.       console.log("[fin de la tâche asynchrone async01],", heure(débutAsync01));
22.       // la tâche asynchrone peut rendre un résultat complexe
23.       resolve({
24.         prop1: [10, 20, 30],
25.         prop2: "abcd",
26.         prop3: p1,
27.       });
28.     }, 1000)
29.   });
30. }
31.
32. // une fonction asynchrone qui rend une promesse [Promise]
33. function async02(p1, p2) {
34.   return new Promise(resolve => {
35.     console.log("[début de la tâche asynchrone async02]");
36.     // tâche asynchrone
37.     const débutAsync02 = moment(Date.now());
38.     setTimeout(function () {
39.       console.log("[fin de la tâche asynchrone async02],", heure(débutAsync02));
40.       // la tâche asynchrone peut rendre un résultat complexe
41.       resolve({
42.         prop1: [11, 21, 31],
43.         prop2: "xyzt",
44.         prop3: p1 + p2
45.       });
46.     }, 2000)
47.   })
48. }
49.
50. // on lance les deux fonctions asynchrones en parallèle
51. // et on attend qu'elles aient terminé toutes les deux
52. // le then ne s'exécute que si les deux fonctions ont émis l'évt [resolved]
53. // le catch s'exécute dès que l'une des deux fonctions émet l'évt [rejected]
54. Promise.all([async01(10), async02(10, 20)])
55.   // le résultat est un tableau [result1, result2] où [result1] est le résultat émis par un
    [resolve] de [async01]
56.   // et [result2] le résultat émis par un [resolve] de [async02]
57.   .then(result => {
58.     console.log(sprintf("[promise-all success], %s, result=%j", heure(débutScript), result));
59.   })
60.   // error est le résultat émis par le premier [reject] de l'une des deux fonctions
    asynchrones
61.   .catch(error => {
62.     console.log(sprintf("[promise-all error], %s, erreur=%j", heure(débutScript), error));
63.   })
64.   // finally est exécuté après le then ou le catch
65.   .finally(() => {
66.     console.log(sprintf("[promise-all finally], %s", heure(débutScript)));
67.   });
68.
69. // s'affichera avant les msgs des fonctions asynchrones et des évts associés
70. console.log("[fin du code principal du script],", heure(débutScript));
71.
72. // utilitaire
```

```
73. function heure(début) {
74.   // heure du moment courant
75.   const now = moment(Date.now());
76.   // formatage heure
77.   let result = "heure=" + now.format("HH:mm:ss:SSS");
78.   if (début) {
79.     const durée = now - début;
80.     const milliseconds = durée % 1000;
81.     const seconds = Math.floor(durée / 1000);
82.     // formatage durée
83.     result = result + sprintf(", durée= %s seconde(s) et %s millisecondes", seconds,
    milliseconds);
84.   }
85.   // résultat
86.   return result;
87. }
```

Commentaires

- lignes 15-30 : on définit une fonction **[async01]** qui rend son résultat au bout d'1 seconde via un événement de timer. La fonction **[async01]** qui est utilisée dans son résultat ligne 26 ;
- lignes 33-47 : on fait de même avec une fonction **[async02]** qui rend son résultat au bout de 2 secondes via un événement de timer. La fonction **[async02]** admet deux paramètres qui sont utilisés dans son résultat ligne 44 ;
- lorsqu'elles seront appelées les deux fonctions **[async01, async02]** :
  - seront lancées ;
  - rendront au code appelant deux promesses **[promise1, promise2]** ;
  - l'exécution reviendra alors au code appelant qui continuera sa course ;
  - au bout d'1 seconde environ **[async01]** émettra un événement pour dire qu'elle a terminé son travail. L'événement en question sera mis en attente dans la boucle événementielle associé au résultat transmis par **[async01]** avec l'événement ;
  - au bout de 2 secondes environ, le même processus se passera pour **[async02]** ;
- ligne 54 : ce n'est que maintenant que les fonctions asynchrones **[async01, async02]** sont exécutées (notations async01(10) et async02(10,20)). Elles le sont au sein d'un tableau passé en paramètre à la méthode **[Promise.all]**. On sait que **[async01, async02]** rendent toutes deux une promesse au code appelant. Aussi le paramètre de **[Promise.all]** est un tableau de deux promesses ;
- **[Promise.all([promise1, promise2, …, promisen]).then(f1).catch(f2).finally(f3)]** est un abonnement à des événements :
  - **[Promise.all]** est de type **[Promise]** ;
  - la fonction **[f1]** de la méthode **[then]** sera exécutée lorsque **toutes les promesses** **[promise1, promise2, …, promisen]** du tableau paramètre de la méthode **[all]** seront passées de l'état **[pending]** à l'état **[resolved]**. Dit autrement, **[f1]** sera exécutée lorsque toutes les promesses du tableau se seront terminées avec succès ;
  - la fonction **[f2]** de la méthode **[catch]** sera exécutée dès que l'une des promesses du tableau passera de l'état **[pending]** à l'état **[rejected]**. Dit autrement, **[f2]** est exécutée dès que l'une des promesses du tableau échoue ;
  - la fonction **[f3]** de la méthode **[finally]** sera exécutée après l'exécution d'une des méthodes **[then, catch]**, donc toujours exécutée ;

L'exécution du code donne les résultats suivants :

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\async\async-05.js"
2.  [début du script], heure=12:17:17:367
3.  [début de la tâche asynchrone async01]
4.  [début de la tâche asynchrone async02]
5.  [fin du code principal du script], heure=12:17:17:375, durée= 0 seconde(s) et 10 millisecondes
6.  [fin de la tâche asynchrone async01], heure=12:17:18:391, durée= 1 seconde(s) et 17 millisecond
    es
7.  [fin de la tâche asynchrone async02], heure=12:17:19:389, durée= 2 seconde(s) et 14 millisecond
    es
8.  [promise-all success], heure=12:17:19:390, durée= 2 seconde(s) et 25 millisecondes, result
    =[{"prop1":[10,20,30],"prop2":"abcd","prop3":10},{"prop1":
    [11,21,31],"prop2":"xyzt","prop3":30}]
9.  [promise-all finally], heure=12:17:19:392, durée= 2 seconde(s) et 27 millisecondes
10.
11. [Done] exited with code=0 in 2.572 seconds
```

- lignes 6-7 : les deux tâches asynchrones **[async01, async02]** sont lancées. Elles fonctionnent en parallèle. Ce n'est pas l'exécution de leur code qui est faite en parallèle mais leurs attentes respectives de la donnée demandée se passent en même temps ;
- ligne 5 : le code principal du script est terminé. Restent à attendre la fin des deux tâches asynchrones **[async01, async02]** ;
- ligne 6 : la tâche asynchrone **[async01]** se termine environ 1 s après son lancement. Elle rend un résultat avec la fonction **[resolve]** donc sa promesse dans le tableau de la ligne 56 du code, passe de l'état **[pending]** à **[resolved]**. Ce n'est pas suffisant pour déclencher la méthode **[then]**, lignes 59-60 du code ;
- ligne 7 : la tâche asynchrone **[async02]** se termine environ 2 s après son lancement. Elle rend un résultat avec la fonction **[resolve]** donc sa promesse dans le tableau de la ligne 56 du code, passe de l'état **[pending]** à **[resolved]**. La méthode **[then]** va être exécutée dès que la boucle événementielle le permettra ;
- ligne 8 : la méthode **[then]** de **[Promise.all]** est exécutée. Elle reçoit en paramètre un tableau **[result1, result2]** où **[result1]** est le résultat émis par **[async01]**, et **[result2]** celui émis par **[async02]** ;
- ligne 9 : la méthode **[finally]** de **[Promise.all]** est exécutée ;

11.6 script [async-06]

Ce nouveau script montre comment l'utilisation conjointe des mots clés **[async / await]** permet d'avoir un code **asynchrone** ressemblant à un code **synchrone**. La gestion des événements est complètement cachée et la compréhension du code facilitée.

Nous reprenons l'exemple précédent en y amenant les modification suivantes :

- on ajoute une troisième fonction asynchrone **[async03]** qui elle renvoie son résultat avec la méthode **[Promise.reject]** qui signale donc à la boucle événementielle qu'elle a « échoué » à faire son travail ;
- on exécute séquentiellement les trois fonctions asynchrones **[async01, async02, async03]**. Dans l'exemple précédent, on avait exécuté en parallèle les fonctions asynchrones **[async01, async02]** ;
- avant l'apparition des mots clés **[async/await]**, l'exécution séquentielle d'actions asynchrones se faisait à l'aide de **[Promise]** emboîtées les unes dans les autres. Dès qu'il y avait plusieurs actions asynchrones à exécuter ainsi, le nombre de promesses augmentait en conséquence et le code devenait moins lisible ;
- avec les mots clés **[async/await]**, l'exécution séquentielle de tâches asynchrones se fait avec une syntaxe à celle de l'exécution de tâches synchrones :

```
1.  // fonction asynchrone - utilisation async / await
2.  async function main() {
3.    // exécution séquentielle des tâches asynchrones
4.    try {
5.      // exécution avec attente de [async01]
6.      const result1 = await async01(...);
7.      console.log("[async01 result]=", result1);
8.      // exécution avec attente de [async02]
9.      const result2 = await async02(...);
10.     console.log("[async02 result]=", result2);
11.     // exécution avec attente de [async03]
12.     const result3 = await async03(...);
13.     console.log("[async03 result]=", result3);
14.   } catch (error) {
15.     // une des actions asynchrones a échoué
16.     console.log(sprintf("[sequential error]= %j, %s", error));
17.   } finally {
18.     // terminé
19.     console.log("[fin exécution séquentielle des tâches asynchrones],");
20.   }
```

- ligne 6 : la fonction asynchrone **[async01]** est lancée (mot clé **await**) et on attend qu'elle ait publié son résultat par l'une des méthodes **[Promise.resolve, Promise.reject]**. C'est donc une opération **bloquante** ;
- ligne 6 : le mot clé **[await]** transforme l'opération asynchrone **[async01]** en opération bloquante. On sait que l'opération **[async01]** rend un résultat de deux façons :
  - elle **rend** au code appelant, quasi immédiatement, un objet **[Promise]** ;
  - elle **publie** ultérieurement un résultat sur la boucle événementielle via les méthodes **[Promise.resolve, Promise.reject]**. C'est ce dernier résultat que récupère **[result1]**, ligne 6. La gestion événementielle de l'action **[async01]** est devenue invisible ;
  - si le résultat **[result]** de **[async01]** est publié par **[Promise.resolve(result)]**, il est affecté à **[result1]** ligne 6 et l'exécution continue ligne 7 ;

    - si le résultat de `[async01]` est publié par `[Promise.reject]`, cela provoque une exeption et l'exécution du code passe à la ligne 14, celle du **catch**. Le paramètre de la clause `[catch]` est l'objet d'erreur (error) publié par `[async01]` avec une expression `[Promise.reject(error)]`. La tâche asynchrone peut également publier l'erreur par un `[throw(error)]`. L'objet `[error]` est celui récupéré dans `[catch(error)]` ;
    - le mot clé `[await]` doit être obligatoirement dans une fonction précédée du mot clé `[async]`, ligne 2. Ce mot clé indique que la fonction `[main]` est une fonction asynchrone ;
    - dans l'expression `[await f(…)]`, `[f]` doit être une fonction asynchrone rendant un objet `[Promise]` au code appelant ;
- on refait la même chose pour l'action asynchrone `[async02]`, ligne 9 et `[async03]`, ligne 12 ;

Toujours avec les mots clés `[async / await]`, il est possible de faire l'exécution parallèle de tâches asynchrones avec la syntaxe suivante :

```
1.  try {
2.      // exécution parallèle des tâches asynchrones
3.      const result = await Promise.all([async01(...), async02(...), async03(...)]);
4.      console.log(sprintf("[parallel success], %s, result=%j", heure(débutParallel), result));
5.    } catch (error) {
6.      // une des actions asynchrones a échoué
7.      console.log(sprintf("[parallel error], %s, erreur=%j", heure(débutParallel), error));
8.    } finally {
9.      // terminé
10.     console.log(sprintf("[fin exécution parallèle des tâches asynchrones],%s",
    heure(débutParallel)));
11. }
```

- ligne 3 : on a une opération **bloquante** : on attend que les trois tâches asynchrones du tableau `[async01(..), async02(..), async03(..)]` aient publié leurs résultats sur la boucle événementielle avec l'une des méthodes `[Promise.resolve, Promise.reject]` ;
- si les trois tâches asynchrones publient leurs résultats avec `[Promise.resolve]`, la constante `[result]` est alors le tableau `[result1, result2, result3]` où :
    - `[result1]` est le résultat publié par `[async01]` avec l'expression `[Promise.resolve(result1)]` ;
    - `[result2]` est le résultat publié par `[async02]` avec l'expression `[Promise.resolve(result2)]` ;
    - `[result3]` est le résultat publié par `[async03]` avec l'expression `[Promise.resolve(result3)]` ;
- si l'une des trois tâches publie son résultat avec une expression `[Promise.reject(error)]` alors une exception se produit ;
    - la constante `[result]` de la ligne 3 ne reçoit pas sa valeur ;
    - l'exécution passe directement au `[catch]` de la ligne 5 ;
    - le paramètre (error) du catch, est l'objet (error) publié par l'expression `[Promise.reject(error)]` ;

En mixant ces deux syntaxes, on peut exécuter indifféremment des tâches asynchrones en séquentiel ou en parallèle, tout cela avec une syntaxe analogue à celle d'un code synchrone. Il faut donc privilégier cette syntaxe beaucoup plus lisible que les précédentes. Cette syntaxe `[async / await]` n'est disponible que depuis la version 6 d'ECMAScript. Il y a encore beaucoup de codes Javascript utilisant des promesses `[Promise]`. C'est pourquoi il est important de comprendre également le fonctionnement de celles-ci.

Le code complet du script `[async-06]` est le suivant :

```
1.  'use strict';
2.
3.  // exécution parallèle ou séquentielle de plusieurs tâches asynchrones
4.  // avec les mots clés async / await
5.
6.  // imports
7.  import moment from 'moment';
8.  import { sprintf } from 'sprintf-js';
9.
10. // début
11. const débutScript = moment(Date.now());
12. console.log("[début du code principal du script],", heure());
13.
14. // une fonction asynchrone rendant une [Promise]
15. function async01(débutAsync01) {
16.   return new Promise(function (resolve) {
17.     console.log("[début fonction asynchrone async01],", heure());
18.     // fonction asynchrone
19.     setTimeout(function () {
```

```
20.       console.log("[fin fonction asynchrone async01],", heure(débutAsync01));
21.       // l'action asynchrone peut rendre un résultat complexe
22.       // ici réussite
23.       resolve({
24.         prop1: [11, 21, 31],
25.         prop2: "abcd"
26.       });
27.     }, 1000)
28.   });
29. }
30.
31. // une fonction asynchrone rendant une [Promise]
32. function async02(débutAsync02) {
33.   console.log("[début fonction asynchrone async02],", heure());
34.   return new Promise(function (resolve) {
35.     // fonction asynchrone
36.     setTimeout(function () {
37.       console.log("[fin fonction asynchrone async02],", heure(débutAsync02));
38.       // l'action asynchrone peut rendre un résultat complexe
39.       // ici réussite
40.       resolve({
41.         prop1: [12, 22, 32],
42.         prop2: "xyzt"
43.       });
44.     }, 2000)
45.   })
46. }
47.
48. // une fonction asynchrone rendant une [Promise]
49. function async03(débutAsync03) {
50.   console.log("[début fonction asynchrone async03],", heure());
51.   return new Promise((resolve, reject) => {
52.     // fonction asynchrone
53.     setTimeout(function () {
54.       console.log("[fin fonction asynchrone async03],", heure(débutAsync03));
55.       // l'action asynchrone peut rendre un résultat complexe
56.       // ici échec
57.       reject({
58.         prop1: [13, 23, 33],
59.         prop2: "échec"
60.       });
61.     }, 3000)
62.   })
63. }
64.
65. // fonction asynchrone - utilisation async / await
66. async function main() {
67.   const débutSequential = moment(Date.now());
68.   // exécution séquentielle des tâches asynchrones
69.   console.log("------------ exécution séquentielle des tâches asynchrones lancée
    ------------------------")
70.   try {
71.     // exécution avec attente de [async01]
72.     const débutAsync01 = moment(Date.now());
73.     const result1 = await async01(débutAsync01);
74.     console.log("[async01 result]=", result1);
75.     // exécution avec attente de [async02]
76.     const débutAsync02 = moment(Date.now());
77.     console.log("début async02-------------", heure());
78.     const result2 = await async02(débutAsync02);
79.     console.log("[async02 result]=", result2);
80.     // exécution avec attente de [async03]
81.     const débutAsync03 = moment(Date.now());
82.     console.log("début async03-------------", heure());
83.     const result3 = await async03(débutAsync03);
84.     console.log("[async03 result]=", result3);
85.   } catch (error) {
86.     // une des actions asynchrones a échoué
87.     console.log(sprintf("[sequential error]= %j, %s", error, heure(débutSequential)));
88.   } finally {
89.     // terminé
90.     console.log("[fin exécution séquentielle des tâches asynchrones],",
    heure(débutSequential));
91.   }
```

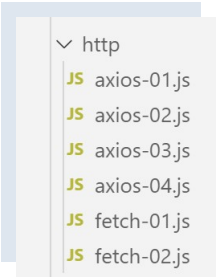
```
92.
93.   const débutParallel = moment(Date.now());
94.   // exécution en parallèle des tâches asynchrones
95.   console.log("------------ exécution parallèle des tâches asynchrones lancée
    ------------------------")
96.   try {
97.     const result = await Promise.all([async01(débutParallel), async02(débutParallel),
    async03(débutParallel)]);
98.     console.log(sprintf("[parallel success], %s, result=%j", heure(débutParallel), result));
99.   } catch (error) {
100.    // une des actions asynchrones a échoué
101.    console.log(sprintf("[parallel error], %s, erreur=%j", heure(débutParallel), error));
102.  } finally {
103.    // terminé
104.    console.log(sprintf("[fin exécution parallèle des tâches asynchrones],%s",
    heure(débutParallel)));
105.  }
106.
107.  // terminé
108.  console.log("[fin de la fonction main],", heure(débutSequential));
109.}
110.// exécution fonction asynchrone main
111.main();
112.
113.// s'affichera avant les différents msgs des fonctions asynchrones et de leurs évts
114.console.log("[fin du code principal du script],", heure(débutScript));
115.
116.// utilitaire
117.function heure(début) {
118.  // heure du moment courant
119.  const now = moment(Date.now());
120.  // formatage heure
121.  let result = "heure=" + now.format("HH:mm:ss:SSS");
122.  if (début) {
123.    const durée = now - début;
124.    const milliseconds = durée % 1000;
125.    const seconds = Math.floor(durée / 1000);
126.    // formatage durée
127.    result = result + sprintf(", durée= %s seconde(s) et %s millisecondes", seconds,
    milliseconds);
128.  }
129.  // résultat
130.  return result;
131.}
```

Les résultats de l'exécution sont les suivants :

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\async\async-06.js"
2.  [début du code principal du script], heure=15:02:00:152
3.  ------------ exécution séquentielle des tâches asynchrones lancée ------------------------
4.  [début fonction asynchrone async01], heure=15:02:00:161
5.  [fin du code principal du script], heure=15:02:00:164, durée= 0 seconde(s) et 15 millisecondes
6.  [fin fonction asynchrone async01], heure=15:02:01:165, durée= 1 seconde(s) et 4 millisecondes
7.  [async01 result]= { prop1: [ 11, 21, 31 ], prop2: 'abcd' }
8.  début async02------------- heure=15:02:01:253
9.  [début fonction asynchrone async02], heure=15:02:01:254
10. [fin fonction asynchrone async02], heure=15:02:03:265, durée= 2 seconde(s) et 12 millisecondes
11. [async02 result]= { prop1: [ 12, 22, 32 ], prop2: 'xyzt' }
12. début async03------------- heure=15:02:03:268
13. [début fonction asynchrone async03], heure=15:02:03:268
14. [fin fonction asynchrone async03], heure=15:02:06:285, durée= 3 seconde(s) et 18 millisecondes
15. [sequential error]= {"prop1":[13,23,33],"prop2":"échec"}, heure=15:02:06:289, durée= 6 seconde(
    s) et 129 millisecondes
16. [fin exécution séquentielle des tâches asynchrones], heure=15:02:06:291, durée= 6 seconde(s) et
    131 millisecondes
17. ------------ exécution parallèle des tâches asynchrones lancée ------------------------
18. [début fonction asynchrone async01], heure=15:02:06:292
19. [début fonction asynchrone async02], heure=15:02:06:293
20. [début fonction asynchrone async03], heure=15:02:06:294
21. [fin fonction asynchrone async01], heure=15:02:07:294, durée= 1 seconde(s) et 2 millisecondes
22. [fin fonction asynchrone async02], heure=15:02:08:298, durée= 2 seconde(s) et 6 millisecondes
23. [fin fonction asynchrone async03], heure=15:02:09:297, durée= 3 seconde(s) et 5 millisecondes
```

```
24. [parallel error], heure=15:02:09:298, durée= 3 seconde(s) et 6 millisecondes, erreur={"prop1":
    [13,23,33],"prop2":"échec"}
25. [fin exécution parallèle des tâches asynchrones],heure=15:02:09:299, durée= 3 seconde(s) et 7 m
    illisecondes
26. [fin de la fonction main], heure=15:02:09:300, durée= 9 seconde(s) et 140 millisecondes
27.
28. [Done] exited with code=0 in 9.668 seconds
```

12 Les fonctions HTTP de Javascript



12.1 Choix d'une bibliothèque HTTP

Nous avons fait ici le choix de deux bibliothèques :

EcmaScript 6 a nativement une fonction HTTP appelée `[fetch]` qui n'est pas implémentée par `[node.js]` (sept 2019). Il existe une bibliothèque appelée `[node-fetch]` qui permet d'utiliser la fonction `[fetch]` sous Node. Cette bibliothèque utilise certaines API propres à `[node.js]`. Un code `[node-fetch]` peut être alors non transportable à 100 % dans un environnement non `[node]`, dans un navigateur par exemple ;

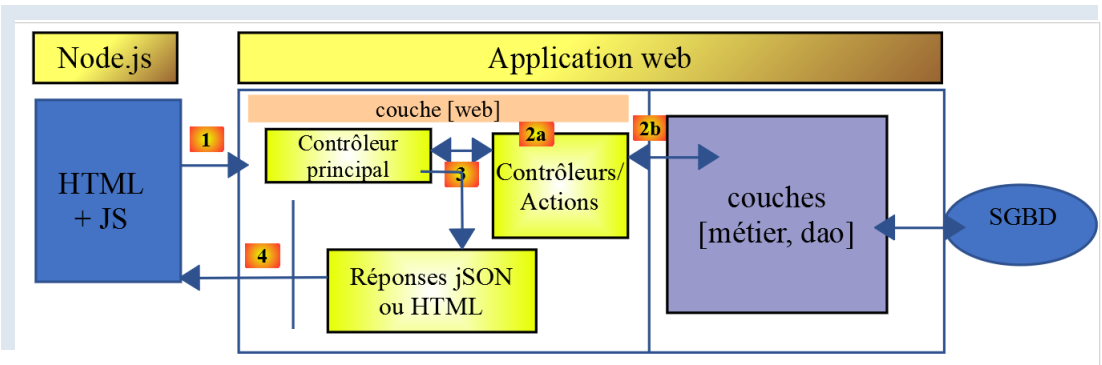
Il existe par ailleurs une bibliothèque nommée `[axios]` dédiée aux requêtes HTTP compatible aussi bien avec `[node.js]` qu'avec les navigateurs. C'est cette bibliothèque que nous utiliserons au final.

Nous allons présenter un même script écrit avec ces deux bibliothèques pour montrer que la démarche de codage avec elles est analogue.

12.2 Mise en place d'un environnement de travail

12.2.1 Installation du serveur de calcul d'impôt

Ultimement, nous allons écrire une application web avec l'architecture suivante :



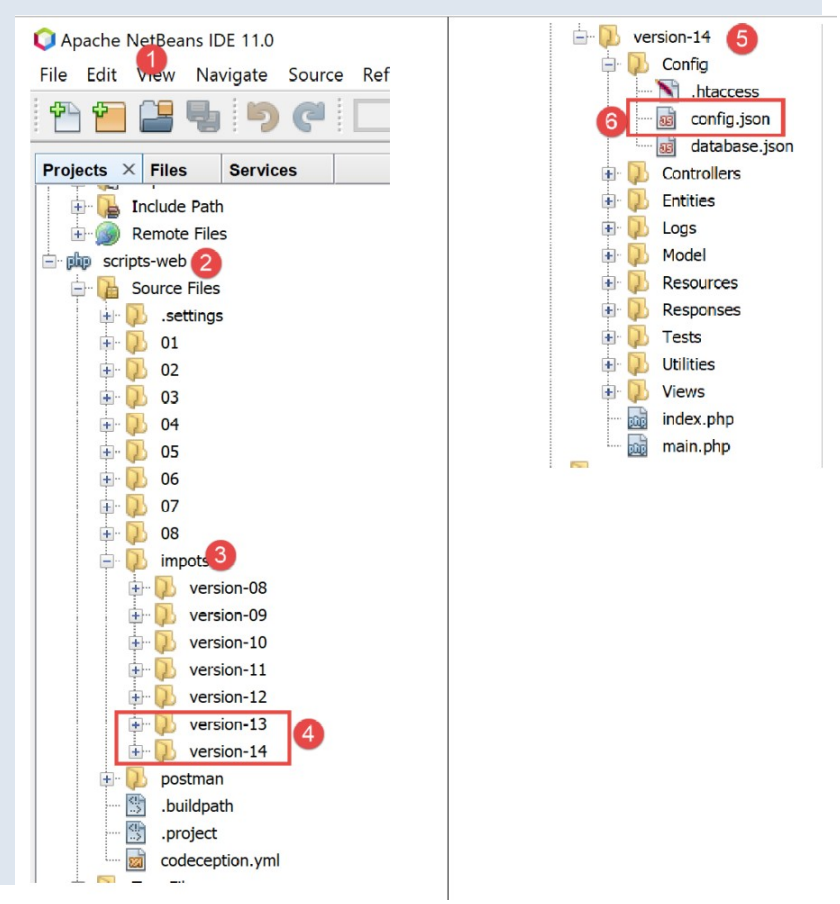
JS : Javascript

Le code Javascript est client :

- d'un service de pages ou fragments statiques ;
- d'un service 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.

Le serveur sera celui du calcul de l'impôt dont nous avons déjà écrit 13 versions. Nous allons en écrire une 14ième. Nous commençons donc par dupliquer, sous Netbeans, le dossier de la version 13, dans le dossier de la version 14 :



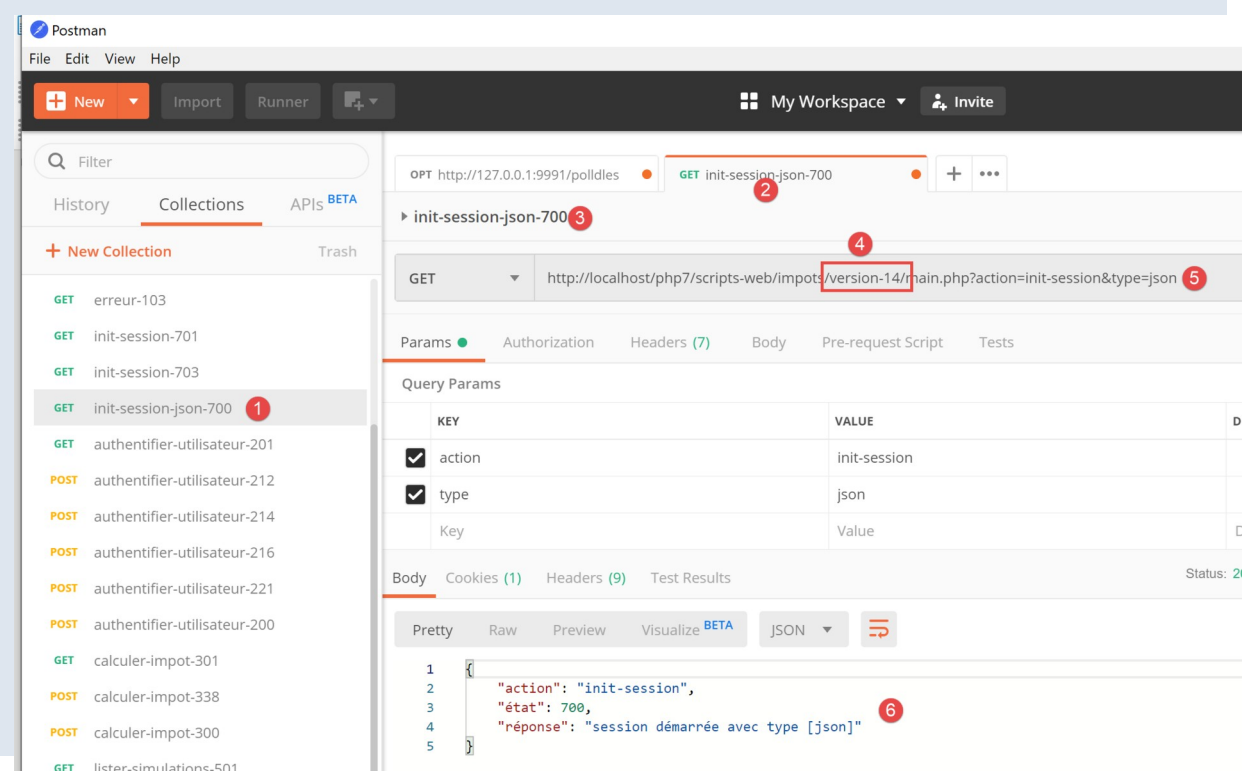
- en **[6]**, nous modifions le fichier **[config.json]** de la version 14 de la façon suivante :

```
1.  {
2.      "databaseFilename": "Config/database.json",
3.      "rootDirectory": "C:/myprograms/laragon-lite/www/php7/scripts-web/impots/version-14",
4.      "relativeDependencies": [
5.
6.          "/Entities/BaseEntity.php",
7.          "/Entities/Simulation.php",
8.          ...
9.      "vues": {
10.         "vue-authentification.php": [700, 221, 400],
11.         "vue-calcul-impot.php": [200, 300, 341, 350, 800],
12.         "vue-liste-simulations.php": [500, 600]
13.     },
14.     "vue-erreurs": "vue-erreurs.php"
15. }
```

- ligne 3, nous changeons le dossier racine de l'application ;

Pour accéder à ce serveur, il faut lancer les services **[Laragon]**.

Ceci fait, nous pouvons tester avec **[Postman]** (cf. article lien), cette nouvelle version du serveur, identique pour l'instant à la version 13. Nous pouvons utiliser la collection de requêtes utilisées pour tester la version 12 du serveur de calcul d'impôt :



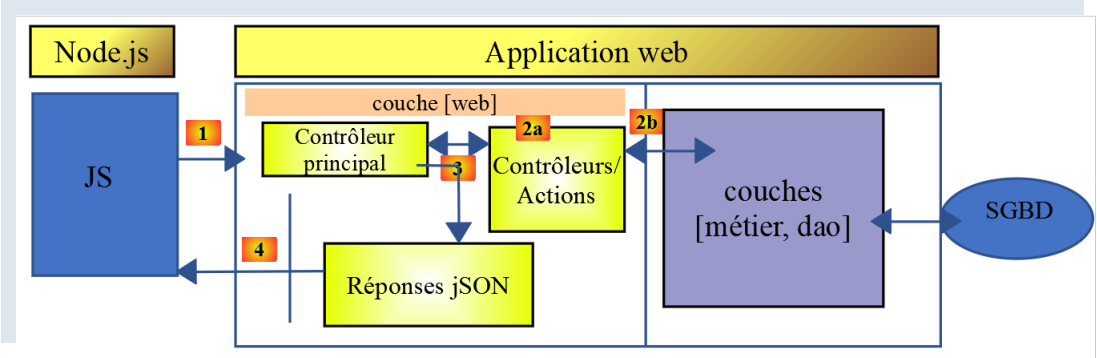
- en **[1-4]**, utiliser la requête **[init-session-700]** pour initialiser une session jSON ;
- en **[4-5]**, mettre **[version-14]** au lieu de **[version-12]** pour tester la version 14 du projet ;
- à l'exécution on doit recevoir la réponse jS0N **[6]** du serveur ;

La version 14 du serveur est désormais opérationnelle. Nous serons amenés à la modifier légèrement. Rappelons l'API de ce serveur :

Action	Rôle	Contexte d'exécution
init-session	Sert à fixer le type (json, xml, html) des réponses souhaitées	Requête **GET main.php?action=init-session&type=x** peut être émise à tout moment
authentifier-utilisateur	Autorise ou non un utilisateur à se connecter	Requête **POST main.php?action=authentifier-utilisateur** La requête doit avoir deux paramètres postés **[user, password]** Ne peut être émise que si le type de la session (json, xml, html) est connu
calculer-impot	Fait une simulation de calcul d'impôt	Requête **POST main.php?action=calculer-impot** La requête doit avoir trois paramètres postés **[marié, enfants, salaire]** Ne peut être émise que si le type de la session (json, xml, html) est connu et l'utilisateur authentifié
lister-simulations	Demande à voir la liste des simulations opérées depuis le début de la session	Requête **GET main.php?action=lister-simulations** La requête n'accepte aucun autre paramètre Ne peut être émise que si le type de la session (json, xml, html) est connu et l'utilisateur authentifié
supprimer-simulation	Supprime une simulation de la liste des simulations	Requête **GET main.php?action=lister-simulations&numéro=x** La requête n'accepte aucun autre paramètre Ne peut être émise que si le type de la session (json, xml, html) est connu et l'utilisateur authentifié
fin-session	Termine la session de simulations.	Techniquement l'ancienne session web est supprimée et une nouvelle session est créée Ne peut être émise que si le type de la session (json, xml, html) est connu et l'utilisateur authentifié

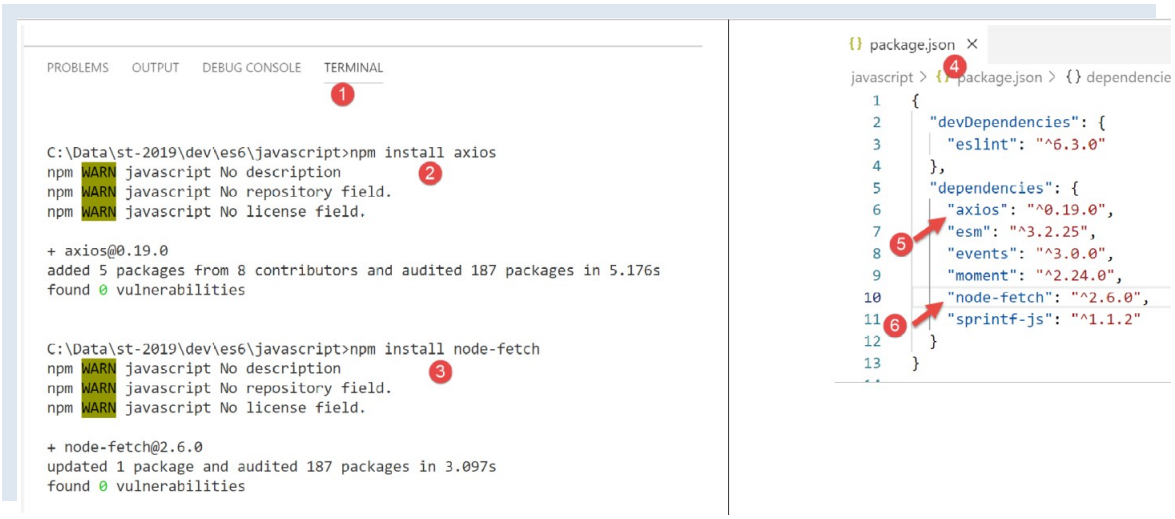
12.2.2 Installation des bibliothèques HTTP du client Javascript

Dans un premier temps, nous travaillerons avec l'architecture suivante :

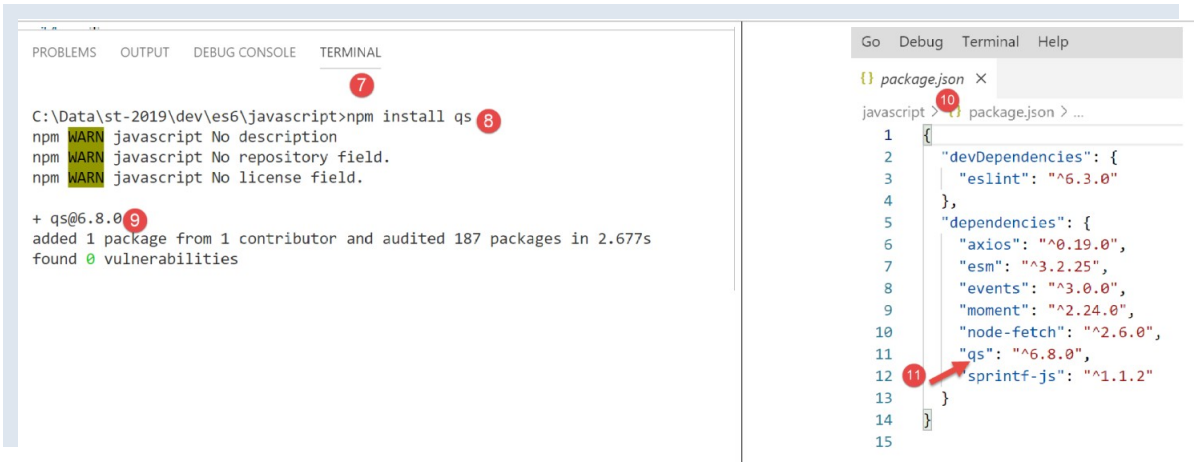


- en **[1]**, un script console **[node.js]** fait une requête HTTP vers le serveur jSON du calcul de l'impôt ;
- en **[4]**, il reçoit cette réponse et l'affiche sur la console ;

Dans l'exemple n° 1, nous utiliserons les bibliothèques **[node-fetch]** et **[axios]** puis nous ne conserverons qu'**[axios]** pour les exemples suivants. Nous installons maintenant ces deux bibliothèques Javascript à partir du terminal de **[VSCode]** :



Nous utiliserons également la bibliothèque **[qs]** qui permet l'encodage URL d'une chaîne de caractères. On se rappelle que cet encodage est utilisé pour encoder les paramètres d'une requête HTTP GET ou POST.



12.3 script [fetch-01]

Le script **[fetch-01]** utilise la bibliothèque **[node-fetch]** pour initialiser une session jSON avec le serveur de calcul d'impôt. Son code est le suivant :

```
1.  'use strict';
2.
3.  // imports
4.  import fetch from 'node-fetch';
5.  import qs from 'qs';
6.  import { sprintf } from 'sprintf-js';
7.  import moment from 'moment';
8.
9.
10. // URL de base du serveur de calcul d'impôt
11. const baseUrl = 'http://localhost/php7/scripts-web/impots/version-14/main.php?';
12. // init session
13. async function initSession() {
14.   // options de la requête HHTP [get /main.php?action=init-session&type=json]
15.   const options = {
16.     method: "GET",
17.     timeout: 2000
18.   };
19.   // exécution de la requête HTTP [get /main.php?action=init-session&type=json]
20.   let débutFetch;
21.   try {
22.     // requête asynchrone - [fetch] rend une promesse
23.     débutFetch = moment(Date.now());
24.     const response = await fetch(baseUrl + qs.stringify({
25.       action: 'init-session',
26.       type: 'json'
27.     }), options);
28.     // [response] est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-
    même)
29.     // on affiche cette réponse pour voir sa structure
30.     console.log(sprintf("réponse fetch formatée en json,=%j, %s", response,
    heure(débutFetch)));
31.     console.log("réponse fetch en javascript=", response);
32.     // on peut avoir aux entêtes HTTP
33.     console.log("entêtes de la réponse=", response.headers);
34.     // si réponse de type application / json, la réponse json du serveur est obtenue avec la
    fonction asynchrone [response.json()]
35.     // dans ce cas le code appelant obtient un objet [Promise]
36.     // [await] permet d'obtenir la réponse [json] du serveur plutôt que sa promesse
37.     const débutJson = moment(Date.now());
38.     const objet = await response.json();
39.     console.log(sprintf("réponse json=%j, type=%s, %s", objet, typeof (objet),
    heure(débutJson)));
40.     return objet;
41.     // si réponse de type text / plain, la réponse texte du serveur est obtenue avec
    [response.text()]
42.     // dans ce cas le code appelant obtient un objet [Promise]
43.     // [await] permet d'obtenir la réponse [texte] du serveur plutôt que sa promesse
44.     // const text = await response.text();
45.     // console.log("réponse texte=", text);
46.     // return text;
47.   } catch (error) {
48.     // on est là parce que le serveur a envoyé un code d'erreur [404 Not Found, ...]
    accompagné d'un corps vide - on affiche l'erreur pour voir sa structure
49.     // ou bien parce que le client [fetch] a lancé une exception (réseau inaccesible, ...)
50.     // on affiche la structure de l'erreur
51.     console.log(sprintf("error fetch en json=%j, %s", error, heure(débutFetch)));
52.     console.log("error fetch en javascript=", typeof (error), error);
53.     // on lance le msg d'erreur reçu
54.     throw error.message;
55.   }
56. }
57.
58. // la fonction main exécute la fonction asynchrone [initSession]
59. async function main() {
```

```
60.   try {
61.     console.log("requête HTTP vers le serveur en cours
    ------------------------------------------");
62.     const response = await initSession();
63.     console.log("succès ------------------------------------------");
64.     console.log("réponse=", response, typeof (response))
65.   } catch (error) {
66.     console.log("erreur ------------------------------------------");
67.     console.log("erreur=", error, typeof (error));
68.   }
69. }
70.
71. // test
72. main();
73.
74. // utilitaire d'affichage heure et durée
75. function heure(début) {
76.   // heure du moment courant
77.   const now = moment(Date.now());
78.   // formatage heure
79.   let result = "heure=" + now.format("HH:mm:ss:SSS");
80.   // faut-il calculer une durée ?
81.   if (début) {
82.     const durée = now - début;
83.     const milliseconds = durée % 1000;
84.     const seconds = Math.floor(durée / 1000);
85.     // formatage heure + durée
86.     result = result + sprintf(", durée= %s seconde(s) et %s millisecondes", seconds,
    milliseconds);
87.   }
88.   // résultat
89.   return result;
90. }
```

Commentaires

- les fonctions HTTP du Javascript sont des fonctions asynchrones. Nous utilisons ici ce que nous avons appris dans la section précédente (cf. lien) ;
- ligne 24 : pour attendre que la réponse de la fonction asynchrone **[fetch]** soit publiée sur la boucle événementielle de **[node.js]**, nous utilisons le mot clé **[await]**. Nous savons qu'alors que cette instruction doit être dans un code préfixé par le mot clé **[async]** (ligne 13) ;
- lignes 13-56 : nous encapsulons le code HTTP dans la fonction asynchrone **[initSession]** ;
- lignes 59-69 : une seconde fonction asynchrone **[main]** est utilisée pour appeler de façon bloquante (async / await) la fonction asynchrone **[initSession]** ;
- ligne 72 : la fonction asynchrone **[main]** est appelée ;
- bien que l'ensemble du code ressemble à du code synchrone, ce sont bien des fonctions asynchrones qui sont exécutées, mais de façon bloquante ;
- ligne 19 : pour initialiser une session jSON avec le serveur de calcul d'impôt, il faut lui envoyer la commande HTTP **[get /main.php?action=init-session&type=json]**. C'est ce que fait le code des lignes 24-27. La syntaxe de **[fetch]** est la suivante **[fetch(URL, options)]** avec :
  - **[URL]** : l'URL interrogée ;
  - **[options]** : un objet définissant les options de la requête. C'est là notamment qu'on définit les entêtes HTTP qu'on veut envoyer à la machine cible ;
- lignes 15-18 : on définit les options de la requête qu'on veut faire :
  - **[method]** : on veut faire un GET ;
  - **[timeout]** : on veut que le client **[fetch]** n'attende pas plus de 2 secondes la réponse du serveur Si ce délai est dépassé, **[fetch]** lancera une exception ;
- ligne 24 : pour obtenir l'URL **[/main.php?action=init-session&type=json]**, on utilise la bibliohèque **[qs]** pour obtenir l'encodage URL des paramètres **[action,type]** du GET. La chaîne obtenue est **[init-session&type=json]** qu'on aurait pu construire nous-mêmes. On voulait simplement montrer comment obtenir une chaîne URL encodée ;
- ligne 24 : le mot clé **[await]** montre que c'est une tâche asynchrone qui est lancée ici et qu'on attend qu'elle publie sa réponse sur la boucle événementielle de **[node.js]** ;
- ligne 24 : dans **[response]**, on obtient un objet complexe qui décrit la totalité de la réponse HTTP reçue (entêtes et document) ;
- lignes 30-31 : on affiche l'objet **[response]** pour voir sa structure, d'abord comme chaîne de caractères puis comme objet Javascript ;

- ligne 33 : on affiche les entêtes HTTP envoyés par le serveur ;
- ligne 38 : on sait que le serveur de calcul d'impôt va envoyer une chaîne jSON. Celle-ci est encapsulée dans l'objet `[response]`. On peut l'obtenir avec la méthode `[response.json()]`. Cependant cette méthode est asynchrone. On écrit donc `[await response.json()]` pour obtenir la chaîne jSON qui va être publiée sur la boucle événementielle de `[node.js]`. En fait ce n'est pas la chaîne jSON qu'on obtient mais l'objet Javascript représenté par celle-ci ;
- ligne 39 : affichage de la chaîne jSON reçue ;
- ligne 40 : on rend l'objet Javascript reçu ;
- ligne 47 : on intercepte une erreur éventuelle de l'instruction `[fetch]`. Celle-ci ne lance une exception que si l'opération HTTP n'a pu aboutir et qu'aucune réponse du serveur n'a été reçue. Si une réponse a été reçue, même avec un code HTTP différent de `[200 OK]`, `[fetch]` ne lance pas d'exception et la réponse du serveur sera disponible ligne 38 ;
- lignes 51-52 : on affiche l'objet `[error]` reçu par la clause `[catch]`, d'abord comme une chaîne jSON puis comme un objet Javascript ;
- ligne 54 : le message d'erreur de `[fetch]` se trouve dans `[error.message]` ;
- lignes 59-69 : la fonction asynchrone `[main]` lance la fonction asynchrone `[initSession]` de façon bloquante (await ligne 62) ;
- ligne 72 : la fonction asynchrone `[main]` est lancée et le code principal du script est alors terminé. Le script global lui sera terminé lorsque les tâches asynchrones lancées auront publié leurs résultats sur la boucle événementielle ;

Les résultats de l'exécution sont les suivants :

Cas 1 : le serveur Laragon n'est pas lancé

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\http\fetch-01.js"
2.  requête HTTP vers le serveur en cours ---------------------------------------------
3.  error fetch en json={"message":"network timeout at: http://localhost/php7/scripts-web/impots/
    version-14/main.php?action=init-session&type=json","type":"request-timeout"}, heure=10:08:48:1
    80, durée= 2 seconde(s) et 62 millisecondes
4.  error fetch en javascript= object { FetchError: network timeout at: http://localhost/php7/
    scripts-web/impots/version-14/main.php?action=init-session&type=json
5.      at Timeout.<anonymous> (c:\Data\st-2019\dev\es6\javascript\node_modules\node-
    fetch\lib\index.js:1448:13)
6.      at ontimeout (timers.js:436:11)
7.      at tryOnTimeout (timers.js:300:5)
8.      at listOnTimeout (timers.js:263:5)
9.      at Timer.processTimers (timers.js:223:10)
10.   message:
11.    'network timeout at: http://localhost/php7/scripts-web/impots/version-14/main.php?
    action=init-session&type=json',
12.   type: 'request-timeout' }
13. erreur ---------------------------------------------
14. erreur= network timeout at: http://localhost/php7/scripts-web/impots/version-14/main.php?
    action=init-session&type=json string
15.
16. [Done] exited with code=0 in 2.804 seconds
```

Commentaires

- ligne 3 : la requête HTTP échoue au bout de 2 secondes et 62 millisecondes à cause du timeout de 2 secondes qu'on avait imposé à la requête HTTP ;
- lignes 4-9 : l'objet Javascript `[error]` intercepté par la clause `[catch(error)]`. Cet objet a deux propriétés :
  - `[FetchError]` : ligne 4 ;
  - `[message]` : lignes 10-12 ;
- ligne 14 : le message d'erreur reçu par la fonction asynchrone `[main]` ;

Cas 2 : le serveur Laragon est lancé

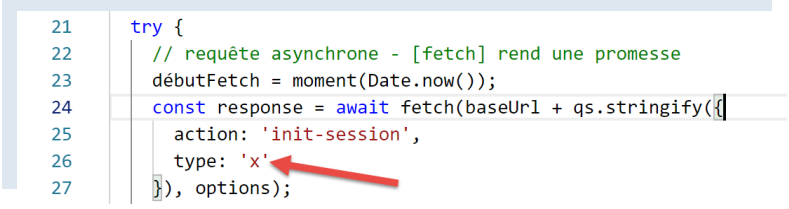
```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\http\fetch-01.js"
2.  requête HTTP vers le serveur en cours ---------------------------------------------
3.  réponse fetch formatée en json,={"size":0,"timeout":2000}, heure=10:13:50:814, durée= 0 seconde
    (s) et 375 millisecondes
4.  réponse fetch en javascript= Response {
5.    size: 0,
```

```
6.    timeout: 2000,
7.    [Symbol(Body internals)]:
8.     { body:
9.        PassThrough {
10.         _readableState: [ReadableState],
11.         readable: true,
12.         domain: null,
13.         _events: [Object],
14.         _eventsCount: 2,
15.         _maxListeners: undefined,
16.         _writableState: [WritableState],
17.         writable: false,
18.         allowHalfOpen: true,
19.         _transformState: [Object] },
20.      disturbed: false,
21.      error: null },
22.   [Symbol(Response internals)]:
23.    { url:
24.       'http://localhost/php7/scripts-web/impots/version-14/main.php?action=init-
    session&type=json',
25.      status: 200,
26.      statusText: 'OK',
27.      headers: Headers { [Symbol(map)]: [Object] },
28.      counter: 0 } }
29. entêtes de la réponse= Headers {
30.   [Symbol(map)]:
31.    [Object: null prototype] {
32.      date: [ 'Sat, 14 Sep 2019 08:13:50 GMT' ],
33.      server: [ 'Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11' ],
34.      'x-powered-by': [ 'PHP/7.2.11' ],
35.      'cache-control': [ 'max-age=0, private, must-revalidate, no-cache, private' ],
36.      'set-cookie': [ 'PHPSESSID=99q2iinusmhl55fa600aie2mmu; path=/' ],
37.      'content-length': [ '86' ],
38.      connection: [ 'close' ],
39.      'content-type': [ 'application/json' ] } }
40. réponse json={"action":"init-session","état":700,"réponse":"session démarrée avec type [json
    ]"}, type=object, heure=10:13:50:825, durée= 0 seconde(s) et 1 millisecondes
41. succès ------------------------------------------
42. réponse= { action: 'init-session',
43.   'état': 700,
44.   'réponse': 'session démarrée avec type [json]' } object
45.
46. [Done] exited with code=0 in 1.022 seconds
```

Commentaires

- ligne 3 : `[fetch]` reçoit la réponse du serveur au bout de 375 ms ;
- lignes 4-39 : la structure de l'objet Javascript `[response]` encapsulant la réponse du serveur. Parmi ses propriétés, certaines peuvent nous intéresser :
  - `[status]` (ligne 25) : code HTTP de la réponse du serveur ;
  - `[statusText]` (ligne 26) : texte associé à ce code ;
  - `[headers]` (ligne 27) : les entêtes HTTP de la réponse du serveur ;
  - `[body]` (ligne 8) : représente le document envoyé par le serveur. L'instruction `[fetch]` offre des méthodes pour l'exploiter ;
- lignes 29-39 : les entêtes HTTP de la réponse du serveur ;
- ligne 40 : la fonction asynchrone `[response.json()]` a publié sa réponse au bout d'1 milliseconde ;
- lignes 42-44 : l'objet Javascript reçu par la fonction asynchrone `[main]` ;

Cas 3 : le serveur Laragon est lancé mais on lui envoie une commande erronée :



- ci-dessus, ligne 26, on passe un type erroné de session au serveur ;

Les résultats de l'exécution sont les suivants :

```
1.  requête HTTP vers le serveur en cours ------------------------------------------
2.  réponse fetch formatée en json,={"size":0,"timeout":2000}, heure=10:27:54:114, durée= 0 seconde
    (s) et 136 millisecondes
3.  réponse fetch en javascript= Response {
4.    size: 0,
5.    timeout: 2000,
6.    [Symbol(Body internals)]:
7.     { body:
8.        PassThrough {
9.          _readableState: [ReadableState],
10.         readable: true,
11.         domain: null,
12.         _events: [Object],
13.         _eventsCount: 2,
14.         _maxListeners: undefined,
15.         _writableState: [WritableState],
16.         writable: false,
17.         allowHalfOpen: true,
18.         _transformState: [Object] },
19.      disturbed: false,
20.      error: null },
21.   [Symbol(Response internals)]:
22.    { url:
23.       'http://localhost/php7/scripts-web/impots/version-14/main.php?action=init-
    session&type=x',
24.      status: 400,
25.      statusText: 'Bad Request',
26.      headers: Headers { [Symbol(map)]: [Object] },
27.      counter: 0 } }
28. entêtes de la réponse= Headers {
29.   [Symbol(map)]:
30.    [Object: null prototype] {
31.      date: [ 'Sat, 14 Sep 2019 08:27:54 GMT' ],
32.      server: [ 'Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11' ],
33.      'x-powered-by': [ 'PHP/7.2.11' ],
34.      'cache-control': [ 'max-age=0, private, must-revalidate, no-cache, private' ],
35.      'set-cookie': [ 'PHPSESSID=5ku9gfok81ikj98hia0meeum57; path=/' ],
36.      'content-length': [ '79' ],
37.      connection: [ 'close' ],
38.      'content-type': [ 'application/json' ] } }
39. réponse json={"action":"init-session","état":703,"réponse":"paramètre type=[x] invalide"}, type
    =object, heure=10:27:54:127, durée= 0 seconde(s) et 2 millisecondes
40. succès -------------------------------------------
41. réponse= { action: 'init-session',
42.   'état': 703,
43.   'réponse': 'paramètre type=[x] invalide' } object
44.
45. [Done] exited with code=0 in 0.712 seconds
```

- la réponse du serveur est reçue ligne 2 ;
- ligne 24 : on peut voir que le code HTTP de la réponse du serveur est 400, un code d'erreur. Néanmoins, `[fetch]` n'a pas lancé d'exception. Tant que `[fetch]` reçoit une réponse du serveur, il l'exploite et ne lance pas d'exception ;
- lignes 41-43 : la réponse obtenue par la fonction asynchrone `[main]` ;

12.4 script [fetch-02]

Le script suivant reprend le script `[fetch-01]` en le débarrassant de tous les détails inutiles :

```
1.  'use strict';
2.
3.  // imports
4.  import fetch from 'node-fetch';
5.  import qs from 'qs';
6.
7.  // URL de base du serveur de calcul d'impôt
8.  const baseUrl = 'http://localhost/php7/scripts-web/impots/version-14/main.php?';
```

```
// init session
async function initSession() {
  // options de la requête HHTP [get /main.php?action=init-session&type=json]
  const options = {
    method: "GET",
    timeout: 2000
  };
  // exécution de la requête HTTP [get /main.php?action=init-session&type=json]
  const response = await fetch(baseUrl + qs.stringify({
    action: 'init-session',
    type: 'json'
  }), options);
  // résultat reçu en jSON
  return await response.json();
}

// la fonction main exécute la fonction asynchrone [initSession]
async function main() {
  try {
    console.log("requête HTTP vers le serveur en cours
----------------------------------------------");
    const response = await initSession();
    console.log("succès ----------------------------------------------");
    console.log("réponse=", response)
  } catch (error) {
    console.log("erreur ----------------------------------------------");
    console.log("erreur=", error.message);
  }
}

// test
main();
```

Les résultats d'une exécution normale :

```
[Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
\Data\st-2019\dev\es6\javascript\http\fetch-02.js"
requête HTTP vers le serveur en cours ----------------------------------------------
succès ----------------------------------------------
réponse= { action: 'init-session',
  'état': 700,
  'réponse': 'session démarrée avec type [json]' }

[Done] exited with code=0 in 0.56 seconds
```

Les résultats d'une exécution avec exception (on arrête le serveur Laragon) :

```
[Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
\Data\st-2019\dev\es6\javascript\http\fetch-02.js"
requête HTTP vers le serveur en cours ----------------------------------------------
erreur ----------------------------------------------
erreur= network timeout at: http://localhost/php7/scripts-web/impots/version-14/main.php?
action=init-session&type=json

[Done] exited with code=0 in 2.701 seconds
```

12.5 script [axios-01]

On reprend ici le script **[fetch-01]** que l'on réécrit avec la bibliothèque **[axios]**. On rappelle que notre intérêt pour cette bibliothèque est qu'elle soit portable entre l'environnement **[node.js]** et ceux des navigateurs usuels. Cela permet :

- dans une phase 1, de tester nos scripts dans un environnement **[node.js]** ;
- dans une phase 2, de les porter sur un navigateur ;

Le script **[axios-01]** reprend la structure du script **[fetch-01]** :

```
'use strict';
import axios from 'axios';

// configuration par défaut d'axios
axios.defaults.timeout = 2000;
axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';
```

```
7.
8.  // init session
9.  async function initSession(axios) {
10.   // options de la requête HHTP [get /main.php?action=init-session&type=json]
11.   const options = {
12.     method: "GET",
13.     // paramètres de l'URL
14.     params: {
15.       action: 'init-session',
16.       type: 'json'
17.     }
18.   };
19.   // exécution de la requête HTTP [get /main.php?action=init-session&type=json]
20.   try {
21.     // requête asynchrone
22.     const response = await axios.request('main.php', options);
23.     // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-
    même)
24.     // on affiche cette réponse pour voir sa structure
25.     console.log("réponse axios=", response);
26.     // la réponse du serveur est dans [response.data]
27.     return response.data;
28.   } catch (error) {
29.     // on est là parce que le serveur a envoyé un code d'erreur [404 Not Found, 500 Internal
    Server Error, ...]
30.     // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
31.     // on l'affiche pour voir sa structure
32.     console.log("axios error=", typeof (error), error);
33.     if (error.response) {
34.       // le serveur a signalé une erreur dans le statut HTTP mais il a aussi envoyé une
    réponse
35.       // alors celle-ci est trouvée dans [error.response.data]
36.       // on sait que le serveur envoie des réponses jSON de structure {action, état, réponse}
37.       // et qu'en cas d'erreur, le msg d'erreur est dans [réponse]
38.       return error.response.data;
39.     } else {
40.       // on lance l'erreur
41.       throw error;
42.     }
43.   }
44. }
45.
46. // la fonction main exécute la fonction asynchrone [initSession]
47. async function main() {
48.   try {
49.     console.log("requête HTTP vers le serveur en cours
    ----------------------------------------------");
50.     const response = await initSession(axios);
51.     console.log("succès ----------------------------------------------");
52.     console.log("réponse=", response, typeof (response))
53.   } catch (error) {
54.     console.log("erreur ----------------------------------------------");
55.     console.log("erreur=", error.message);
56.   }
57. }
58.
59. // test
60. main();
```

Commentaires

- ligne 2 : on importe la bibliothèque **[axios]** ;
- lignes 5-6 : configuration par défaut des requêtes HTTP. Les options **[axios.defaults]** sont valables pour toutes les requêtes HTTP émises par l'objet **[axios]** sans qu'on ait besoin de les rappeler à chaque nouvelle requête ;
- ligne 5 : timeout de 2 secondes pour toutes les requêtes ;
- ligne 6 : toutes les URL seront exprimées relativement à l'URL de base ;
- ligne 9 : la fonction asynchrone **[initSession]** ;
- lignes 11-18 : les options de la requête HTTP qui va être émise (en plus des options par défaut déjà définies aux lignes 5-6) ;
- lignes 14-17 : les paramètres de l'URL **[action=init-session&type=json]**. L'objet **[params]** sera automatiquement transformée en chaîne de caractères URL encodée ;

- ligne 22 : appel bloquant de la fonction asynchrone **[axios.request]**. Le 1er paramètre est l'URL cible construite comme **[main.php]** ajouté à l'URL de base définie ligne 6. Le second paramètre est l'objet **[options]** des lignes 11-18 ;
- ligne 25 : **[response]** est un objet Javascript encapsulant la totalité de la réponse HTTP du serveur (entêtes HTTP + document réponse). On l'affiche pour voir sa structure Javascript ;
- ligne 27 : si le serveur a envoyé un document, alors il est trouvé dans **[response.data]**. Ici nous savons que le serveur envoie une réponse jSON accompagnée de l'entête HTTP **[Content-type : application/json]**. La présence de cet entête fait que **[axios]** désérialise automatiquement **[response.data]** en un objet Javascript ;
- ligne 28 : la fonction **[axios]** peut lancer une exception. C'est là que **[axios]** diffère de **[fetch]**. Une exception est lancée dans les cas suivants :
  - la requête HTTP n'a pas pu être émise (erreur côté client) ;
  - le serveur a envoyé un code HTTP d'erreur (400, 404, 500, …) (ce point est en fait configurable). On rappelle que si ce code HTTP est accompagné d'une réponse, **[fetch]** ne lançait pas d'exception alors qu'**[axios]** en lance une. Néanmoins, si le code HTTP d'erreur est accompagné d'un document, celui-ci est mis dans **[error.response]** ;
- ligne 32 : on affiche la structure Javascript de l'objet **[error]** ;
- lignes 33-38 : si l'objet **[error]** contient un objet **[response]** alors c'est cette réponse qu'on rend au code appelant ;
- lignes 39-42 : dans les autres cas, on remonte l'objet **[error]** au code appelant ;
- lignes 47-57 : la fonction asynchrone **[main]** ;
- ligne 50 : appel bloquant à la fonction asynchrone **[initSession]**. On récupère la réponse jSON du serveur comme un objet Javascript ;
- lignes 53-56 : interception de l'erreur éventuelle. Le message d'erreur est dans **[error.message]** ;

Les résultats de l'exécution sont les suivants :

Cas 1 : le serveur Laragon n'est pas lancé

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\http\axios-01.js"
2.  requête HTTP vers le serveur en cours ---------------------------------------------
3.  axios error= object { Error: timeout of 2000ms exceeded
4.      at createError (c:
    \Data\st-2019\dev\es6\javascript\node_modules\axios\lib\core\createError.js:16:15)
5.      at Timeout.handleRequestTimeout (c:
    \Data\st-2019\dev\es6\javascript\node_modules\axios\lib\adapters\http.js:252:16)
6.      at ontimeout (timers.js:436:11)
7.      at tryOnTimeout (timers.js:300:5)
8.      at listOnTimeout (timers.js:263:5)
9.      at Timer.processTimers (timers.js:223:10)
10.   config:
11.    { url:
12.       'http://localhost/php7/scripts-web/impots/version-14/main.php',
13.      method: 'get',
14.      params: { action: 'init-session', type: 'json' },
15.      headers:
16.       { Accept: 'application/json, text/plain, */*',
17.         'User-Agent': 'axios/0.19.0' },
18.      baseURL: 'http://localhost/php7/scripts-web/impots/version-14',
19.      transformRequest: [ [Function: transformRequest] ],
20.      transformResponse: [ [Function: transformResponse] ],
21.      timeout: 2000,
22.      adapter: [Function: httpAdapter],
23.      xsrfCookieName: 'XSRF-TOKEN',
24.      xsrfHeaderName: 'X-XSRF-TOKEN',
25.      maxContentLength: -1,
26.      validateStatus: [Function: validateStatus],
27.      data: undefined },
28.   code: 'ECONNABORTED',
29.   request:
30.    Writable {
31.      _writableState:
32.       WritableState {
33.         objectMode: false,
34.         highWaterMark: 16384,
35.         finalCalled: false,
36.         needDrain: false,
37.         ending: false,
38.         ended: false,
```

```
39.         finished: false,
40.         destroyed: false,
41.         decodeStrings: true,
42.         defaultEncoding: 'utf8',
43.         length: 0,
44.         writing: false,
45.         corked: 0,
46.         sync: true,
47.         bufferProcessing: false,
48.         onwrite: [Function: bound onwrite],
49.         writecb: null,
50.         writelen: 0,
51.         bufferedRequest: null,
52.         lastBufferedRequest: null,
53.         pendingcb: 0,
54.         prefinished: false,
55.         errorEmitted: false,
56.         emitClose: true,
57.         bufferedRequestCount: 0,
58.         corkedRequestsFree: [Object] },
59.       writable: true,
60.       domain: null,
61.       _events:
62.        [Object: null prototype] {
63.         response: [Function: handleResponse],
64.         error: [Function: handleRequestError] },
65.       _eventsCount: 2,
66.       _maxListeners: undefined,
67.       _options:
68.        { protocol: 'http:',
69.         maxRedirects: 21,
70.         maxBodyLength: 10485760,
71.         path:
72.          '/php7/scripts-web/impots/version-14/main.php?action=init-session&type=json',
73.         method: 'GET',
74.         headers: [Object],
75.         agent: undefined,
76.         auth: undefined,
77.         hostname: 'localhost',
78.         port: null,
79.         nativeProtocols: [Object],
80.         pathname: '/php7/scripts-web/impots/version-14/main.php',
81.         search: '?action=init-session&type=json' },
82.       _redirectCount: 0,
83.       _redirects: [],
84.       _requestBodyLength: 0,
85.       _requestBodyBuffers: [],
86.       _onNativeResponse: [Function],
87.       _currentRequest:
88.        ClientRequest {
89.         domain: null,
90.         _events: [Object],
91.         _eventsCount: 6,
92.         _maxListeners: undefined,
93.         output: [],
94.         outputEncodings: [],
95.         outputCallbacks: [],
96.         outputSize: 0,
97.         writable: true,
98.         _last: true,
99.         chunkedEncoding: false,
100.        shouldKeepAlive: false,
101.        useChunkedEncodingByDefault: false,
102.        sendDate: false,
103.        _removedConnection: false,
104.        _removedContLen: false,
105.        _removedTE: false,
106.        _contentLength: 0,
107.        _hasBody: true,
108.        _trailer: '',
109.        finished: true,
110.        _headerSent: true,
111.        socket: [Socket],
112.        connection: [Socket],
```

```
113.        _header:
114.         'GET /php7/scripts-web/impots/version-14/main.php?action=init-session&type=json HTTP/
    1.1\r\nAccept: application/json, text/plain, */*\r\nUser-Agent: axios/0.19.0\r\nHost: localhost
    \r\nConnection: close\r\n\r\n',
115.        _onPendingData: [Function: noopPendingOutput],
116.        agent: [Agent],
117.        socketPath: undefined,
118.        timeout: undefined,
119.        method: 'GET',
120.        path:
121.         '/php7/scripts-web/impots/version-14/main.php?action=init-session&type=json',
122.        _ended: false,
123.        res: null,
124.        aborted: 1568528450762,
125.        timeoutCb: null,
126.        upgradeOrConnect: false,
127.        parser: [HTTPParser],
128.        maxHeadersCount: null,
129.        _redirectable: [Circular],
130.        [Symbol(isCorked)]: false,
131.        [Symbol(outHeadersKey)]: [Object] },
132.     _currentUrl:
133.      'http://localhost/php7/scripts-web/impots/version-14/main.php?action=init-
    session&type=json' },
134.  response: undefined,
135.  isAxiosError: true,
136.  toJSON: [Function] }
137.erreur --------------------------------------------
138.erreur= timeout of 2000ms exceeded
139.
140.[Done] exited with code=0 in 2.784 seconds
```

Commentaires

- lignes 33-136 : l'objet `[error]` contient de nombreuses informations ;
  - `[Error]`, lignes 3-9 : une description de l'erreur qui s'est produite ;
  - `[config]`, lignes 10-27 : la configuration de la requête HTTP qui a mené à cette erreur ;
  - `[config.url]`, lignes 11-12 : l'URL cible ;
  - `[config.method]`, ligne 13 : méthode de la requête ;
  - `[config.params]`, ligne 14 : les paramètres de l'URL ;
  - `[config.headers]`, lignes 16-17 : les entêtes HTTP de la requête ;
  - `[config.baseURL]`, ligne 18 : l'URL de base de l'URL cible ;
  - `[config.timeout]`, ligne 21 : le timeout de la requête ;
  - `[code]`, ligne 28 : un code d'erreur ;
  - `[request]`, lignes 29-133 : une description détaillée de la requête HTTP. On remarquera que la plupart des propriétés sont préfixées par l'underscore _ montrant par là que ce sont des propriétés internes à l'objet `[request]` pas destinées à être exploitées directement par le développeur ;
  - `[response]`, ligne 134 : la réponse du serveur, ici inexistante ;
- ligne 138 : le message d'erreur affiché par la fonction `[main]` ;

Cas 2 : le serveur Laragon est lancé

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\http\axios-01.js"
2.  requête HTTP vers le serveur en cours ----------------------------------------------
3.  réponse axios= { status: 200,
4.    statusText: 'OK',
5.    headers:
6.     { date: 'Sun, 15 Sep 2019 07:09:26 GMT',
7.       server: 'Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11',
8.       'x-powered-by': 'PHP/7.2.11',
9.       'cache-control': 'max-age=0, private, must-revalidate, no-cache, private',
10.      'set-cookie': [ 'PHPSESSID=uas6lugtblstktcifpd8e5irm6; path=/' ],
11.      'content-length': '86',
12.      connection: 'close',
13.      'content-type': 'application/json' },
14.   config:
15.    { url:
16.       'http://localhost/php7/scripts-web/impots/version-14/main.php',
17.      method: 'get',
```

```
18.       params: { action: 'init-session', type: 'json' },
19.       headers:
20.        { Accept: 'application/json, text/plain, */*',
21.          'User-Agent': 'axios/0.19.0' },
22.       baseURL: 'http://localhost/php7/scripts-web/impots/version-14',
23.       transformRequest: [ [Function: transformRequest] ],
24.       transformResponse: [ [Function: transformResponse] ],
25.       timeout: 2000,
26.       adapter: [Function: httpAdapter],
27.       xsrfCookieName: 'XSRF-TOKEN',
28.       xsrfHeaderName: 'X-XSRF-TOKEN',
29.       maxContentLength: -1,
30.       validateStatus: [Function: validateStatus],
31.       data: undefined },
32.    request:
33.     ClientRequest {
34.       domain: null,
35.       _events:
36.        [Object: null prototype] {
37.          socket: [Function],
38.          abort: [Function],
39.          aborted: [Function],
40.          error: [Function],
41.          timeout: [Function],
42.          prefinish: [Function: requestOnPrefinish] },
43.       _eventsCount: 6,
44.       _maxListeners: undefined,
45.       output: [],
46.       outputEncodings: [],
47.       outputCallbacks: [],
48.       outputSize: 0,
49.       writable: true,
50.       _last: true,
51.       chunkedEncoding: false,
52.       shouldKeepAlive: false,
53.       useChunkedEncodingByDefault: false,
54.       sendDate: false,
55.       _removedConnection: false,
56.       _removedContLen: false,
57.       _removedTE: false,
58.       _contentLength: 0,
59.       _hasBody: true,
60.       _trailer: '',
61.       finished: true,
62.       _headerSent: true,
63.       socket:
64.        Socket {
65.          connecting: false,
66.          _hadError: false,
67.          _handle: [TCP],
68.          _parent: null,
69.          _host: 'localhost',
70.          _readableState: [ReadableState],
71.          readable: true,
72.          domain: null,
73.          _events: [Object],
74.          _eventsCount: 7,
75.          _maxListeners: undefined,
76.          _writableState: [WritableState],
77.          writable: false,
78.          allowHalfOpen: false,
79.          _sockname: null,
80.          _pendingData: null,
81.          _pendingEncoding: '',
82.          server: null,
83.          _server: null,
84.          parser: null,
85.          _httpMessage: [Circular],
86.          [Symbol(asyncId)]: 6,
87.          [Symbol(lastWriteQueueSize)]: 0,
88.          [Symbol(timeout)]: null,
89.          [Symbol(kBytesRead)]: 0,
90.          [Symbol(kBytesWritten)]: 0 },
91.       connection:
```

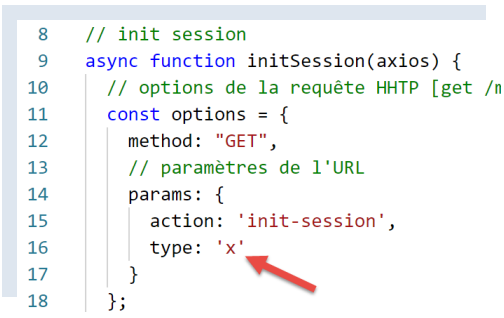
```
92.       Socket {
93.         connecting: false,
94.         _hadError: false,
95.         _handle: [TCP],
96.         _parent: null,
97.         _host: 'localhost',
98.         _readableState: [ReadableState],
99.         readable: true,
100.        domain: null,
101.        _events: [Object],
102.        _eventsCount: 7,
103.        _maxListeners: undefined,
104.        _writableState: [WritableState],
105.        writable: false,
106.        allowHalfOpen: false,
107.        _sockname: null,
108.        _pendingData: null,
109.        _pendingEncoding: '',
110.        server: null,
111.        _server: null,
112.        parser: null,
113.        _httpMessage: [Circular],
114.        [Symbol(asyncId)]: 6,
115.        [Symbol(lastWriteQueueSize)]: 0,
116.        [Symbol(timeout)]: null,
117.        [Symbol(kBytesRead)]: 0,
118.        [Symbol(kBytesWritten)]: 0 },
119.     _header:
120.      'GET /php7/scripts-web/impots/version-14/main.php?action=init-session&type=json HTTP/
    1.1\r\nAccept: application/json, text/plain, */*\r\nUser-Agent: axios/0.19.0\r\nHost: localhost
    \r\nConnection: close\r\n\r\n',
121.     _onPendingData: [Function: noopPendingOutput],
122.     agent:
123.      Agent {
124.        domain: null,
125.        _events: [Object],
126.        _eventsCount: 1,
127.        _maxListeners: undefined,
128.        defaultPort: 80,
129.        protocol: 'http:',
130.        options: [Object],
131.        requests: {},
132.        sockets: [Object],
133.        freeSockets: {},
134.        keepAliveMsecs: 1000,
135.        keepAlive: false,
136.        maxSockets: Infinity,
137.        maxFreeSockets: 256 },
138.     socketPath: undefined,
139.     timeout: undefined,
140.     method: 'GET',
141.     path:
142.      '/php7/scripts-web/impots/version-14/main.php?action=init-session&type=json',
143.     _ended: true,
144.     res:
145.      IncomingMessage {
146.        _readableState: [ReadableState],
147.        readable: false,
148.        domain: null,
149.        _events: [Object],
150.        _eventsCount: 3,
151.        _maxListeners: undefined,
152.        socket: [Socket],
153.        connection: [Socket],
154.        httpVersionMajor: 1,
155.        httpVersionMinor: 0,
156.        httpVersion: '1.0',
157.        complete: true,
158.        headers: [Object],
159.        rawHeaders: [Array],
160.        trailers: {},
161.        rawTrailers: [],
162.        aborted: false,
163.        upgrade: false,
```

```
164.        url: '',
165.        method: null,
166.        statusCode: 200,
167.        statusMessage: 'OK',
168.        client: [Socket],
169.        _consuming: false,
170.        _dumped: false,
171.        req: [Circular],
172.        responseUrl:
173.         'http://localhost/php7/scripts-web/impots/version-14/main.php?action=init-
    session&type=json',
174.        redirects: [] },
175.     aborted: undefined,
176.     timeoutCb: null,
177.     upgradeOrConnect: false,
178.     parser: null,
179.     maxHeadersCount: null,
180.     _redirectable:
181.      Writable {
182.        _writableState: [WritableState],
183.        writable: true,
184.        domain: null,
185.        _events: [Object],
186.        _eventsCount: 2,
187.        _maxListeners: undefined,
188.        _options: [Object],
189.        _redirectCount: 0,
190.        _redirects: [],
191.        _requestBodyLength: 0,
192.        _requestBodyBuffers: [],
193.        _onNativeResponse: [Function],
194.        _currentRequest: [Circular],
195.        _currentUrl:
196.         'http://localhost/php7/scripts-web/impots/version-14/main.php?action=init-
    session&type=json' },
197.     [Symbol(isCorked)]: false,
198.     [Symbol(outHeadersKey)]:
199.      [Object: null prototype] { accept: [Array], 'user-agent': [Array], host: [Array] } },
200.  data:
201.   { action: 'init-session',
202.     'état': 700,
203.     'réponse': 'session démarrée avec type [json]' } }
204.succès ------------------------------------------
205.réponse= { action: 'init-session',
206.  'état': 700,
207.  'réponse': 'session démarrée avec type [json]' } object
208.
209.[Done] exited with code=0 in 1.115 seconds
```

Commentaires

- lignes 3-203 : l'objet Javascript `[response]` qui encapsule la réponse HTTP du serveur ;
- ligne 3, `[status]` : le code HTTP de la réponse ;
- ligne 4, `[statusText]` : le texte associé au code HTTP précédent ;
- lignes 5-13, `[headers]` : les entêtes HTTP de la réponse :
  - ligne 10, `[Set-Cookie]` : le cookie de session ;
  - ligne 13, `[Content-Type]` : le type du document envoyé par le serveur ;
- lignes 14-31, `[config]` : la configuration de la requête HTTP émise ;
- lignes 32-199, `[request]` : l'objet Javascript détaillant la requête HTTP émise ;
- lignes 200-203, `[request.data]` : l'objet Javascript encapsulant la réponse jSON du serveur ;
- lignes 205-207 : la réponse récupérée par la fonction asynchrone `[main]` ;

Cas 3 : on fait une requête `[init-session]` erronée au serveur Laragon ;



```
8   // init session
9   async function initSession(axios) {
10    // options de la requête HHTP [get /m
11    const options = {
12      method: "GET",
13      // paramètres de l'URL
14      params: {
15        action: 'init-session',
16        type: 'x'
17      }
18    };
```

Les résultats de l'exécution sont les suivants :

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\http\axios-01.js"
2.  requête HTTP vers le serveur en cours ---------------------------------------------
3.  axios error= object { Error: Request failed with status code 400
4.     ...
5.    config:
6.     { url:
7.        'http://localhost/php7/scripts-web/impots/version-14/main.php',
8.       ...
9.       data: undefined },
10.   request:
11.    ...
12.      [Symbol(outHeadersKey)]:
13.       [Object: null prototype] { accept: [Array], 'user-agent': [Array], host: [Array] } },
14.   response:
15.    { status: 400,
16.      statusText: 'Bad Request',
17.      headers:
18.       { date: 'Sun, 15 Sep 2019 07:25:58 GMT',
19.         server: 'Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11',
20.         'x-powered-by': 'PHP/7.2.11',
21.         'cache-control': 'max-age=0, private, must-revalidate, no-cache, private',
22.         'set-cookie': [Array],
23.         'content-length': '79',
24.         connection: 'close',
25.         'content-type': 'application/json' },
26.      config:
27.       { url:
28.          'http://localhost/php7/scripts-web/impots/version-14/main.php',
29.         ...
30.         data: undefined },
31.      request:
32.       ...
33.         [Symbol(outHeadersKey)]: [Object] },
34.      data:
35.       { action: 'init-session',
36.         'état': 703,
37.         'réponse': 'paramètre type=[x] invalide' } },
38.   isAxiosError: true,
39.   toJSON: [Function] }
40. succès ----------------------------------------------
41. réponse= { action: 'init-session',
42.   'état': 703,
43.   'réponse': 'paramètre type=[x] invalide' } object
44.
45. [Done] exited with code=0 in 0.69 seconds
```

Parce que le serveur a répondu avec un code HTTP 400 (ligne 15), **[axios]** a lancé une exception (encore une fois ce comportement est configurable). Bien que **[axios]** ait lancé une exception, on obtient bien la réponse HTTP du serveur dans **[error.response]** (ligne 14) et le document jSON envoyé **[error.response.data]** (ligne 34). Lignes 41-43 : la fonction **[main]** récupère correctement la réponse jSON du serveur.

12.6 script [axios-02]

Maintenant que nous avons détaillé les objets manipulés par la bibliothèque **[axios]** lors d'une requête HTTP, on peut réécrire le script **[axios-01]** de la façon suivante :

```
1.  'use strict';
2.  import axios from 'axios';
3.
4.  // configuration par défaut d'axios
5.  axios.defaults.timeout = 2000;
6.  axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';
7.
8.  // init session
9.  async function initSession(axios) {
10.   // options de la requête HHTP [get /main.php?action=init-session&type=json]
11.   const options = {
12.     method: "GET",
13.     // paramètres de l'URL
14.     params: {
15.       action: 'init-session',
16.       type: 'json'
17.     }
18.   };
19.   try {
20.     // exécution de la requête HTTP [get /main.php?action=init-session&type=json]
21.     const response = await axios.request('main.php', options);
22.     // la réponse du serveur est dans [response.data]
23.     return response.data;
24.   } catch (error) {
25.     // réponse du serveur
26.     if (error.response) {
27.       // la réponse jSON est dans [error.response.data]
28.       return error.response.data;
29.     } else {
30.       // on relance l'erreur
31.       throw error;
32.     }
33.   }
34. }
35.
36. // la fonction main exécute la fonction asynchrone [initSession]
37. async function main() {
38.   try {
39.     console.log("requête HTTP vers le serveur en cours
    ------------------------------------------");
40.     const response = await initSession(axios);
41.     console.log("succès ------------------------------------------");
42.     console.log("réponse=", response, typeof (response))
43.   } catch (error) {
44.     console.log("erreur ------------------------------------------");
45.     console.log("erreur=", error.message);
46.   }
47. }
48.
49. // test
50. main();
```

12.7 script [axios-03]

Le script **[axios-03]** reprend la méthodogie du script **[axios-02]**. On ajoute cette fois la requête HTTP **[authentifier-utilisateur]** vers le serveur qui se fait à l'aide d'un POST :

```
1.  'use strict';
2.  import axios from 'axios';
3.  import qs from 'qs'
4.
5.  // configuration axios
6.  axios.defaults.timeout = 2000;
7.  axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';
8.
9.
10. // init session
```

```
11. async function initSession(axios) {
12.   // options de la requête HHTP [get /main.php?action=init-session&type=json]
13.   const options = {
14.     method: "GET",
15.     // paramètres de l'URL
16.     params: {
17.       action: 'init-session',
18.       type: 'json'
19.     }
20.   };
21.   try {
22.     // exécution de la requête HTTP [get /main.php?action=init-session&type=json]
23.     const response = await axios.request('main.php', options);
24.     // la réponse du serveur est dans [response.data]
25.     return response.data;
26.   } catch (error) {
27.     // réponse du serveur
28.     if (error.response) {
29.       // la réponse jSON est dans [error.response.data]
30.       return error.response.data;
31.     } else {
32.       // on relance l'erreur
33.       throw error;
34.     }
35.   }
36. }
37.
38. async function authentifierUtilisateur(axios, user, password) {
39.   // options de la requête HHTP [POST /main.php?action=authentifier-utilisateur]
40.   const options = {
41.     method: "POST",
42.     headers: {
43.       'Content-type': 'application/x-www-form-urlencoded',
44.     },
45.     // corps du POST
46.     data: qs.stringify({
47.       user: user,
48.       password: password
49.     }),
50.     // paramètres de l'URL
51.     params: {
52.       action: 'authentifier-utilisateur'
53.     }
54.   };
55.   try {
56.     // exécution de la requête HTTP [post /main.php?action=authentifier-utilisateur]
57.     const response = await axios.request('main.php', options);
58.     // la réponse du serveur est dans [response.data]
59.     return response.data;
60.   } catch (error) {
61.     // réponse du serveur
62.     if (error.response) {
63.       // la réponse jSON est dans [error.response.data]
64.       return error.response.data;
65.     } else {
66.       // on relance l'erreur
67.       throw error;
68.     }
69.   }
70. }
71.
72. // la fonction main exécute les fonctions asynchrones une par une
73. async function main() {
74.   try {
75.     // init-session
76.     console.log("action init-session en cours ------------------------------------------");
77.     const response1 = await initSession(axios);
78.     console.log("succès ------------------------------------------");
79.     console.log("réponse=", response1);
80.     // authentifier-utilisateur
81.     console.log("action authentifier-utilisateur en cours
    ------------------------------------------");
82.     const response2 = await authentifierUtilisateur(axios, 'admin', 'admin');
83.     console.log("succès ------------------------------------------");
```

```
84.     console.log("réponse=", response2)
85.   } catch (error) {
86.     console.log("erreur -------------------------------------------");
87.     console.log("erreur=", error);
88.   }
89. }
90.
91. // test
92. main();
```

Commentaires

- lignes 38-70 : la fonction asynchrone **[authentifierUtilisateur]** ;
- ligne 39 : il faut faire la requête **[POST /main.php?action=authentifier-utilisateur]** ;
- lignes 40-54 : les options de la requête HTTP ;
- ligne 41 : c'est un POST ;
- lignes 42-44 : les paramètres du POST seront URL encodés dans un document que le client envoie avec sa requête ;
- lignes 46-49 : la propriété **[data]** doit contenir la chaîne du POST URL encodée. Pour cela, on utilise ici la bibliothèque **[qs]** importée ligne 3 ;
- lignes 55-69 : pour l'exécution de la requête, on retrouve le même code que dans la méthode **[initSession]** ;
- lignes 73-89 : la méthode **[asynchrone]** appelle successivement, de manière bloquante, les méthodes **[initSession, authentifierUtilisateur]**, lignes 77 et 82 ;
- ligne 82 : on utilise le couple (admin, admin) comme identifiants de connexion. On sait qu'ils sont reconnus par le serveur ;

Les résultats de l'exécution sont les suivants :

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\http\axios-03.js"
2.  action init-session en cours ------------------------------------------
3.  succès ----------------------------------------
4.  réponse= { action: 'init-session',
5.    'état': 700,
6.    'réponse': 'session démarrée avec type [json]' }
7.  action authentifier-utilisateur en cours ------------------------------------------
8.  succès ----------------------------------------
9.  réponse= { action: 'authentifier-utilisateur',
10.   'état': 103,
11.   'réponse':
12.    [ 'pas de session en cours. Commencer par action [init-session]' ] }
13.
14. [Done] exited with code=0 in 0.834 seconds
```

- lignes 9-12 : l'authentification utilisateur échoue : le serveur n'a pas retenu le fait qu'on avait initié une session jSON. Cela vient du fait qu'on n'a pas renvoyé le cookie de session envoyé en réponse à la 1ère requête **[init-session]** ;

12.8 script [axios-04]

Le script **[axios-04]** amène deux améliorations au script **[axios-03]** :

- il gère le cookie de session ;
- il factorise dans une fonction **[getRemoteData]** ce qui est commun aux fonctions **[initSession]** et **[authentifierUtilisateur]** ;

```
1.  'use strict';
2.  import axios from 'axios';
3.  import qs from 'qs'
4.
5.  // configuration axios
6.  axios.defaults.timeout = 2000;
7.  axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';
8.
9.  // cookie de session
10. const sessionCookieName = "PHPSESSID";
11. let sessionCookie = '';
12.
```

```
13. // init session
14. async function initSession(axios) {
15.   // options de la requête HHTP [get /main.php?action=init-session&type=json]
16.   const options = {
17.     method: "GET",
18.     // paramètres de l'URL
19.     params: {
20.       action: 'init-session',
21.       type: 'json'
22.     }
23.   };
24.   // exécution de la requête HTTP
25.   return await getRemoteData(axios, options);
26. }
27.
28. async function authentifierUtilisateur(axios, user, password) {
29.   // options de la requête HHTP [post /main.php?action=authentifier-utilisateur]
30.   const options = {
31.     method: "POST",
32.     headers: {
33.       'Content-type': 'application/x-www-form-urlencoded',
34.     },
35.     // corps du POST
36.     data: qs.stringify({
37.       user: user,
38.       password: password
39.     }),
40.     // paramètres de l'URL
41.     params: {
42.       action: 'authentifier-utilisateur'
43.     }
44.   };
45.   // exécution de la requête HTTP
46.   return await getRemoteData(axios, options);
47. }
48.
49. async function getRemoteData(axios, options) {
50.   // pour le cookie de session
51.   if (!options.headers) {
52.     options.headers = {};
53.   }
54.   options.headers.Cookie = sessionCookie;
55.   // exécution de la requête HTTP
56.   let response;
57.   try {
58.     // requête asynchrone
59.     response = await axios.request('main.php', options);
60.   } catch (error) {
61.     // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
62.     if (error.response) {
63.       // la réponse du serveur est dans [error.response]
64.       response = error.response;
65.     } else {
66.       // on relance l'erreur
67.       throw error;
68.     }
69.   }
70.   // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-même)
71.   // on récupère le cookie de session s'il existe
72.   const setCookie = response.headers['set-cookie'];
73.   if (setCookie) {
74.     // setCookie est un tableau
75.     // on cherche le cookie de session dans ce tableau
76.     let trouvé = false;
77.     let i = 0;
78.     while (!trouvé && i < setCookie.length) {
79.       // on cherche le cookie de session
80.       const results = RegExp('^(' + sessionCookieName + '.+?);').exec(setCookie[i]);
81.       if (results) {
82.         // on mémorise le cookie de session
83.         // eslint-disable-next-line require-atomic-updates
84.         sessionCookie = results[1];
85.         // on a trouvé
86.         trouvé = true;
```

```
87.       } else {
88.         // élément suivant
89.         i++;
90.       }
91.     }
92.   }
93.   // la réponse du serveur est dans [response.data]
94.   return response.data;
95. }
96.
97. // la fonction main exécute les fonctions asynchrones une par une
98. async function main() {
99.   try {
100.    // init-session
101.    console.log("action init-session en cours ------------------------------------------");
102.    const response1 = await initSession(axios);
103.    console.log("succès ----------------------------------------------");
104.    console.log("réponse=", response1);
105.    // authentifier-utilisateur
106.    console.log("action authentifier-utilisateur en cours
    ------------------------------------------");
107.    const response2 = await authentifierUtilisateur(axios, 'admin', 'admin');
108.    console.log("succès ----------------------------------------------");
109.    console.log("réponse=", response2)
110.  } catch (error) {
111.    console.log("erreur ----------------------------------------------");
112.    console.log("erreur=", error.message);
113.  }
114.}
115.
116.// test
117.main();
```

Commentaires

- lignes 14-26 : la fonction **[initSession]**. Elle se contente désormais de préparer la requête HTTP à envoyer au serveur mais ne l'exécute pas. Elle confie ce rôle à la méthode **[getRemoteDate]** des lignes 49-95 ;
- lignes 28-47 : la fonction **[authentifierUtilisateur]** suit la même démarche ;
- ligne 49 : la fonction **[getRemoteData]** reçoit les deux informations qui lui permettent d'exécuter une requête HTTP:
  - **[axios]**, l'objet qui va se charger d'envoyer la requête et de recevoir la réponse ;
  - **[options]**, les options de configuration de la requête à envoyer au serveur ;
- ligne 59 : exécution de la requête et attente bloquante de sa réponse jSON ;
- lignes 60-68 : gestion de l'éventuelle exception ;
- ligne 64 : on récupère la réponse qui peut être encapsulée dans l'objet d'erreur ;
- ligne 67 : si le serveur a lancé une exception sans y inclure la réponse du serveur, alors on remonte l'erreur reçue au code appelant ;
- la fonction **[getRemoteData]** gère le cookie de session :
  - il le mémorise dans la variable **[sessionCookie]** (ligne 11) lorsqu'il le reçoit la 1ère fois ;
  - il le renvoie ensuite à chaque nouvelle requête HTTP ;
- ligne 72-92 : **[getRemoteData]** analyse chaque réponse du serveur pour savoir s'il a envoyé l'entête HTTP **[Set-Cookie]**. On sait que le serveur envoie un cookie de session nommé **[PHPSESSID]** (ligne 10). C'est donc ce cookie que l'on recherche (ligne 10) ;
- ligne 72 : on récupère les entêtes HTTP **[Set-Cookie]** s'ils existent (la casse n'a pas d'importance). Il peut y avoir en effet plusieurs entêtes **[Set-Cookie]** et c'est donc un tableau que l'on récupère ;
- ligne 73 : si on a récupéré un tableau de cookies ;
- lignes 78-90 : on cherche le cookie de session parmi tous les cookies du tableau ;
- ligne 80 : l'expression relationnelle qui permet de chercher le cookie de session dans le cookie n° i ;
- ligne 81 : si la comparaison a ramené des résultats ;
- ligne 84 : on a dans results**[1]**, la 1ère parenthèse du modèle de l'expression relationnelle, ç-à-d (PHPSESSID=xxxx) jusqu'au ; (non inclus) qui termine le cookie de session ;
- lignes 50-54 : à chaque requête, le cookie de session est inclus dans les entêtes HTTP de la requête. La 1ère fois, ce cookie est vide et sera alors ignoré par le serveur ;

Les résultats de l'exécution sont les suivants :

```
[Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
\Data\st-2019\dev\es6\javascript\http\axios-04.js"
action init-session en cours ---------------------------------------------
succès -------------------------------------------
réponse= { action: 'init-session',
  'état': 700,
  'réponse': 'session démarrée avec type [json]' }
action authentifier-utilisateur en cours ---------------------------------------------
succès -------------------------------------------
réponse= { action: 'authentifier-utilisateur',
  'état': 200,
  'réponse': 'Authentification réussie [admin, admin]' }

[Done] exited with code=0 in 0.982 seconds
```

13 Les classes

Nous introduisons ici les classes d'ECMAScript 6. Tout d'abord nous montrons que les fonctions peuvent être déjà utilisées comme des classes.

13.1 script [class-00]

Le script suivant montre une utilisation inhabituelle de fonctions. On les utilise ici comme des objets.

```
25. 'use strict';
26. // une fonction peut être utilisée comme un objet
27.
28. // une coquille vide
29. function f() {
30.
31. }
32. // à qui on attribue des propriétés de l'extérieur
33. f.prop1 = "val1";
34. f.show = function () {
35.   console.log(this.prop1);
36. }
37. // utilisation de f
38. f.show();
39.
40. // une fonction g fonctionnant comme une classe
41. function g() {
42.   this.prop2 = "val2";
43.   this.show = function () {
44.     console.log(this.prop2);
45.   }
46. }
47. // instanciation de la fonction avec [new]
48. new g().show();
```

Commentaires

- lignes 5-7 : le corps de la fonction f ne définit aucune propriété ;
- lignes 9-12 : on donne de l'extérieur des propriétés à la fonction f ;
- ligne 14 : utilisation de la function (objet) f. Notez qu'on n'écrit pas `[f()]` mais simplement `[f]`. On a là la notation d'un objet ;
- lignes 17-22 : on définit une fonction `[g]` comme si c'était une classe avec propriétés et méthodes ;
- ligne 24 : la fonction `[g]` est instanciée par `[new g()]` ;

Résultats de l'exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\classes\class-00.js"
2.  val1
3.  val2
```

ES6 a introduit la notion de classe qui nous permet désormais d'éviter de passer par des fonctions pour avoir des classes.

13.2 script [class-01]

Le script **[class-01]** présente une classe **[Personne]** :

```
1.  // classe
2.  class Personne {
3.
4.    // constructeur
5.    constructor(nom, prénom, âge) {
6.      this.nom = nom;
7.      this.prénom = prénom;
8.      this.âge = âge;
9.    }
10.
11.   // getters et setters
12.   get nom() {
13.     return this._nom;
14.   }
15.   set nom(value) {
16.     this._nom = value;
17.   }
18.
19.   get prénom() {
20.     return this._prénom;
21.   }
22.   set prénom(value) {
23.     this._prénom = value;
24.   }
25.
26.   get âge() {
27.     return this._âge;
28.   }
29.   set âge(value) {
30.     this._âge = value;
31.   }
32.
33.   // toString en JSON
34.   toString() {
35.     return JSON.stringify(this);
36.   }
37. }
38.
39. // appel de la classe
40. function main() {
41.   const personne = new Personne("Poirot", "Hercule", 66);
42.   console.log("personne=", personne.toString(), typeof (personne), personne instanceof
    (Personne));
43. }
44.
45. // appel de main
46. main();
```

Commentaires

- ligne 2 : le mot clé **[class]** désigne une classe ;
- lignes 5-9 : le mot clé **[constructor]** désigne le constructeur de la classe. Il ne peut y en avoir qu'un au plus. Il sert à construire et initialiser une instance de la classe. Notez qu'il n'y a pas de déclaration des propriétés **[nom, prénom, âge]** ;
- lignes 11-36 : propriétés de la classe. On retrouve ici des choses déjà vues dans le paragraphe des objets 4, page 43. Seule la syntaxe diffère ;
- ligne 41 : création d'un objet de type **[Personne]**. A partir de maintenant l'objet **[personne]** s'utilise comme un objet littéral. **[typeof (personne)]** vaut « object » et l'expression **[personne instanceof (Personne)]** est vraie. Il est donc possible de connaître le type exact d'une instance de classe ;

Résultats de l'exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\classes\class-01.js"
2.  personne= {"_nom":"Poirot","_prénom":"Hercule","_âge":66} object true
```

13.3 script [class-02]

Ce script montre la possibilité d'hériter d'une classe avec le mot clé **[extends]**.

Tout d'abord nous isolons la classe **[Personne]** dans un module **[Personne.js]** :

```
1.  // classe
2.  class Personne {
3.
4.    // constructeur
5.    constructor(nom, prénom, âge) {
6.      this.nom = nom;
7.      this.prénom = prénom;
8.      this.âge = âge;
9.    }
10.
11.   // getters et setters
12.   get nom() {
13.     return this._nom;
14.   }
15.   set nom(value) {
16.     this._nom = value;
17.   }
18.
19.   get prénom() {
20.     return this._prénom;
21.   }
22.   set prénom(value) {
23.     this._prénom = value;
24.   }
25.
26.   get âge() {
27.     return this._âge;
28.   }
29.   set âge(value) {
30.     this._âge = value;
31.   }
32.
33.   // toString en JSON
34.   toString() {
35.     return JSON.stringify(this);
36.   }
37. }
38. // export classe
39. export default Personne;
```

- ligne 39 : nous exportons la classe **[Personne]** pour que des scripts puissent l'importer ;

Le script **[class-02]** crée une classe **[Enseignant]** dérivée de la classe **[Personne]** :

```
1.  // imports
2.  import Personne from './Personne';
3.
4.  // classe
5.  class Enseignant extends Personne {
6.
7.    // constructeur
8.    constructor(nom, prénom, âge, discipline) {
9.      super(nom, prénom, âge);
10.     this.discipline = discipline;
11.   }
12.
13.   // getters et setters
14.   get discipline() {
15.     return this._discipline;
16.   }
17.   set discipline(value) {
18.     this._discipline = value;
```

```
19.   }
20.
21. }
22.
23. // appel de la classe
24. function main() {
25.   const enseignant = new Enseignant("Poirot", "Hercule", 66, "détective");
26.   console.log("enseignant=", enseignant.toString(), typeof (enseignant), enseignant instanceof
    Enseignant);
27. }
28.
29. // appel de main
30. main();
```

Commentaires

- ligne 2 : on importe la classe **[Personne]** à partir du module **[Personne.js]** qui se trouve dans le même dossier que **[class-02]** ;
- ligne 5 : la classe **[Enseignant]** étend (hérite de) la classe **[Personne]** avec le mot clé **[extends]** : elle lui ajoute une propriété **[_discipline]** avec les getter / setter qui vont avec ;
- lignes 8-11 : le constructeur de la classe **[Enseignant]** reçoit quatre valeurs pour initialiser les quatre propriétés de la classe ;
- ligne 9 : le mot clé **[super]** appelle le constructeur de la classe parent **[Personne]** qui va donc initialiser les propriétés **[_nom, _prénom, _âge]** ;
- ligne 10 : on initialise la propriété **[_discipline]** qui lui appartient à la classe **[Enseignant]** ;
- lignes 14-19 : le getter et le setter de la propriété **[_discipline]** ;
- ligne 25 : on crée un objet de type **[Enseignant]** ;
- ligne 26 : on utilise la méthode **[enseignant.toString()]**. La classe **[Enseignant]** n'a pas cette méthode. C'est alors celle de sa classe parent qui est automatiquement utilisée. Cette méthode rend l'expression **[JSON.stringify(this)]** où **[this]** va être ici un objet **[Enseignant]** et non un objet **[Personne]**. C'est qu'on appelle en programmation objet, le polymorphisme des classes. Un grand mot pour Javascript qui n'est pas un langage orienté objets. Néanmoins, Javascript fait ici ce qu'on attend de lui : il affiche bien un enseignant ;

Les résultats de l'exécution sont les suivants :

```
1. [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
   \Data\st-2019\dev\es6\javascript\classes\class-02.js"
2. enseignant= {"_nom":"Poirot","_prénom":"Hercule","_âge":66,"_discipline":"détective"} object tr
   ue
```

- ligne 2 : Javascript reconnaît bien que la variable **[enseignant]** est de type **[Enseignant]** ;

13.4 script [class-03]

Le script **[class-03]** montre qu'une classe fille peut redéfinir propriétés et méthodes de sa classe parent. Ici, nous redéfinissons la méthode **[toString]** de la classe parent :

```
1.  // imports
2.  import Personne from './Personne';
3.
4.  // classe
5.  class Enseignant extends Personne {
6.
7.    // constructeur
8.    constructor(nom, prénom, âge, discipline) {
9.      super(nom, prénom, âge);
10.     this.discipline = discipline;
11.   }
12.
13.   // getters et setters
14.   get discipline() {
15.     return this._discipline;
16.   }
17.   set discipline(value) {
18.     this._discipline = value;
19.   }
20.
21.   // redéfinition de toString
```

```
22.   toString() {
23.     return "[Enseignant]" + JSON.stringify(this);
24.   }
25. }
26.
27. // appel de la classe
28. function main() {
29.   const enseignant = new Enseignant("Poirot", "Hercule", 66, "détective");
30.   console.log("enseignant=", enseignant.toString(), typeof (enseignant), enseignant instanceof
    Enseignant);
31. }
32.
33. // appel de main
34. main();
```

Les résultats de l'exécution sont les suivants :

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\classes\class-03.js"
2.  enseignant= [Enseignant]
    {"_nom":"Poirot","_prénom":"Hercule","_âge":66,"_discipline":"détective"} object true
```

13.5 script [class-04]

Le script **[class-04]** montre de nouveau le polymorphisme à l'oeuvre : là où une fonction attend un paramètre formel de type **[Personne]**, on peut passer un type dérivé tel que **[Enseignant]**. En effet, de par la dérivation, le type **[Enseignant]** a tous les attributs du type **[Personne]**.

Tout d'abord, nous isolons le type **[Enseignant]** dans un module **[Enseignant.js]** :

```
1.  // imports
2.  import Personne from './Personne';
3.
4.  // classe
5.  class Enseignant extends Personne {
6.
7.    // constructeur
8.    constructor(nom, prénom, âge, discipline) {
9.      super(nom, prénom, âge);
10.     this.discipline = discipline;
11.   }
12.
13.   // getters et setters
14.   get discipline() {
15.     return this._discipline;
16.   }
17.   set discipline(value) {
18.     this._discipline = value;
19.   }
20.
21. }
22.
23. // export classe
24. export default Enseignant;
```

- ligne 24 : la classe **[Enseignant]** est exportée pour que d'autres scripts puissent l'importer ;

Le script **[class-04]** est le suivant :

```
1.  // imports
2.  import Enseignant from './Enseignant';
3.  import Personne from './Personne';
4.
5.  // fonction acceptant une personne comme paramètre
6.  function show(personne) {
7.    // dans tous les cas
8.    console.log("paramètre=", personne.toString(), typeof (personne));
9.    // instance de Personne
10.   if (personne instanceof Personne) {
11.     console.log("personne=", personne.toString());
12.   }
```

```
13.   // instance de Enseignant
14.   if (personne instanceof Enseignant) {
15.     console.log("enseignant=", personne.toString());
16.   }
17. }
18.
19. // appel de show avec un enseignant
20. show(new Enseignant("Poirot", "Hercule", 66, "détective"));
21. show(new Personne("Marple", "Miss", 70));
```

- ligne 6 : la fonction **[show]** attend un type **[Personne]** ou dérivé ;
- ligne 8 : on affiche la chaîne du paramètre et son type. On va trouver **[object]** ;
- lignes 10-16 : on est capable de savoir si c'est un type **[Personne]** ou un type **[Enseignant]**. Le code peut donc être adapté au type réel du paramètre ;

Les résultats de l'exécution sont les suivants :

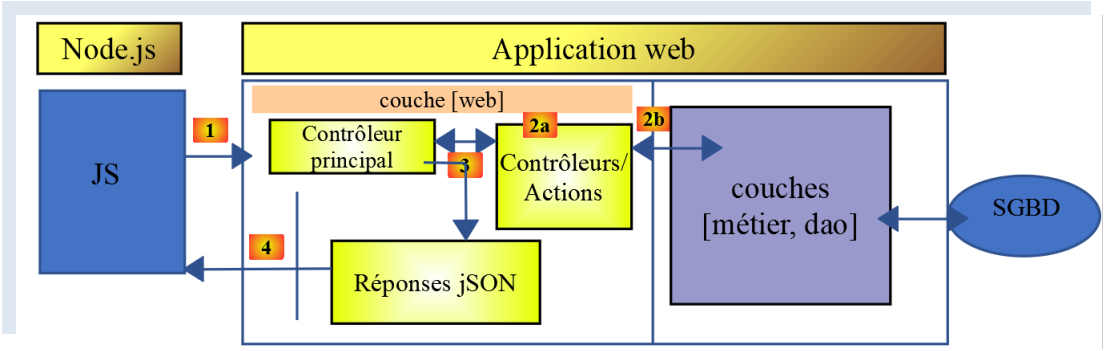
```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\classes\class-04.js"
2.  paramètre= {"_nom":"Poirot","_prénom":"Hercule","_âge":66,"_discipline":"détective"} object
3.  personne= {"_nom":"Poirot","_prénom":"Hercule","_âge":66,"_discipline":"détective"}
4.  enseignant= {"_nom":"Poirot","_prénom":"Hercule","_âge":66,"_discipline":"détective"}
5.  paramètre= {"_nom":"Marple","_prénom":"Miss","_âge":70} object
6.  personne= {"_nom":"Marple","_prénom":"Miss","_âge":70}
```

- lignes 4 et 6 : Javascript connaît correctement le type des instances de classe ;

14 Clients HTTP Javascript du service de calcul de l'impôt

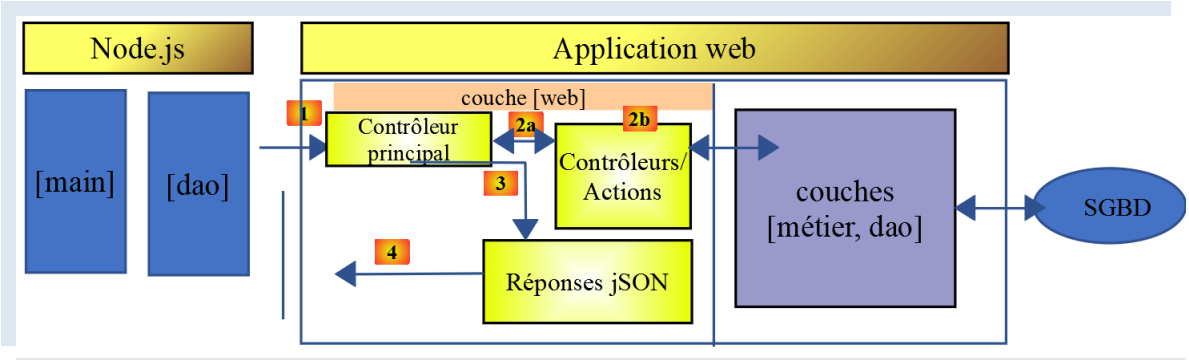
14.1 Introduction

Nous nous proposons ici d'écrire un client `[node.js]` de la version 14 du service de calcul de l'impôt. L'architecture client / serveur sera la suivante :

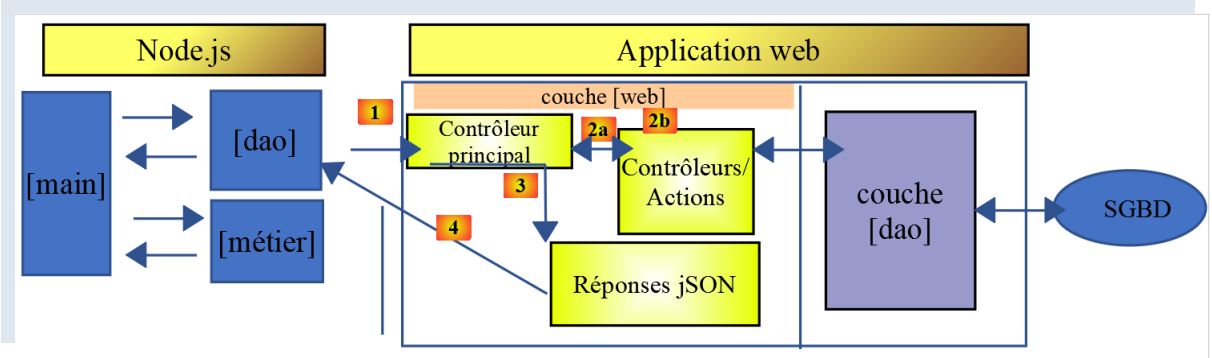


Nous étudierons deux versions du client :

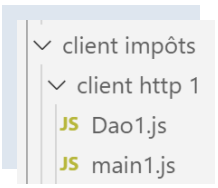
- la version 1 du client aura la structure `[main, dao]` en couches suivante :



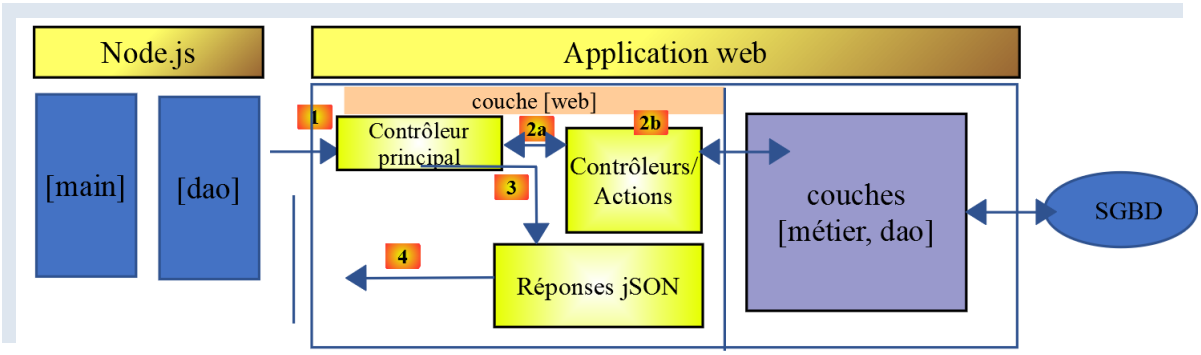
- la version 2 du client aura une structure `[main, métier, dao]`. La couche `[métier]` du serveur sera déportée sur le client :



14.2 Client HTTP 1



Comme nous l'avons dit, le client HTTP 1 implémente l'architecture client / serveur suivante :



Nous implémenterons :

- la couche **[dao]** sous la forme d'une classe ;
- la couche **[main]** sous la forme d'un script utilisant cette classe ;

14.2.1 La couche [dao]

La couche **[dao]** sera implémentée par la classe suivante **[Dao1.js]** :

```
1.  'use strict';
2.
3.  // imports
4.  import qs from 'qs'
5.
6.  class Dao1 {
7.
8.    // constructeur
9.    constructor(axios) {
10.     // bibliothèque axios pour faire les requêtes HTTP
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.   // calcul de l'impôt
54.   async  calculerImpot(marié, enfants, salaire) {
55.     // options de la requête HHTP [post /main.php?action=calculer-impot]
56.     const options = {
57.       method: "POST",
58.       headers: {
59.         'Content-type': 'application/x-www-form-urlencoded',
60.       },
61.       // corps du POST [marié, enfants, salaire]
62.       data: qs.stringify({
63.         marié: marié,
64.         enfants: enfants,
65.         salaire: salaire
66.       }),
67.       // paramètres de l'URL
68.       params: {
69.         action: 'calculer-impot'
70.       }
71.     };
72.     // exécution de la requête HTTP
73.     const data = await this.getRemoteData(options);
74.     // résultat
75.     return data;
76.   }
77.
78.   // liste des simulations
79.   async  listeSimulations() {
80.     // options de la requête HHTP [get /main.php?action=lister-simulations]
81.     const options = {
82.       method: "GET",
83.       // paramètres de l'URL
84.       params: {
85.         action: 'lister-simulations'
86.       },
87.     };
88.     // exécution de la requête HTTP
89.     const data = await this.getRemoteData(options);
90.     // résultat
91.     return data;
92.   }
93.
94.   // liste des simulations
95.   async  supprimerSimulation(index) {
96.     // options de la requête HHTP  [get /main.php?action=supprimer-simulation&numéro=index]
97.     const options = {
98.       method: "GET",
99.       // paramètres de l'URL
100.      params: {
101.        action: 'supprimer-simulation',
102.        numéro: index
103.      },
104.    };
```

```
105.    // exécution de la requête HTTP
106.    const data = await this.getRemoteData(options);
107.    // résultat
108.    return data;
109.  }
110.
111.  async  getRemoteData(options) {
112.    // pour le cookie de session
113.    if (!options.headers) {
114.      options.headers = {};
115.    }
116.    options.headers.Cookie = this.sessionCookie;
117.    // exécution de la requête HTTP
118.    let response;
119.    try {
120.      // requête asynchrone
121.      response = await this.axios.request('main.php', options);
122.    } catch (error) {
123.      // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
124.      if (error.response) {
125.        // la réponse du serveur est dans [error.response]
126.        response = error.response;
127.      } else {
128.        // on relance l'erreur
129.        throw error;
130.      }
131.    }
132.    // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-
    même)
133.    // on récupère le cookie de session s'il existe
134.    const setCookie = response.headers['set-cookie'];
135.    if (setCookie) {
136.      // setCookie est un tableau
137.      // on cherche le cookie de session dans ce tableau
138.      let trouvé = false;
139.      let i = 0;
140.      while (!trouvé && i < setCookie.length) {
141.        // on cherche le cookie de session
142.        const results = RegExp('^(' + this.sessionCookieName + '.+?);').exec(setCookie[i]);
143.        if (results) {
144.          // on mémorise le cookie de session
145.          // eslint-disable-next-line require-atomic-updates
146.          this.sessionCookie = results[1];
147.          // on a trouvé
148.          trouvé = true;
149.        } else {
150.          // élément suivant
151.          i++;
152.        }
153.      }
154.    }
155.    // la réponse du serveur est dans [response.data]
156.    return response.data;
157.  }
158.}
159.
160.// export de la classe
161.export default Dao1;
```

- nous utilisons ici ce que nous avons appris au paragraphe lien, où nous avons présenté la bibliothèque **[axios]** permettant de faire des requêtes HTTP aussi bien sous **[node.js]** que dans un navigateur. On regardera en particulier le script du paragraphe lien ;
- lignes 9-15 : le constructeur de la classe. Celle-ci aura trois propriétés :
  - **[axios]** : l'objet **[axios]** permettant de faire les requêtes HTTP. Celui-ci est transmis par le code appelant ;
  - **[sessionCookieName]** : selon les serveurs, le cookie de session porte des noms différents. Ici, c'est **[PHPSESSID]** ;
  - **[sessionCookie]** : le cookie de session envoyé par le serveur et mémorisé par le client ;
- lignes 53-76 : la fonction asynchrone **[calculerImpot]** fait la requête **[post /main.php?action=calculer-impot]** en postant les paramètres **[marié, enfants, salaire]**. Elle rend la chaîne jSON transmise par le serveur sous la forme d'un objet Javascript ;

- lignes 79-92 : la fonction asynchrone **[listeSimulations]** fait la requête [`get /main.php?action=lister-simulations`. Elle rend la chaîne jSON transmise par le serveur sous la forme d'un objet Javascript ;
- lignes 95-109 : la fonction asynchrone **[supprimerSimulation]** fait la requête **[get /main.php?action=supprimer-simulation&numéro=index]**. Elle rend la chaîne jSON transmise par le serveur sous la forme d'un objet Javascript ;
- ligne 121 : on utilise la notation **[this.axios]** car ici, l'objet **[axios]** transmis au constructeur a été mémorisé dans la propriété **[this.axios]** ;
- ligne 161 : la classe **[Dao1]** est exportée pour pouvoir être utilisée ;

14.2.2 Le script [main1.js]

Le script **[main1.js]** fait une série d'appels au serveur à l'aide de la classe **[Dao1]** :

- initialisation d'une session jSON ;
- authentification avec **[admin, admin]** ;
- demande trois calculs d'impôts ;
- demande la liste des simulations ;
- supprime l'une d'elles ;

Le code est le suivant :

```
1.  // import axios
2.  import axios from 'axios';
3.  // import de la classe Dao1
4.  import Dao from './Dao1';
5.
6.  // fonction asynchrone [main]
7.  async function main() {
8.    // configuration axios
9.    axios.defaults.timeout = 2000;
10.   axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';
11.   // instanciation couche [dao]
12.   const dao = new Dao(axios);
13.   // utilisation de la couche [dao]
14.   try {
15.     // init session
16.     log("-----------init-session");
17.     let response = await dao.initSession();
18.     log(response);
19.     // authentification
20.     log("-----------authentifier-utilisateur");
21.     response = await dao.authentifierUtilisateur("admin", "admin");
22.     log(response);
23.     // calculs d'impôt
24.     log("-----------calculer-impot x 3");
25.     response = await Promise.all([
26.       dao.calculerImpot("oui", 2, 45000),
27.       dao.calculerImpot("non", 2, 45000),
28.       dao.calculerImpot("non", 1, 30000)
29.     ]);
30.     log(response);
31.     // liste des simulations
32.     log("-----------liste-des-simulations");
33.     response = await dao.listeSimulations();
34.     log(response);
35.     // suppression d'une simulation
36.     log("-----------suppression simulation n° 1");
37.     response = await dao.supprimerSimulation(1);
38.     log(response);
39.   } catch (error) {
40.     // on logue l'erreur
41.     console.log("erreur=", error.message);
42.   }
43. }
44.
45. // log jSON
46. function log(object) {
47.   console.log(JSON.stringify(object, null, 2));
48. }
49.
```

```
50. // exécution
51. main();
```

Commentaires

- ligne 2 : on importe la bibliothèque **[axios]** ;
- ligne 4 : on importe la classe **[Dao]** ;
- ligne 7 : la fonction **[main]** qui dialogue avec le serveur est asynchrone ;
- lignes 9-10 : configuration par défaut des requêtes HTTP qui seront faites au serveur :
  - ligne 9 : **[timeout]** de 2 secondes ;
  - ligne 10 : toutes les URL ont pour préfixe, l'URL base de la version 14 du serveur de calcul de l'impôt ;
- ligne 12 : la couche **[Dao]** est construite. On peut désormais l'utiliser ;
- lignes 46-48 : la fonction **[log]** a pour objet d'afficher la chaîne jSON d'un objet Javascript sous une forme embellie : sous forme verticale avec une indentation de deux espaces (3ième paramètre) ;
- lignes 15-18 : initialisation de la session jSON ;
- lignes 19-22 : authentification ;
- lignes 23-30 : trois calculs d'impôt sont demandés en parallèle. Grâce à **[await Promise.all]**, l'exécution est bloquée tant que les trois résultats n'ont pas été tous obtenus ;
- lignes 31-34 : liste des simulations ;
- lignes 35-38 : suppression d'une simulation ;
- lignes 39-42 : gestion de l'éventuelle exception ;

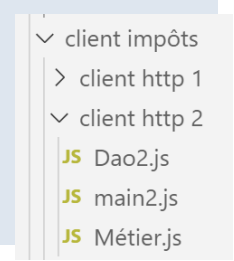
Les résultats de l'exécution sont les suivants :

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\client impôts\client http 1\main1.js"
2.  "-----------init-session"
3.  {
4.    "action": "init-session",
5.    "état": 700,
6.    "réponse": "session démarrée avec type [json]"
7.  }
8.  "-----------authentifier-utilisateur"
9.  {
10.   "action": "authentifier-utilisateur",
11.   "état": 200,
12.   "réponse": "Authentification réussie [admin, admin]"
13. }
14. "-----------calculer-impot x 3"
15. [
16.   {
17.     "action": "calculer-impot",
18.     "état": 300,
19.     "réponse": {
20.       "marié": "oui",
21.       "enfants": "2",
22.       "salaire": "45000",
23.       "impôt": 502,
24.       "surcôte": 0,
25.       "décôte": 857,
26.       "réduction": 126,
27.       "taux": 0.14
28.     }
29.   },
30.   {
31.     "action": "calculer-impot",
32.     "état": 300,
33.     "réponse": {
34.       "marié": "non",
35.       "enfants": "2",
36.       "salaire": "45000",
37.       "impôt": 3250,
38.       "surcôte": 370,
39.       "décôte": 0,
40.       "réduction": 0,
41.       "taux": 0.3
42.     }
43.   },
44.   {
45.     "action": "calculer-impot",
```

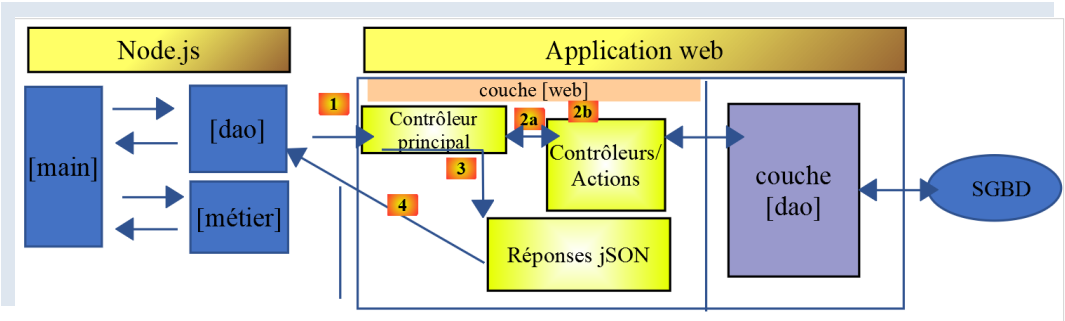
```
    "état": 300,
    "réponse": {
      "marié": "non",
      "enfants": "1",
      "salaire": "30000",
      "impôt": 1687,
      "surcôte": 0,
      "décôte": 0,
      "réduction": 0,
      "taux": 0.14
    }
  }
]
"-----------liste-des-simulations"
{
  "action": "lister-simulations",
  "état": 500,
  "réponse": [
    {
      "marié": "oui",
      "enfants": "2",
      "salaire": "45000",
      "impôt": 502,
      "surcôte": 0,
      "décôte": 857,
      "réduction": 126,
      "taux": 0.14,
      "arrayOfAttributes": null
    },
    {
      "marié": "non",
      "enfants": "2",
      "salaire": "45000",
      "impôt": 3250,
      "surcôte": 370,
      "décôte": 0,
      "réduction": 0,
      "taux": 0.3,
      "arrayOfAttributes": null
    },
    {
      "marié": "non",
      "enfants": "1",
      "salaire": "30000",
      "impôt": 1687,
      "surcôte": 0,
      "décôte": 0,
      "réduction": 0,
      "taux": 0.14,
      "arrayOfAttributes": null
    }
  ]
}
"-----------suppression simulation n° 1"
{
  "action": "supprimer-simulation",
  "état": 600,
  "réponse": [
    {
      "marié": "oui",
      "enfants": "2",
      "salaire": "45000",
      "impôt": 502,
      "surcôte": 0,
      "décôte": 857,
      "réduction": 126,
      "taux": 0.14,
      "arrayOfAttributes": null
    },
    {
      "marié": "non",
      "enfants": "1",
      "salaire": "30000",
      "impôt": 1687,
```

```
120.      "surcôte": 0,
121.      "décôte": 0,
122.      "réduction": 0,
123.      "taux": 0.14,
124.      "arrayOfAttributes": null
125.    }
126.  ]
127.}
128.
129.[Done] exited with code=0 in 0.516 seconds
```

14.3 Client HTTP 2



L'architecture du client HTTP2 est la suivante :



On a déporté la couche **[métier]** du serveur vers le client Javascript. Contrairement à ce que nous avons pu faire dans le cours PHP7, la couche **[main]** n'aura pas ici à passer par la couche **[métier]** pour atteindre la couche **[dao]**. Nous utiliserons ces deux couches comme des centres de compétences :

- la couche **[main]** passe par la couche **[dao]** dès qu'elle a besoin de données qui sont sur le serveur ;
- la couche **[main]** demande à la couche **[métier]** de faire les calculs de l'impôt ;
- la couche **[métier]** est indépendante de la couche **[dao]** et ne fait jamais appel à elle ;

14.3.1 La classe Javascript [Métier]

L'essence de la classe **[Métier]** en PHP a été décrite dans l'article lien. C'est un code plutôt complexe qu'on rappelle ici, non pour l'expliquer, mais pour pouvoir le traduire en Javascript :

```
1.  <?php
2.
3.  // espace de noms
4.  namespace Application;
5.
6.  class Metier implements InterfaceMetier {
7.    // couche Dao
8.    private $dao;
9.    // données administration fiscale
10.   private $taxAdminData;
11.
12.   //-------------------------------------------
13.   // setter couche [dao]
14.   public function setDao(InterfaceDao $dao) {
15.     $this->dao = $dao;
```

```
16.     return $this;
17.   }
18.
19.   public function __construct(InterfaceDao $dao) {
20.     // on mémorise une référence sur la couche [dao]
21.     $this->dao = $dao;
22.     // on récupère les données permettant le calcul de l'impôt
23.     // la méthode [getTaxAdminData] peut lancer une exception ExceptionImpots
24.     // on la laisse alors remonter au code appelant
25.     $this->taxAdminData = $this->dao->getTaxAdminData();
26.   }
27.
28. // calcul de l'impôt
29. // ------------------------------------------------------------------------
30.   public function calculerImpot(string $marié, int $enfants, int $salaire): array {
31.     // $marié : oui, non
32.     // $enfants : nombre d'enfants
33.     // $salaire : salaire annuel
34.     // $this->taxAdminData : données de l'administration fiscale
35.     //
36.     // on vérifie qu'on a bien les données de l'administration fiscale
37.     if ($this->taxAdminData === NULL) {
38.       $this->taxAdminData = $this->getTaxAdminData();
39.     }
40.     // calcul de l'impôt avec enfants
41.     $result1 = $this->calculerImpot2($marié, $enfants, $salaire);
42.     $impot1 = $result1["impôt"];
43.     // calcul de l'impôt sans les enfants
44.     if ($enfants != 0) {
45.       $result2 = $this->calculerImpot2($marié, 0, $salaire);
46.       $impot2 = $result2["impôt"];
47.       // application du plafonnement du quotient familial
48.       $plafonDemiPart = $this->taxAdminData->getPlafondQfDemiPart();
49.       if ($enfants < 3) {
50.         // $PLAFOND_QF_DEMI_PART euros pour les 2 premiers enfants
51.         $impot2 = $impot2 - $enfants * $plafonDemiPart;
52.       } else {
53.         // $PLAFOND_QF_DEMI_PART euros pour les 2 premiers enfants, le double pour les
    suivants
54.         $impot2 = $impot2 - 2 * $plafonDemiPart - ($enfants - 2) * 2 * $plafonDemiPart;
55.       }
56.     } else {
57.       $impot2 = $impot1;
58.       $result2 = $result1;
59.     }
60.     // on prend l'impôt le plus fort
61.     if ($impot1 > $impot2) {
62.       $impot = $impot1;
63.       $taux = $result1["taux"];
64.       $surcôte = $result1["surcôte"];
65.     } else {
66.       $surcôte = $impot2 - $impot1 + $result2["surcôte"];
67.       $impot = $impot2;
68.       $taux = $result2["taux"];
69.     }
70.     // calcul d'une éventuelle décôte
71.     $décôte = $this->getDecôte($marié, $salaire, $impot);
72.     $impot -= $décôte;
73.     // calcul d'une éventuelle réduction d'impôts
74.     $réduction = $this->getRéduction($marié, $salaire, $enfants, $impot);
75.     $impot -= $réduction;
76.     // résultat
77.     return ["impôt" => floor($impot), "surcôte" => $surcôte, "décôte" => $décôte, "réduction"
    => $réduction, "taux" => $taux];
78.   }
79.
80. // ------------------------------------------------------------------------
81.   private function calculerImpot2(string $marié, int $enfants, float $salaire): array {
82.     // $marié : oui, non
83.     // $enfants : nombre d'enfants
84.     // $salaire : salaire annuel
85.     // $this->taxAdminData : données de l'administration fiscale
86.     //
87.     // nombre de parts
```

```
88.     $marié = strtolower($marié);
89.     if ($marié === "oui") {
90.       $nbParts = $enfants / 2 + 2;
91.     } else {
92.       $nbParts = $enfants / 2 + 1;
93.     }
94.     // 1 part par enfant à partir du 3ième
95.     if ($enfants >= 3) {
96.       // une demi-part de + pour chaque enfant à partir du 3ième
97.       $nbParts += 0.5 * ($enfants - 2);
98.     }
99.     // revenu imposable
100.    $revenuImposable = $this->getRevenuImposable($salaire);
101.    // surcôte
102.    $surcôte = floor($revenuImposable - 0.9 * $salaire);
103.    // pour des pbs d'arrondi
104.    if ($surcôte < 0) {
105.      $surcôte = 0;
106.    }
107.    // quotient familial
108.    $quotient = $revenuImposable / $nbParts;
109.    // calcul de l'impôt
110.    $limites = $this->taxAdminData->getLimites();
111.    $coeffR = $this->taxAdminData->getCoeffR();
112.    $coeffN = $this->taxAdminData->getCoeffN();
113.    // est mis à la fin du tableau limites pour arrêter la boucle qui suit
114.    $limites[count($limites) - 1] = $quotient;
115.    // recherche du taux d'imposition
116.    $i = 0;
117.    while ($quotient > $limites[$i]) {
118.      $i++;
119.    }
120.    // du fait qu'on a placé $quotient à la fin du tableau $limites, la boucle précédente
121.    // ne peut déborder du tableau $limites
122.    // maintenant on peut calculer l'impôt
123.    $impôt = floor($revenuImposable * $coeffR[$i] - $nbParts * $coeffN[$i]);
124.    // résultat
125.    return ["impôt" => $impôt, "surcôte" => $surcôte, "taux" => $coeffR[$i]];
126.  }
127.
128.  // revenuImposable=salaireAnnuel-abattement
129.  // l'abattement a un min et un max
130.  private function getRevenuImposable(float $salaire): float {
131.    // abattement de 10% du salaire
132.    $abattement = 0.1 * $salaire;
133.    // cet abattement ne peut dépasser $this->taxAdminData->getAbattementDixPourCentMax()
134.    if ($abattement > $this->taxAdminData->getAbattementDixPourCentMax()) {
135.      $abattement = $this->taxAdminData->getAbattementDixPourcentMax();
136.    }
137.    // l'abattement ne peut être inférieur à $this->taxAdminData-
    >getAbattementDixPourcentMin()
138.    if ($abattement < $this->taxAdminData->getAbattementDixPourcentMin()) {
139.      $abattement = $this->taxAdminData->getAbattementDixPourcentMin();
140.    }
141.    // revenu imposable
142.    $revenuImposable = $salaire - $abattement;
143.    // résultat
144.    return floor($revenuImposable);
145.  }
146.
147.// calcule une décôte éventuelle
148.  private function getDecôte(string $marié, float $salaire, float $impots): float {
149.    // au départ, une décôte nulle
150.    $décôte = 0;
151.    // montant maximal d'impôt pour avoir la décôte
152.    $plafondImpôtPourDécôte = $marié === "oui" ?
153.      $this->taxAdminData->getPlafondImpotCouplePourDecote() :
154.      $this->taxAdminData->getPlafondImpotCelibatairePourDecote();
155.    if ($impots < $plafondImpôtPourDécôte) {
156.      // montant maximal de la décôte
157.      $plafondDécôte = $marié === "oui" ?
158.        $this->taxAdminData->getPlafondDecoteCouple() :
159.        $this->taxAdminData->getPlafondDecoteCelibataire();
160.      // décôte théorique
```

```
161.      $décôte = $plafondDécôte - 0.75 * $impots;
162.      // la décôte ne peut dépasser le montant de l'impôt
163.      if ($décôte > $impots) {
164.        $décôte = $impots;
165.      }
166.      // pas de décôte <0
167.      if ($décôte < 0) {
168.        $décôte = 0;
169.      }
170.    }
171.    // résultat
172.    return ceil($décôte);
173.  }
174.
175.// calcule une réduction éventuelle
176.  private function getRéduction(string $marié, float $salaire, int $enfants, float $impots):
    float {
177.    // le plafond des revenus pour avoir droit à la réduction de 20%
178.    $plafondRevenuPourRéduction = $marié === "oui" ?
179.      $this->taxAdminData->getPlafondRevenusCouplePourReduction() :
180.      $this->taxAdminData->getPlafondRevenusCelibatairePourReduction();
181.    $plafondRevenuPourRéduction += $enfants * $this->taxAdminData->getValeurReducDemiPart();
182.    if ($enfants > 2) {
183.      $plafondRevenuPourRéduction += ($enfants - 2) * $this->taxAdminData-
    >getValeurReducDemiPart();
184.    }
185.    // revenu imposable
186.    $revenuImposable = $this->getRevenuImposable($salaire);
187.    // réduction
188.    $réduction = 0;
189.    if ($revenuImposable < $plafondRevenuPourRéduction) {
190.      // réduction de 20%
191.      $réduction = 0.2 * $impots;
192.    }
193.    // résultat
194.    return ceil($réduction);
195.  }
196.
197.  // calcul des impôts en mode batch
198.  public function executeBatchImpots(string $taxPayersFileName, string $resultsFileName, string
    $errorsFileName): void {
199.    // on laisse remonter les exceptions qui proviennent de la couche [dao]
200.    // on récupère les données contribuables
201.    $taxPayersData = $this->dao->getTaxPayersData($taxPayersFileName, $errorsFileName);
202.    // tableau des résultats
203.    $results = [];
204.    // on les exploite
205.    foreach ($taxPayersData as $taxPayerData) {
206.      // on calcule l'impôt
207.      $result = $this->calculerImpot(
208.        $taxPayerData->getMarié(),
209.        $taxPayerData->getEnfants(),
210.        $taxPayerData->getSalaire());
211.      // on complète [$taxPayerData]
212.      $taxPayerData->setMontant($result["impôt"]);
213.      $taxPayerData->setDécôte($result["décôte"]);
214.      $taxPayerData->setSurCôte($result["surcôte"]);
215.      $taxPayerData->setTaux($result["taux"]);
216.      $taxPayerData->setRéduction($result["réduction"]);
217.      // on met le résultat dans le tableau des résultats
218.      $results [] = $taxPayerData;
219.    }
220.    // enregistrement des résultats
221.    $this->dao->saveResults($resultsFileName, $results);
222.  }
223.
224.}
```

- lignes 19-26 : le constructeur de la classe PHP. Parce que nous avons dit qu'on construisait une couche **[métier]** indépendante de la couche **[dao]**, nous ferons en Javascript deux modifications à ce constructeur :
  - il ne recevra pas une instance de la couche **[dao]** (il n'en a plus besoin) ;
  - il ne demandera pas les données fiscales de l'administration **[taxAdminData]** à la couche **[dao]** : c'est le code appelant qui transmettra cette donnée au constructeur ;

- lignes 197-122 : nous n'implémenterons pas la méthode **[executeBatchImpots]** dont le but final était d'enregistrer des résultats de simulations dans un fichier texte. Nous voulons un code qui fonctionne à la fois sous **[node.js]** et dans un navigateur. Or sauvegarder des données sur le système de fichiers de la machine exécutant le navigateur client n'est pas possible ;

Avec ces restrictions, le code de la classe Javascript **[Métier]** est le suivant :

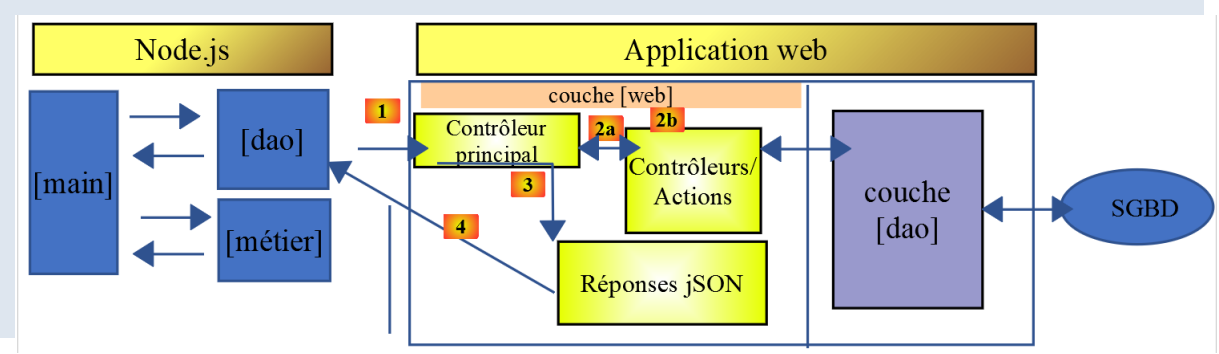
```
1.  'use strict';
2.
3.  // classe Métier
4.  class Métier {
5.
6.    // constructeur
7.    constructor(taxAdmindata) {
8.      // this.taxAdminData : données de l'administration fiscale
9.      this.taxAdminData = taxAdmindata;
10.   }
11.
12.   // calcul de l'impôt
13.   // ------------------------------------------------------------------------
14.   calculerImpot(marié, enfants, salaire) {
15.     // marié : oui, non
16.     // enfants : nombre d'enfants
17.     // salaire : salaire annuel
18.     // this.taxAdminData : données de l'administration fiscale
19.     //
20.     // calcul de l'impôt avec enfants
21.     const result1 = this.calculerImpot2(marié, enfants, salaire);
22.     const impot1 = result1["impôt"];
23.     // calcul de l'impôt sans les enfants
24.     let result2, impot2, plafondDemiPart;
25.     if (enfants !== 0) {
26.       result2 = this.calculerImpot2(marié, 0, salaire);
27.       impot2 = result2["impôt"];
28.       // application du plafonnement du quotient familial
29.       plafondDemiPart = this.taxAdminData.plafondQfDemiPart;
30.       if (enfants < 3) {
31.         // PLAFOND_QF_DEMI_PART euros pour les 2 premiers enfants
32.         impot2 = impot2 - enfants * plafondDemiPart;
33.       } else {
34.         // PLAFOND_QF_DEMI_PART euros pour les 2 premiers enfants, le double pour les suivants
35.         impot2 = impot2 - 2 * plafondDemiPart - (enfants - 2) * 2 * plafondDemiPart;
36.       }
37.     } else {
38.       // pas de reclacul de l'impôt
39.       impot2 = impot1;
40.       result2 = result1;
41.     }
42.     // on prend l'impôt le plus fortdans [impot1, impot2]
43.     let impot, taux, surcôte;
44.     if (impot1 > impot2) {
45.       impot = impot1;
46.       taux = result1["taux"];
47.       surcôte = result1["surcôte"];
48.     } else {
49.       surcôte = impot2 - impot1 + result2["surcôte"];
50.       impot = impot2;
51.       taux = result2["taux"];
52.     }
53.     // calcul d'une éventuelle décôte
54.     const décôte = this.getDecôte(marié, impot);
55.     impot -= décôte;
56.     // calcul d'une éventuelle réduction d'impôts
57.     const réduction = this.getRéduction(marié, salaire, enfants, impot);
58.     impot -= réduction;
59.     // résultat
60.     return {
61.       "impôt": Math.floor(impot), "surcôte": surcôte, "décôte": décôte, "réduction": réduction,
62.       "taux": taux
63.     };
64.   }
65.
66.   // ------------------------------------------------------------------------
```

```
67.   calculerImpot2(marié, enfants, salaire) {
68.     // marié : oui, non
69.     // enfants : nombre d'enfants
70.     // salaire : salaire annuel
71.     // this->taxAdminData : données de l'administration fiscale
72.     //
73.     // nombre de parts
74.     marié = marié.toLowerCase();
75.     let nbParts;
76.     if (marié === "oui") {
77.       nbParts = enfants / 2 + 2;
78.     } else {
79.       nbParts = enfants / 2 + 1;
80.     }
81.     // 1 part par enfant à partir du 3ième
82.     if (enfants >= 3) {
83.       // une demi-part de + pour chaque enfant à partir du 3ième
84.       nbParts += 0.5 * (enfants - 2);
85.     }
86.     // revenu imposable
87.     const revenuImposable = this.getRevenuImposable(salaire);
88.     // surcôte
89.     let surcôte = Math.floor(revenuImposable - 0.9 * salaire);
90.     // pour des pbs d'arrondi
91.     if (surcôte < 0) {
92.       surcôte = 0;
93.     }
94.     // quotient familial
95.     const quotient = revenuImposable / nbParts;
96.     // calcul de l'impôt
97.     const limites = this.taxAdminData.limites;
98.     const coeffR = this.taxAdminData.coeffR;
99.     const coeffN = this.taxAdminData.coeffN;
100.    // est mis à la fin du tableau limites pour arrêter la boucle qui suit
101.    limites[limites.length - 1] = quotient;
102.    // recherche du taux d'imposition
103.    let i = 0;
104.    while (quotient > limites[i]) {
105.      i++;
106.    }
107.    // du fait qu'on a placé quotient à la fin du tableau limites, la boucle précédente
108.    // ne peut déborder du tableau limites
109.    // maintenant on peut calculer l'impôt
110.    const impôt = Math.floor(revenuImposable * coeffR[i] - nbParts * coeffN[i]);
111.    // résultat
112.    return { "impôt": impôt, "surcôte": surcôte, "taux": coeffR[i] };
113.  }
114.
115.  // revenuImposable=salaireAnnuel-abattement
116.  // l'abattement a un min et un max
117.  getRevenuImposable(salaire) {
118.    // abattement de 10% du salaire
119.    let abattement = 0.1 * salaire;
120.    // cet abattement ne peut dépasser taxAdminData.getAbattementDixPourCentMax()
121.    if (abattement > this.taxAdminData.abattementDixPourCentMax) {
122.      abattement = this.taxAdminData.abattementDixPourcentMax;
123.    }
124.    // l'abattement ne peut être inférieur à taxAdminData.getAbattementDixPourcentMin()
125.    if (abattement < this.taxAdminData.abattementDixPourcentMin) {
126.      abattement = this.taxAdminData.abattementDixPourcentMin;
127.    }
128.    // revenu imposable
129.    const revenuImposable = salaire - abattement;
130.    // résultat
131.    return Math.floor(revenuImposable);
132.  }
133.
134.  // calcule une décôte éventuelle
135.  getDecôte(marié, impots) {
136.    // au départ, une décôte nulle
137.    let décôte = 0;
138.    // montant maximal d'impôt pour avoir la décôte
139.    let plafondImpôtPourDécôte = marié === "oui" ?
140.      this.taxAdminData.plafondImpotCouplePourDecote :
```

```
141.      this.taxAdminData.plafondImpotCelibatairePourDecote;
142.    let plafondDécôte;
143.    if (impots < plafondImpôtPourDécôte) {
144.      // montant maximal de la décôte
145.      plafondDécôte = marié === "oui" ?
146.        this.taxAdminData.plafondDecoteCouple :
147.        this.taxAdminData.plafondDecoteCelibataire;
148.      // décôte théorique
149.      décôte = plafondDécôte - 0.75 * impots;
150.      // la décôte ne peut dépasser le montant de l'impôt
151.      if (décôte > impots) {
152.        décôte = impots;
153.      }
154.      // pas de décôte <0
155.      if (décôte < 0) {
156.        décôte = 0;
157.      }
158.    }
159.    // résultat
160.    return Math.ceil(décôte);
161.  }
162.
163.  // calcule une réduction éventuelle
164.  getRéduction(marié, salaire, enfants, impots) {
165.    // le plafond des revenus pour avoir droit à la réduction de 20%
166.    let plafondRevenuPourRéduction = marié === "oui" ?
167.      this.taxAdminData.plafondRevenusCouplePourReduction :
168.      this.taxAdminData.plafondRevenusCelibatairePourReduction;
169.    plafondRevenuPourRéduction += enfants * this.taxAdminData.valeurReducDemiPart;
170.    if (enfants > 2) {
171.      plafondRevenuPourRéduction += (enfants - 2) * this.taxAdminData.valeurReducDemiPart;
172.    }
173.    // revenu imposable
174.    const revenuImposable = this.getRevenuImposable(salaire);
175.    // réduction
176.    let réduction = 0;
177.    if (revenuImposable < plafondRevenuPourRéduction) {
178.      // réduction de 20%
179.      réduction = 0.2 * impots;
180.    }
181.    // résultat
182.    return Math.ceil(réduction);
183.  }
184.}
185.
186.// export de la classe
187.export default Métier;
```

- le code Javascript suit scrupuleusement le code PHP ;
- la classe **[Métier]** est exportée, ligne 187 ;

14.3.2 La classe Javascript [Dao2]



La classe **[Dao2]** implémente la couche **[dao]** du client Javascript ci-dessus de la façon suivante :

```
1.  'use strict';
```

```
2.
3.  // imports
4.  import qs from 'qs'
5.
6.  class Dao2 {
7.
8.    // constructeur
9.    constructor(axios) {
10.     this.axios = axios;
11.     // cookie de session
12.     this.sessionCookieName = "PHPSESSID";
13.     this.sessionCookie = '';
14.   }
15.
16.   // init session
17.   async  initSession() {
18.     // options de la requête HHTP [get /main.php?action=init-session&type=json]
19.     const options = {
20.       method: "GET",
21.       // paramètres de l'URL
22.       params: {
23.         action: 'init-session',
24.         type: 'json'
25.       }
26.     };
27.     // exécution de la requête HTTP
28.     return await this.getRemoteData(options);
29.   }
30.
31.   async  authentifierUtilisateur(user, password) {
32.     // options de la requête HHTP [post /main.php?action=authentifier-utilisateur]
33.     const options = {
34.       method: "POST",
35.       headers: {
36.         'Content-type': 'application/x-www-form-urlencoded',
37.       },
38.       // corps du POST
39.       data: qs.stringify({
40.         user: user,
41.         password: password
42.       }),
43.       // paramètres de l'URL
44.       params: {
45.         action: 'authentifier-utilisateur'
46.       }
47.     };
48.     // exécution de la requête HTTP
49.     return await this.getRemoteData(options);
50.   }
51.
52.   async getAdminData() {
53.     // options de la requête HHTP [get /main.php?action=get-admindata]
54.     const options = {
55.       method: "GET",
56.       // paramètres de l'URL
57.       params: {
58.         action: 'get-admindata'
59.       }
60.     };
61.     // exécution de la requête HTTP
62.     const data = await this.getRemoteData(options);
63.     // résultat
64.     return data;
65.   }
66.
67.   async  getRemoteData(options) {
68.     // pour le cookie de session
69.     if (!options.headers) {
70.       options.headers = {};
71.     }
72.     options.headers.Cookie = this.sessionCookie;
73.     // exécution de la requête HTTP
74.     let response;
75.     try {
```

```
76.       // requête asynchrone
77.       response = await this.axios.request('main.php', options);
78.     } catch (error) {
79.       // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
80.       if (error.response) {
81.         // la réponse du serveur est dans [error.response]
82.         response = error.response;
83.       } else {
84.         // on relance l'erreur
85.         throw error;
86.       }
87.     }
88.     // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-
    même)
89.     // on récupère le cookie de session s'il existe
90.     const setCookie = response.headers['set-cookie'];
91.     if (setCookie) {
92.       // setCookie est un tableau
93.       // on cherche le cookie de session dans ce tableau
94.       let trouvé = false;
95.       let i = 0;
96.       while (!trouvé && i < setCookie.length) {
97.         // on cherche le cookie de session
98.         const results = RegExp('^(' + this.sessionCookieName + '.+?);').exec(setCookie[i]);
99.         if (results) {
100.          // on mémorise le cookie de session
101.          // eslint-disable-next-line require-atomic-updates
102.          this.sessionCookie = results[1];
103.          // on a trouvé
104.          trouvé = true;
105.        } else {
106.          // élément suivant
107.          i++;
108.        }
109.      }
110.    }
111.    // la réponse du serveur est dans [response.data]
112.    return response.data;
113.  }
114.}
115.
116.// export de la classe
117.export default Dao2;
```

Commentaires

- la classe `[Dao2]` n'implémente que trois des requêtes possibles vers le serveur de calcul d'impôt :
  - `[init-session]` (lignes 17-29) : pour initialiser la session jSON ;
  - `[authentifier-utilisateur]` (lignes 31-50) : pour s'authentifier ;
  - `[get-admindata]` (lignes 52-65) : pour avoir les données de l'administration fiscale qui vont permettre de faire les calculs de l'impôt, côté client ;
- lignes 52-65 : nous introduisons une nouvelle action `[get-admindata]` vers le serveur. Cette action n'était pas jusqu'alors implémentée. Nous le faisons maintenant.

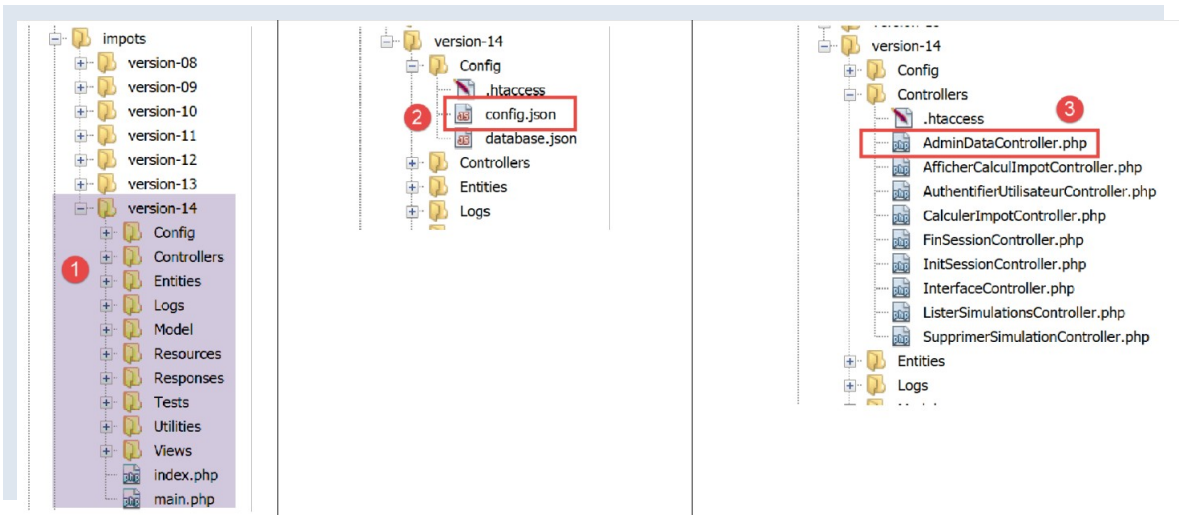
14.3.3 Modification du serveur de calcul de l'impôt

Le serveur de calcul de l'impôt doit implémenter une nouvelle action. Nous allons le faire sur la version 14 du serveur. L'action à implémenter a les caractéristiques suivantes :

- elle est demandée par une opération `[get /main.php?action=get-admindata] ;`
- elle rend la chaîne jSON d'un objet encapsulant les données de l'administration fiscale ;

Nous allons revoir comment ajouter une action à notre serveur.

La modification se fera sous Netbeans :



En **[2]**, nous modifions le fichier **[config.json]** pour ajouter la nouvelle action :

```
{
    "databaseFilename": "Config/database.json",
    "corsAllowed": true,
    "rootDirectory": "C:/myprograms/laragon-lite/www/php7/scripts-web/impots/version-14",
    "relativeDependencies": [

        "/Entities/BaseEntity.php",
        "/Entities/Simulation.php",
        "/Entities/Database.php",
        "/Entities/TaxAdminData.php",
        "/Entities/ExceptionImpots.php",

        "/Utilities/Logger.php",
        "/Utilities/SendAdminMail.php",

        "/Model/InterfaceServerDao.php",
        "/Model/ServerDao.php",
        "/Model/ServerDaoWithSession.php",
        "/Model/InterfaceServerMetier.php",
        "/Model/ServerMetier.php",

        "/Responses/InterfaceResponse.php",
        "/Responses/ParentResponse.php",
        "/Responses/JsonResponse.php",
        "/Responses/XmlResponse.php",
        "/Responses/HtmlResponse.php",

        "/Controllers/InterfaceController.php",
        "/Controllers/InitSessionController.php",
        "/Controllers/ListerSimulationsController.php",
        "/Controllers/AuthentifierUtilisateurController.php",
        "/Controllers/CalculerImpotController.php",
        "/Controllers/SupprimerSimulationController.php",
        "/Controllers/FinSessionController.php",
        "/Controllers/AfficherCalculImpotController.php",
        "/Controllers/AdminDataController.php"
    ],
    "absoluteDependencies": [
        "C:/myprograms/laragon-lite/www/vendor/autoload.php",
        "C:/myprograms/laragon-lite/www/vendor/predis/predis/autoload.php"
    ],
    "users": [
        {
            "login": "admin",
            "passwd": "admin"
        }
    ],
    "adminMail": {
```

```
49.         "smtp-server": "localhost",
50.         "smtp-port": "25",
51.         "from": "guest@localhost",
52.         "to": "guest@localhost",
53.         "subject": "plantage du serveur de calcul d'impôts",
54.         "tls": "FALSE",
55.         "attachments": []
56.     },
57.     "logsFilename": "Logs/logs.txt",
58.     "actions":
59.             {
60.                 "init-session": "\\InitSessionController",
61.                 "authentifier-utilisateur": "\\AuthentifierUtilisateurController",
62.                 "calculer-impot": "\\CalculerImpotController",
63.                 "lister-simulations": "\\ListerSimulationsController",
64.                 "supprimer-simulation": "\\SupprimerSimulationController",
65.                 "fin-session": "\\FinSessionController",
66.                 "afficher-calcul-impot": "\\AfficherCalculImpotController",
67.                 "get-admindata": "\\AdminDataController"
68.             },
69.     "types": {
70.         "json": "\\JsonResponse",
71.         "html": "\\HtmlResponse",
72.         "xml": "\\XmlResponse"
73.     },
74.     "vues": {
75.         "vue-authentification.php": [700, 221, 400],
76.         "vue-calcul-impot.php": [200, 300, 341, 350, 800],
77.         "vue-liste-simulations.php": [500, 600]
78.     },
79.     "vue-erreurs": "vue-erreurs.php"
80. }
```

La modification consiste :

- ligne 67 : ajouter l'action **[get-admindata]** et l'associer à un contrôleur ;
- ligne 36 : déclarer ce contrôleur dans la liste des classes à charger par l'application PHP ;

La phase suivante est d'implémenter le contrôleur **[AdminDataController] [3]** :

```
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.     $erreur = $method !== "get" || $request->query->count() != 1;
28.     if ($erreur) {
29.       // on note l'erreur
30.       $message = "il faut utiliser la méthode [get] avec l'unique paramètre [action] dans
    l'URL";
31.       $état = 1001;
32.       // retour résultat au contrôleur principal
```

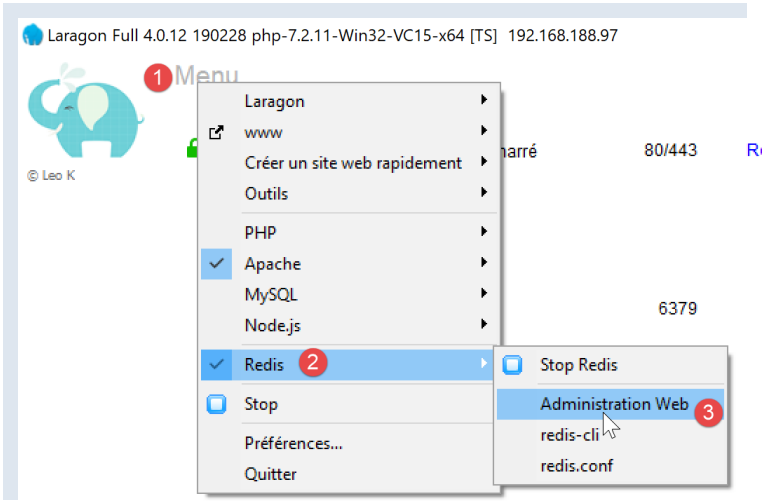
```
33.       return [Response::HTTP_BAD_REQUEST, $état, ["réponse" => $message], []];
34.     }
35.
36.     // on peut travailler
37.     // Redis
38.     \Predis\Autoloader::register();
39.     try {
40.       // client [predis]
41.       $redis = new \Predis\Client();
42.       // on se connecte au serveur pour voir s'il est là
43.       $redis->connect();
44.     } catch (\Predis\Connection\ConnectionException $ex) {
45.       // ça s'est mal passé
46.       // retour résultat avec erreur au contrôleur principal
47.       $état = 1050;
48.       return [Response::HTTP_INTERNAL_SERVER_ERROR, $état,
49.         ["réponse" => "[redis], " . utf8_encode($ex->getMessage())], []];
50.     }
51.
52.     // récupération des données de l'administration fiscale
53.     // on cherche d'abord dans le cache [redis]
54.     if (!$redis->get("taxAdminData")) {
55.       try {
56.         // on va chercher les données fiscales en base de données
57.         $dao = new ServerDaoWithRedis($config["databaseFilename"], NULL);
58.         // taxAdminData
59.         $taxAdminData = $dao->getTaxAdminData();
60.         // on met dans redis les données récupérées
61.         $redis->set("taxAdminData", $taxAdminData);
62.       } catch (\RuntimeException $ex) {
63.         // ça s'est mal passé
64.         // retour résultat avec erreur au contrôleur principal
65.         $état = 1041;
66.         return [Response::HTTP_INTERNAL_SERVER_ERROR, $état,
67.           ["réponse" => utf8_encode($ex->getMessage())], []];
68.       }
69.     } else {
70.       // les données fiscales sont prises dans la mémoire [redis] de portée [application]
71.       $arrayOfAttributes = \json_decode($redis->get("taxAdminData"), true);
72.       // on instancie un objet [TaxAdminData] à partir du tableau d'attributs précédent
73.       $taxAdminData = (new TaxAdminData())->setFromArrayOfAttributes($arrayOfAttributes);
74.     }
75.
76.     // retour résultat au contrôleur principal
77.     $état = 1000;
78.     return [Response::HTTP_OK, $état, ["réponse" => $taxAdminData], []];
79.   }
80.
81. }
```

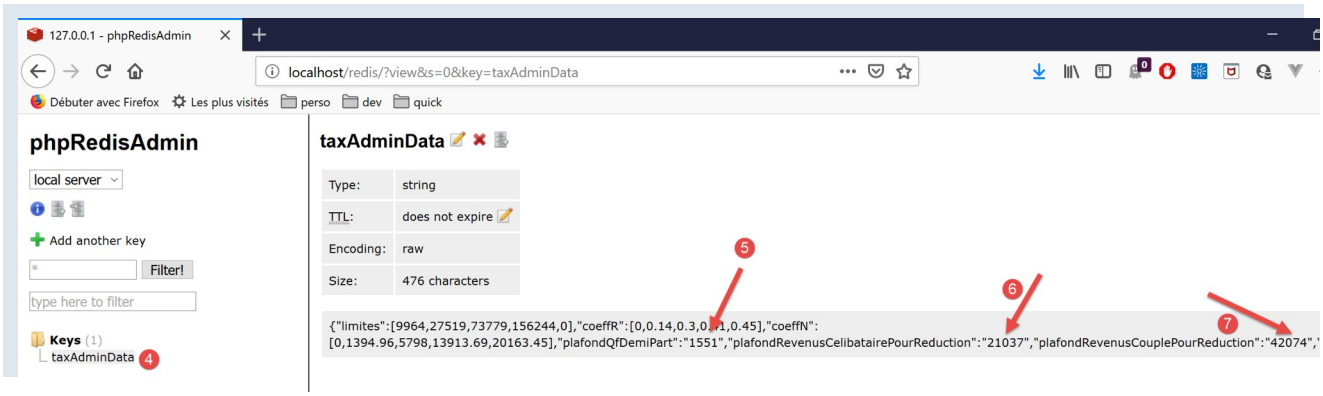
Commentaires

- ligne 12 : comme les autres contrôleurs du serveur, `[AdminDataController]` implémente l'interface `[Interface-Controller]` constituée par la méthode `[execute]` des lignes 19-79 ;
- ligne 78 : comme pour les autres contrôleurs du serveur, la méthode `[AdminDataController.execute]` rend un tableau `[$status, $état, ['réponse'=>$response]]` avec :
  - `[$status]` : le code de statut de la réponse HTTP ;
  - `[$état]` : un code interne à l'application représentant l'état dans lequel se trouve le serveur après exécution de la requête du client ;
  - `[$response]` : un tableau encapsulant la réponse à envoyer au client. Ici, ce tableau sera ultérieurement transformé en chaîne jSON ;
- lignes 25-34 : on vérifie que l'action `[get-admindata]` du client est syntaxiquement correcte ;
- lignes 37-74 : on récupère un objet `[TaxAdminData]` trouvé soit :
  - lignes 56-59 : dans la base de données si on ne l'a pas trouvé dans le cache `[redis]` ;
  - lignes 70-73 : dans le cache `[redis]` ;

Ce code reprend celui du contrôleur `[CalculerImpotController]` expliqué dans l'article lien. En effet, ce contrôleur devait lui aussi récupérer l'objet `[TaxAdminData]` encapsulant les données de l'administration fiscale.

Lors des tests du client Javascript, la forme jSON de `[TaxAdminData]` a posé problème lorsque cet objet était trouvé dans le cache `[redis]`. Pour le comprendre, examinons sous quelle forme cet objet est stocké dans `[redis]` :





- en `[5-7]`, on voit que des valeurs numériques ont été stockées sous forme de chaînes de caractères. PHP s'en est accommodé car l'opérateur + dans les calculs entre nombres et chaînes provoque implicitement un changement de type de la chaîne vers un nombre. Mais Javascript fait le contraire : l'opérateur + dans les calculs entre nombres et chaînes provoque implicitement un changement de type du nombre vers une chaîne de caractères. Les calculs de la classe Javascript `[Métier]` sont alors erronés ;

Pour remédier à ce problème, nous modifions la méthode `[TaxAdminData.setFromArrayOfAttributes]` utilisée ligne 71 du contrôleur pour instancier un objet `[TaxAdminData]` (cf. article) à partir de la chaîne jSON trouvée dans le cache `[redis]` :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  class TaxAdminData extends BaseEntity {
6.    // tranches d'impôt
7.    protected $limites;
8.    protected $coeffR;
9.    protected $coeffN;
10.   // constantes de calcul de l'impôt
11.   protected $plafondQfDemiPart;
12.   protected $plafondRevenusCelibatairePourReduction;
13.   protected $plafondRevenusCouplePourReduction;
14.   protected $valeurReducDemiPart;
```

```
15.   protected $plafondDecoteCelibataire;
16.   protected $plafondDecoteCouple;
17.   protected $plafondImpotCouplePourDecote;
18.   protected $plafondImpotCelibatairePourDecote;
19.   protected $abattementDixPourcentMax;
20.   protected $abattementDixPourcentMin;
21.
22.   // initialisation
23.   public function setFromJsonFile(string $taxAdminDataFilename) {
24.     // parent
25.     parent::setFromJsonFile($taxAdminDataFilename);
26.     // on vérifie les valeurs des attributs
27.     $this->checkAttributes();
28.     // on rend l'objet
29.     return $this;
30.   }
31.
32.   protected function check($value): \stdClass {
33.     // $value est un tableau d'éléments de type string ou un unique élément
34.     if (!\is_array($value)) {
35.       $tableau = [$value];
36.     } else {
37.       $tableau = $value;
38.     }
39.     // on transforme le tableau de strings en tableau de réels
40.     $newTableau = [];
41.     $result = new \stdClass();
42.     // les éléments du tableau doivent être des nombres décimaux positifs ou nuls
43.     $modèle = '/^\s*([+]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/';
44.     for ($i = 0; $i < count($tableau); $i ++) {
45.       if (preg_match($modèle, $tableau[$i])) {
46.         // on met le float dans newTableau
47.         $newTableau[] = (float) $tableau[$i];
48.       } else {
49.         // on note l'erreur
50.         $result->erreur = TRUE;
51.         // on quitte
52.         return $result;
53.       }
54.     }
55.     // on rend le résultat
56.     $result->erreur = FALSE;
57.     if (!\is_array($value)) {
58.       // une seule valeur
59.       $result->value = $newTableau[0];
60.     } else {
61.       // une liste de valeurs
62.       $result->value = $newTableau;
63.     }
64.     return $result;
65.   }
66.
67.   // initialisation par un tableau d'attributs
68.   public function setFromArrayOfAttributes(array $arrayOfAttributes) {
69.     // parent
70.     parent::setFromArrayOfAttributes($arrayOfAttributes);
71.     // on vérifie les valeurs des attributs
72.     $this->checkAttributes();
73.     // on rend l'objet
74.     return $this;
75.   }
76.
77.   // vérification des valeurs des attributs
78.   protected function checkAttributes() {
79.     // on vérifie que les valeurs des attributs sont des réels >=0
80.     foreach ($this as $key => $value) {
81.       if (is_string($value)) {
82.         // $value doit être un nbre réel >=0 ou un tableau de réels >=0
83.         $result = $this->check($value);
84.         // erreur ?
85.         if ($result->erreur) {
86.           // on lance une exception
87.           throw new ExceptionImpots("La valeur de l'attribut [$key] est invalide");
88.         } else {
```

```
89.           // on note la valeur
90.           $this->$key = $result->value;
91.         }
92.       }
93.     }
94.
95.     // on rend l'objet
96.     return $this;
97.   }
98.
99.   // getters et setters
100.  ...
101.
102.}
```

Commentaires

- ligne 5 : la classe **[TaxAdminData]** étend la classe **[BaseEntity]** qui a déjà la méthode **[setFromArrayOfAttributes]**. Celle-ci ne convenant pas, nous la redéfinissons aux lignes 67-75 ;
- ligne 70 : la méthode **[setFromArrayOfAttributes]** de la classe parent est d'abord utilisée pour initialiser les attributs de la classe ;
- ligne 72 : la méthode **[checkAttributes]** vérifie que les valeurs associées sont bien des nombres. Si ce sont des chaînes, celles-ci sont converties en nombres ;
- ligne 74 : l'objet **[$this]** rendu est alors un objet avec des attributs à valeurs numériques ;
- lignes 78-93 : la méthode **[checkAttributes]** vérifie que les valeurs associées aux attributs de l'objet sont bien numériques ;
- ligne 80 : on parcourt la liste des attributs ;
- ligne 81 : si la valeur d'un attribut est de type **[string]** ;
- ligne 83 : alors on vérifie que cette chaîne représente un nombre ;
- ligne 90 : si c'est le cas, la chaîne est transformée en nombre et affectée à l'attribut testé ;
- lignes 85-86 : si ce n'est pas le cas, une exception est lancée ;
- lignes 32-65 : la fonction **[check]** fait un peu plus que nécessaire. Elle traite aussi bien des tableaux que des valeurs uniques. Or ici, elle n'est appelée que pour vérifier une valeur de type **[string]**. Elle rend un objet avec les propriétés **[erreur, value]** où :
  - **[erreur]** est un booléen signalant une erreur ou non ;
  - **[value]** est le paramètre **[value]** de la ligne 32, transformée en nombre ou tableau de nombres selon les cas ;

La classe **[BaseEntity]** qui pouvait avoir un attribut nommé **[arrayOfAttributes]** est modifiée pour ne plus avoir celui-ci : il pollue en effet la chaîne jSON de **[TaxAdminData]**. La classe est réécrite de la façon suivante :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  class BaseEntity {
6.
7.    // initialisation à partir d'un fichier JSON
8.    public function setFromJsonFile(string $jsonFilename) {
9.      // on récupère le contenu du fichier des données fiscales
10.     $fileContents = \file_get_contents($jsonFilename);
11.     $erreur = FALSE;
12.     // erreur ?
13.     if (!$fileContents) {
14.       // on note l'erreur
15.       $erreur = TRUE;
16.       $message = "Le fichier des données [$jsonFilename] n'existe pas";
17.     }
18.     if (!$erreur) {
19.       // on récupère le code JSON du fichier de configuration dans un tableau associatif
20.       $arrayOfAttributes = \json_decode($fileContents, true);
21.       // erreur ?
22.       if ($arrayOfAttributes === FALSE) {
23.         // on note l'erreur
24.         $erreur = TRUE;
25.         $message = "Le fichier de données JSON [$jsonFilename] n'a pu être exploité
    correctement";
26.       }
27.     }
```

```
28.     // erreur ?
29.     if ($erreur) {
30.       // on lance une exception
31.       throw new ExceptionImpots($message);
32.     }
33.     // initialisation des attributs de la classe
34.     foreach ($arrayOfAttributes as $key => $value) {
35.       $this->$key = $value;
36.     }
37.     // on vérifie la présence de tous les attributs
38.     $this->checkForAllAttributes($arrayOfAttributes);
39.     // on rend l'objet
40.     return $this;
41.   }
42.
43.   public function checkForAllAttributes($arrayOfAttributes) {
44.     // on vérifie que toutes les clés ont été initialisées
45.     foreach (\array_keys($arrayOfAttributes) as $key) {
46.       if (!isset($this->$key)) {
47.         throw new ExceptionImpots("L'attribut [$key] de la classe "
48.           . get_class($this) . " n'a pas été initialisé");
49.       }
50.     }
51.   }
52.
53.   public function setFromArrayOfAttributes(array $arrayOfAttributes) {
54.     // on initialise certains attributs de la classe (pas forcément tous)
55.     foreach ($arrayOfAttributes as $key => $value) {
56.       $this->$key = $value;
57.     }
58.     // on retourne l'objet
59.     return $this;
60.   }
61.
62.   // toString
63.   public function __toString() {
64.     // attributs de l'objet
65.     $arrayOfAttributes = \get_object_vars($this);
66.     // chaîne jSON de l'objet
67.     return \json_encode($arrayOfAttributes, JSON_UNESCAPED_UNICODE);
68.   }
69.
70. }
```

Commentaires

- ligne 20 : l'attribut **[$this→arrayOfAttributes]** a été transformée en variable qui doit être désormais passée à la méthode **[checkForAllAttributes]**, ligne 38 qui auparavant opérait sur l'attribut **[$this→arrayOfAttributes]** ;

A cause de ce changement sur **[BaseEntity]**, la classe **[Database]** doit être également légèrement modifiée :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  class Database extends BaseEntity {
6.    // attributs
7.    protected $dsn;
8.    protected $id;
9.    protected $pwd;
10.   protected $tableTranches;
11.   protected $colLimites;
12.   protected $colCoeffR;
13.   protected $colCoeffN;
14.   protected $tableConstantes;
15.   protected $colPlafondQfDemiPart;
16.   protected $colPlafondRevenusCelibatairePourReduction;
17.   protected $colPlafondRevenusCouplePourReduction;
18.   protected $colValeurReducDemiPart;
19.   protected $colPlafondDecoteCelibataire;
20.   protected $colPlafondDecoteCouple;
21.   protected $colPlafondImpotCelibatairePourDecote;
22.   protected $colPlafondImpotCouplePourDecote;
```

```
23.   protected $colAbattementDixPourcentMax;
24.   protected $colAbattementDixPourcentMin;
25.
26.   // setter
27.   // initialisation
28.   public function setFromJsonFile(string $jsonFilename) {
29.     // parent
30.     parent::setFromJsonFile($jsonFilename);
31.     // on retourne l'objet
32.     return $this;
33.   }
34.
35.   // getters et setters
36.   ...
37. }
```

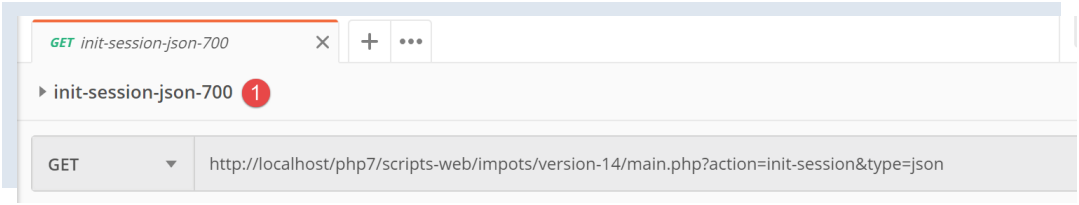
Commentaires

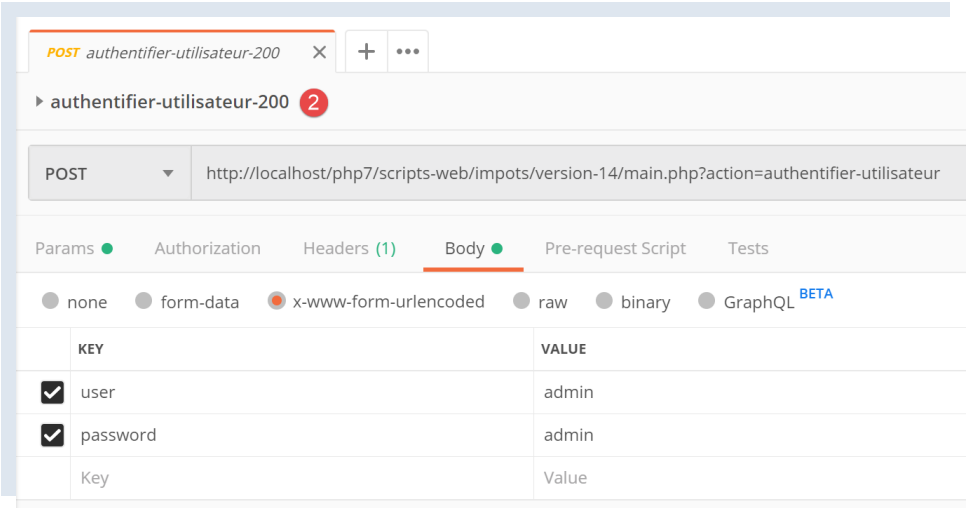
- dans le code original, après la ligne 30, on appelait la méthode **`[parent::checkForAllAttributes]`**. Cela n'a plus à être fait puisque c'est désormais pris automatiquement en charge par la méthode **`[parent::setFromJsonFile($jsonFilename)]`** ;

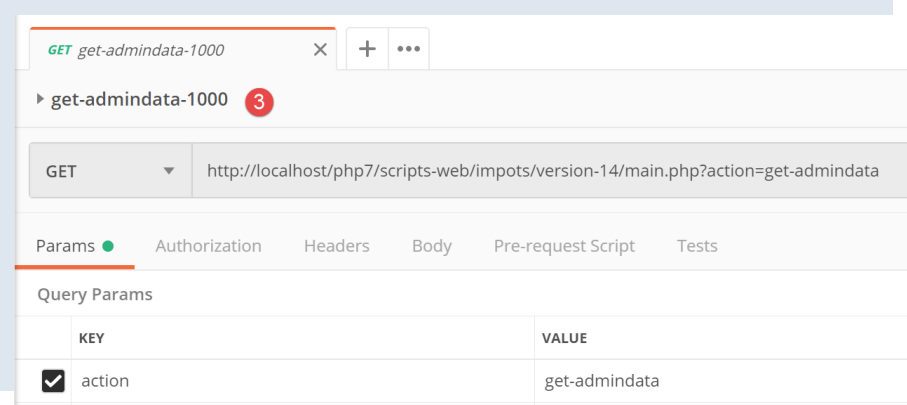
14.3.4 Tests `[Postman]` du serveur

`[Postman]` a été présenté dans l'article lien.

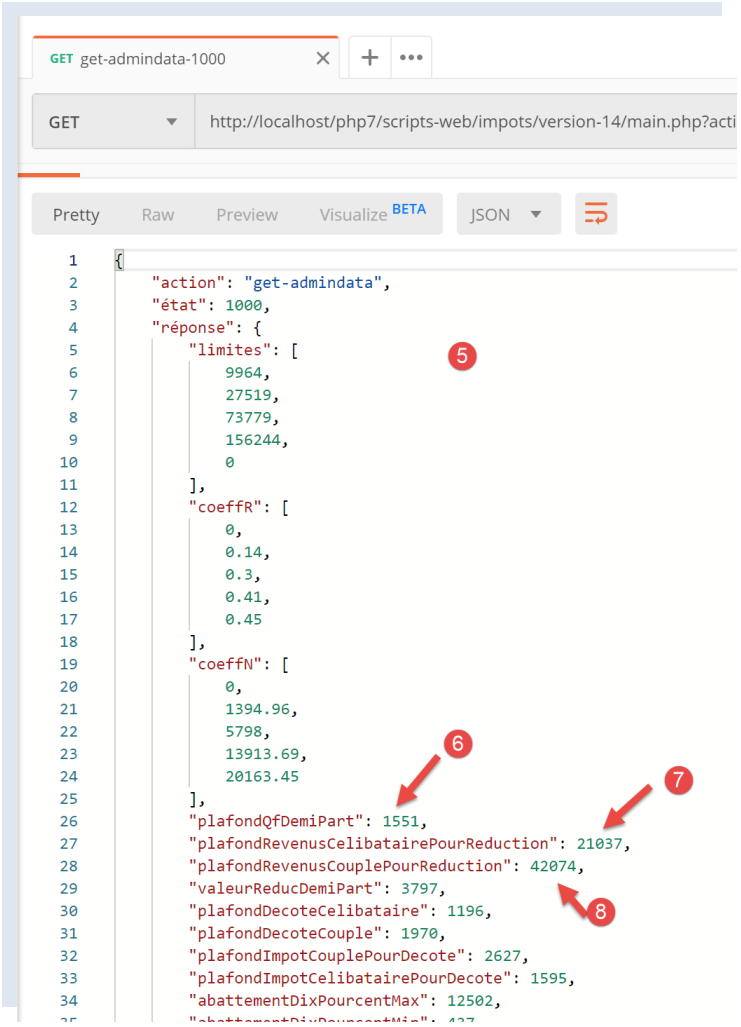
Nous utilisons les tests Postman suivants :





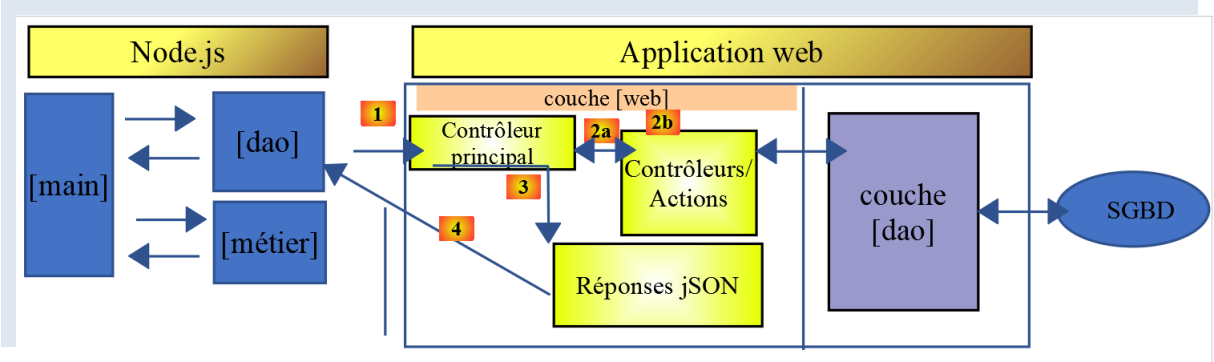


Le résultat jSON de cette dernière requête est la suivante :



- en **[5-8]**, on peut remarquer que les attributs de la chaîne jSON ont bien des valeurs numériques (et non chaînes de caractères). Ce résultat va permettre à la classe Javascript **[Métier]** de s'exécuter normalement ;

14.3.5 Le script principal [main]



Le script principal **[main]** du client Javascript est le suivant :

```
1.  // imports
2.  import axios from 'axios';
3.
4.  // imports
5.  import Dao from './Dao2';
6.  import Métier from './Métier';
7.
8.  // fonction asynchrone [main]
9.  async function main() {
10.   // configuration axios
11.   axios.defaults.timeout = 2000;
12.   axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';
13.   // instanciation couche [dao]
14.   const dao = new Dao(axios);
15.   // requêtes HTTP
16.   let taxAdminData;
17.   try {
18.     // init session
19.     log("-----------init-session");
20.     let response = await dao.initSession();
21.     log(response);
22.     // authentification
23.     log("-----------authentifier-utilisateur");
24.     response = await dao.authentifierUtilisateur("admin", "admin");
25.     log(response);
26.     // données fiscales
27.     log("-----------get-admindata");
28.     response = await dao.getAdminData();
29.     log(response);
30.     taxAdminData = response.réponse;
31.   } catch (error) {
32.     // on logue l'erreur
33.     console.log("erreur=", error.message);
34.     // fin
35.     return;
36.   }
37.
38.   // instanciation couche [métier]
39.   const métier = new Métier(taxAdminData);
40.
41.   // calculs d'impôt
42.   log("-----------calculer-impot x 3");
43.   const simulations = [];
44.   simulations.push(métier.calculerImpot("oui", 2, 45000));
45.   simulations.push(métier.calculerImpot("non", 2, 45000));
46.   simulations.push(métier.calculerImpot("non", 1, 30000));
47.   // liste des simulations
48.   log("-----------liste-des-simulations");
49.   log(simulations);
50.   // suppression d'une simulation
51.   log("-----------suppression simulation n° 1");
52.   simulations.splice(1, 1);
53.   log(simulations);
54. }
```

```
55.
56. // log jSON
57. function log(object) {
58.   console.log(JSON.stringify(object, null, 2));
59. }
60.
61. // exécution
62. main();
```

Commentaires

- lignes 5-6 : imports des classes **[Dao]** et **[Métier]** ;
- ligne 9 : la fonction asynchrone **[main]** qui va organiser le dialogue avec le serveur grâce à la classe **[Dao]** et demander à la classe **[Métier]** de faire les calculs d'impôt ;
- lignes 10-36 : le script appelle successivement et de façon bloquante, les méthodes **[initSession, authentifierUtilisateur, getAdminData]** de la couche **[dao]** ;
- ligne 38 : on n'a plus besoin de la couche **[dao]**. On a tous les éléments pour faire travailler la couche **[métier]** du client Javascript ;
- lignes 41-46 : on fait trois calculs d'impôt dont on cumule les résultats dans un tableau **[simulations]** ;
- ligne 49 : on affiche le tableau des simulations ;
- ligne 52 : on supprime l'une d'elles ;

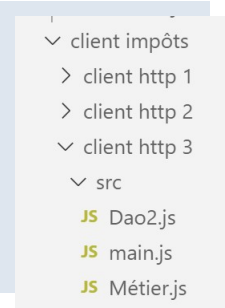
Les résultats de l'exécution du script principal sont les suivants :

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:
    \Data\st-2019\dev\es6\javascript\client impôts\client http 2\main2.js"
2.  "-----------init-session"
3.  {
4.    "action": "init-session",
5.    "état": 700,
6.    "réponse": "session démarrée avec type [json]"
7.  }
8.  "-----------authentifier-utilisateur"
9.  {
10.   "action": "authentifier-utilisateur",
11.   "état": 200,
12.   "réponse": "Authentification réussie [admin, admin]"
13. }
14. "-----------get-admindata"
15. {
16.   "action": "get-admindata",
17.   "état": 1000,
18.   "réponse": {
19.     "limites": [
20.       9964,
21.       27519,
22.       73779,
23.       156244,
24.       0
25.     ],
26.     "coeffR": [
27.       0,
28.       0.14,
29.       0.3,
30.       0.41,
31.       0.45
32.     ],
33.     "coeffN": [
34.       0,
35.       1394.96,
36.       5798,
37.       13913.69,
38.       20163.45
39.     ],
40.     "plafondQfDemiPart": 1551,
41.     "plafondRevenusCelibatairePourReduction": 21037,
42.     "plafondRevenusCouplePourReduction": 42074,
43.     "valeurReducDemiPart": 3797,
44.     "plafondDecoteCelibataire": 1196,
45.     "plafondDecoteCouple": 1970,
46.     "plafondImpotCouplePourDecote": 2627,
```

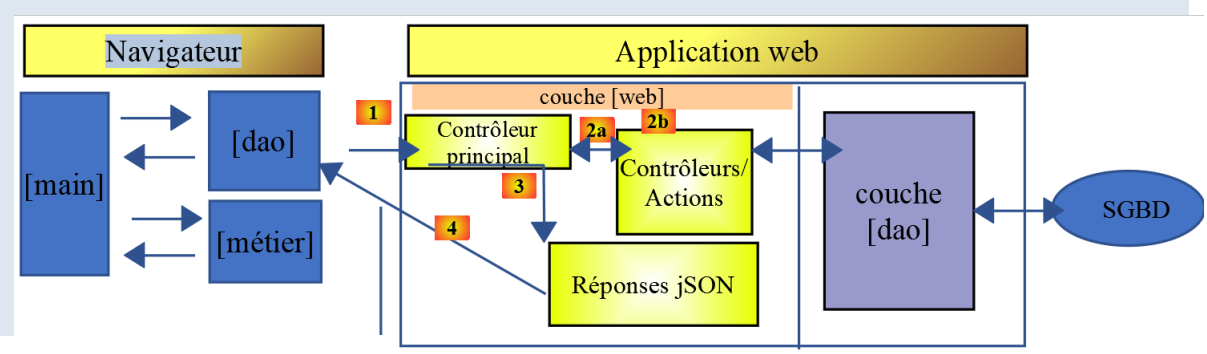
```
    "plafondImpotCelibatairePourDecote": 1595,
    "abattementDixPourcentMax": 12502,
    "abattementDixPourcentMin": 437
  }
}
"-----------calculer-impot x 3"
"-----------liste-des-simulations"
[
  {
    "impôt": 502,
    "surcôte": 0,
    "décôte": 857,
    "réduction": 126,
    "taux": 0.14
  },
  {
    "impôt": 3250,
    "surcôte": 370,
    "décôte": 0,
    "réduction": 0,
    "taux": 0.3
  },
  {
    "impôt": 1687,
    "surcôte": 0,
    "décôte": 0,
    "réduction": 0,
    "taux": 0.14
  }
]
"-----------suppression simulation n° 1"
[
  {
    "impôt": 502,
    "surcôte": 0,
    "décôte": 857,
    "réduction": 126,
    "taux": 0.14
  },
  {
    "impôt": 1687,
    "surcôte": 0,
    "décôte": 0,
    "réduction": 0,
    "taux": 0.14
  }
]

[Done] exited with code=0 in 0.583 seconds
```

14.4 Client HTTP 3



Dans cette section, nous portons l'application **`[Client HTTP 2]`** dans un navigateur selon l'architecture suivane :



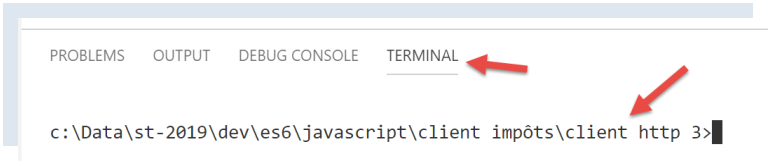
Le portage n'est pas immédiat. Si **[node.js]** sait exécuter du Javascript ES6, ce n'est pas le cas en général des navigateurs. Il faut alors utiliser des outils qui traduisent le code ES6 en code ES5 compris par les navigateurs récents. Heureusement ces outils sont à la fois puissants et plutôt simples d'utilisation.

Nous avons ici suivi l'article **[How to write ES6 code that's safe to run in the browser - Web Developer's Journal]**.

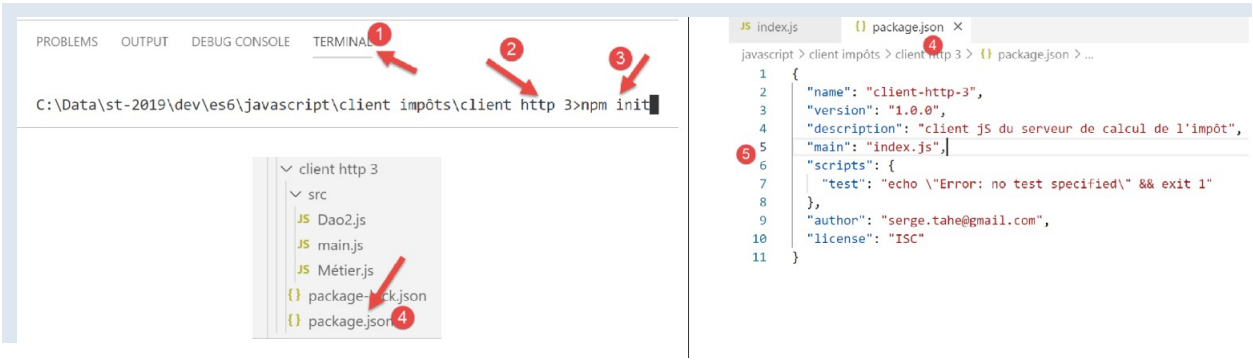
Dans le dossier **[client HTTP 3/src]**, on a mis les éléments **[main.js, Métier.js, Dao2.js]** de l'application **[Client Http 2]** que nous venons de développer.

14.4.1 Initialisation du projet

Nous allons travailler dans le dossier **[client http 3]**. Nous ouvrons un terminal dans **[VSCode]** et nous nous positionnons sur ce dossier :



Nous initialisons ce projet avec la commande **[npm init]** et nous acceptons pour les questions posées les réponses proposées par défaut :



- en **[4-5]**, le fichier de configuration du projet **[package.json]** généré à partir des différentes réponses données ;

14.4.2 Installation des dépendances du projet

Nous allons installer les dépendances suivantes :

- **[@babel/core]** : le coeur de l'outil **[Babel] [https://babeljs.io]** qui transforme du code ES 2015+ en code exécutable sur les navigateurs récents et plus anciens ;

- **[@babel/preset-env]** : fait partie de l'outillage Babel. Intervient avant la transpilation ES6 → ES5 ;
- **[babel-loader]** : cette dépendance permet à l'outil **[webpack]** de faire appel à l'outil **[Babel]** ;
- **[webpack]** : chef d'orchestre. C'est **[webpack]** qui fait appel à Babel pour faire la transpilation des codes ES6 → ES5 puis lui qui assemble la totalité des fichiers résultants dans un unique fichier ;
- **[webpack-cli]** : nécessaire à **[webpack]** ;
- **[@webpack-cli/init]** : utilisé pour configurer **[webpack]** ;
- **[webpack-dev-server]** : fournit un serveur web de développement opérant par défaut sur le port 8080. Lorsque les fichiers sources sont modifiés, recharge automatiquement l'application web ;

Les dépendances du projet sont installées de la façon suivante dans un terminal de **[VSCode]** :

```
npm --save-dev install @babel/core @babel/preset-env babel-loader webpack webpack-cli webpack-dev-server @webpack-cli/init
```

Après l'installation des dépendances, le fichier **[package.json]** a évolué de la façon suivante :

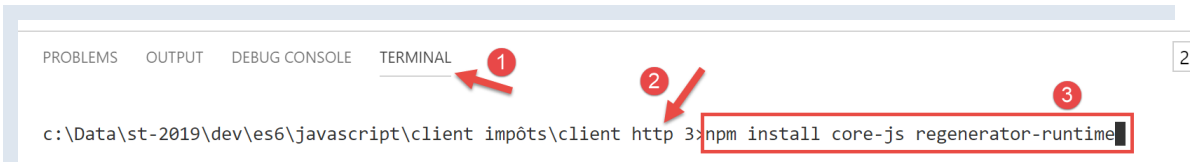
```
1.  {
2.    "name": "client-http-3",
3.    "version": "1.0.0",
4.    "description": "client jS du serveur de calcul de l'impôt",
5.    "main": "index.js",
6.    "scripts": {
7.      "test": "echo \"Error: no test specified\" && exit 1"
8.    },
9.    "author": "serge.tahe@gmail.com",
10.   "license": "ISC",
11.   "devDependencies": {
12.     "@babel/core": "^7.6.0",
13.     "@babel/preset-env": "^7.6.0",
14.     "@webpack-cli/init": "^0.2.2",
15.     "babel-loader": "^8.0.6",
16.     "cross-env": "^6.0.0",
17.     "webpack": "^4.40.2",
18.     "webpack-cli": "^3.3.9",
19.     "webpack-dev-server": "^3.8.1"
20.   }
21. }
```

- lignes 12-19 : les dépendances du projet sont des **[devDependencies]** : on en a besoin pendant la phase de développement mais plus dans la phase de production. En effet, en production, c'est le fichier **[dist/main.js]** qui est utilisé. Il est codé en ES5 et n'a plus besoin des outils de transpilation de code ES6 vers du code ES5 ;

Il nous faut ajouter deux dépendances au projet :

- **[core-js]** : contient des « polyfills » pour ECMAScript 2019. Un polyfill permet d'exécuter un code récent, comme ECMAScript 2019 (sept 2019), sur des navigateurs anciens ;
- **[regenerator-runtime]** : selon le site de la bibliothèque --> **[Source transformer enabling ECMAScript 6 generator functions in JavaScript-of-today]** ;

Ces deux dépendances remplacent, à partir de Babel 7, la dépendance **[@babel/polyfill]** qui jouait auparavant ce rôle et qui est maintenant (sept 2019) dépréciée. Elles sont installées de la façon suivante :



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

```
1.  {
2.    "name": "client-http-3",
3.    "version": "1.0.0",
4.    "description": "My webpack project",
5.    "main": "index.js",
6.    "scripts": {
7.      "test": "echo \"Error: no test specified\" && exit 1",
8.      "build": "webpack",
9.      "start": "webpack-dev-server"
10.   },
11.   "author": "serge.tahe@gmail.com",
12.   "license": "ISC",
13.   "devDependencies": {
14.     "@babel/core": "^7.6.0",
15.     "@babel/preset-env": "^7.6.0",
16.     "@webpack-cli/init": "^0.2.2",
17.     "babel-loader": "^8.0.6",
18.     "babel-plugin-syntax-dynamic-import": "^6.18.0",
19.     "html-webpack-plugin": "^3.2.0",
20.     "webpack": "^4.40.2",
21.     "webpack-cli": "^3.3.9",
22.     "webpack-dev-server": "^3.8.1"
23.   },
24.   "dependencies": {
25.     "core-js": "^3.2.1",
26.     "regenerator-runtime": "^0.13.3"
27.   }
28. }
```

L'utilisation des dépendances **[core-js, regenerator-runtime]** impose de mettre les **[imports]** suivants (lignes 3-4) dans le script principal **[src/main.js]** :

```
1.  // imports
2.  import axios from 'axios';
3.  import "core-js/stable";
4.  import "regenerator-runtime/runtime";
5.
6.  // imports
7.  import Dao from './Dao2';
8.  import Métier from './Métier';
```

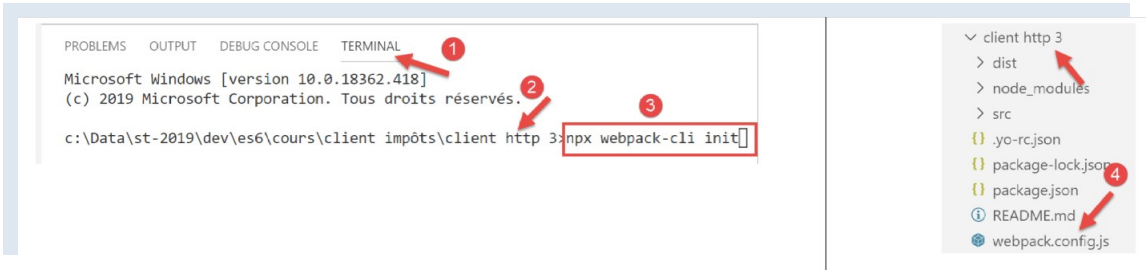
14.4.3 Configuration de [webpack]

[webpack] est l'outil qui va piloter :

- la transpilation ES6 → ES5 de tous les fichiers Javascript du projet ;
- l'assemblage des fichiers générés dans un unique fichier ;

Cet outil est piloté par un fichier de configuration **[webpack.config.js]** qui peut être généré grâce à une dépendance nommée **[@webpack-cli/init]** (sept 2019). Celle-ci a été installée avec les autres au paragraphe lien.

Nous exécutons la commande **[npx webpack-cli init]** dans un terminal **[VSCode]** :



Après avoir répondu aux différentes questions (dont on peut accepter la plupart des réponses proposées par défaut), un fichier **[webpack.config.js]** est généré à la racine du projet **[4]** :

Le fichier **[webpack.config.js]** ressemble à ceci :

```
1.  /* eslint-disable */
2.
3.  const path = require('path');
4.  const webpack = require('webpack');
5.
6.  /*
7.   * SplitChunksPlugin is enabled by default and replaced
8.   * deprecated CommonsChunkPlugin. It automatically identifies modules which
9.   * should be splitted of chunk by heuristics using module duplication count and
10.  * module category (i. e. node_modules). And splits the chunks…
11.  *
12.  * It is safe to remove "splitChunks" from the generated configuration
13.  * and was added as an educational example.
14.  *
15.  * https://webpack.js.org/plugins/split-chunks-plugin/
16.  *
17.  */
18.
19. const HtmlWebpackPlugin = require('html-webpack-plugin');
20.
21. /*
22.  * We've enabled HtmlWebpackPlugin for you! This generates a html
23.  * page for you when you compile webpack, which will make you start
24.  * developing and prototyping faster.
25.  *
26.  * https://github.com/jantimon/html-webpack-plugin
27.  *
28.  */
29.
30. module.exports = {
31.         mode: 'development',
32.         entry: './src/index.js',
33.
34.         output: {
35.                 filename: '[name].[chunkhash].js',
36.                 path: path.resolve(__dirname, 'dist')
37.         },
38.
39.         plugins: [new webpack.ProgressPlugin(), new HtmlWebpackPlugin()],
40.
41.         module: {
42.                 rules: [
43.                         {
44.                                 test: /.(js|jsx)$/,
45.                                 include: [path.resolve(__dirname, 'src')],
46.                                 loader: 'babel-loader',
47.
48.                                 options: {
49.                                         plugins: ['syntax-dynamic-import'],
50.
51.                                         presets: [
52.                                                 [
53.                                                         '@babel/preset-env',
54.                                                         {
55.                                                                 modules: false
56.                                                         }
57.                                                 ]
58.                                         ]
59.                                 }
60.                         }
61.                 ]
62.         },
63.
64.         optimization: {
65.                 splitChunks: {
66.                         cacheGroups: {
67.                                 vendors: {
68.                                         priority: -10,
69.                                         test: /[\\/]node_modules[\\/]/
70.                                 }
71.                         },
```

```
72.
73.                         chunks: 'async',
74.                         minChunks: 1,
75.                         minSize: 30000,
76.                         name: true
77.                 }
78.         },
79.
80.         devServer: {
81.                 open: true
82.         }
83. };
```

Je ne comprends pas tous les détails de ce fichier mais on peut remarquer quelques points :

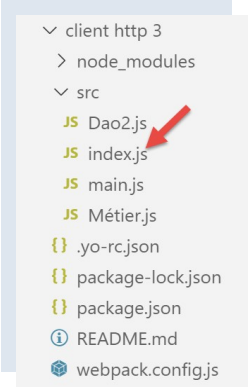
- ligne 1 : le fichier ne contient pas du code ES6. **[Eslint]** déclare alors des erreurs qui remontent jusqu'à la racine du projet **[javascript]**. C'est gênant. Pour éviter qu'Eslint n'analyse un fichier, il suffit de mettre le commentaire de la ligne 1 ;
- ligne 31 : on travaille en mode **[développement]** ;
- ligne 32 : le script d'entrée est ici **[src/index.js]**. Nous serons amenés à changer cela ;
- ligne 36 : le dossier où seront tockées les produits de **[webpack]** sera le dossier **[dist]** ;
- ligne 46 : on voit que **[webpack]** utilise **[babel-loader]**, une des dépendances que nous avons installées ;
- ligne 54 : on voit que **[webpack]** utilise **[@babel-preset/env]**, une des dépendances que nous avons installées ;

L'initialisation de **[webpack]** a modifié le fichier **[package.json]** (il demande l'autorisation) :

```
1.  {
2.    "name": "client-http-3",
3.    "version": "1.0.0",
4.    "description": "My webpack project",
5.    "main": "index.js",
6.    "scripts": {
7.      "test": "echo \"Error: no test specified\" && exit 1",
8.      "build": "webpack",
9.      "start": "webpack-dev-server"
10.   },
11.   "author": "serge.tahe@gmail.com",
12.   "license": "ISC",
13.   "devDependencies": {
14.     "@babel/core": "^7.6.0",
15.     "@babel/preset-env": "^7.6.0",
16.     "@webpack-cli/init": "^0.2.2",
17.     "babel-loader": "^8.0.6",
18.     "babel-plugin-syntax-dynamic-import": "^6.18.0",
19.     "html-webpack-plugin": "^3.2.0",
20.     "webpack": "^4.40.2",
21.     "webpack-cli": "^3.3.9",
22.     "webpack-dev-server": "^3.8.1"
23.   },
24.   "dependencies": {
25.     "core-js": "^3.2.1",
26.     "regenerator-runtime": "^0.13.3"
27.   }
28. }
```

- ligne 4 : elle a été modifiée ;
- lignes 8-9, 18-19 : elles ont été ajoutées ;
- ligne 8 : la tâche **[npm]** qui permet de compiler le projet ;
- ligne 9 : la tâche **[npm]** qui permet de l'exécuter ;
- ligne 18 : ?
- ligne 19 : permet la génération d'un fichier **[dist/index.html]** embarquant automatiquement le script **[dist/main.js]** généré par **[webpack]** et c'est celui-ci qui est exploité lorsque le projet est exécuté ;

Enfin la configuration de **[webpack]** a généré un fichier **[src/index.js]** :



Le contenu de **[index.js]** est le suivant (sept 2019) :

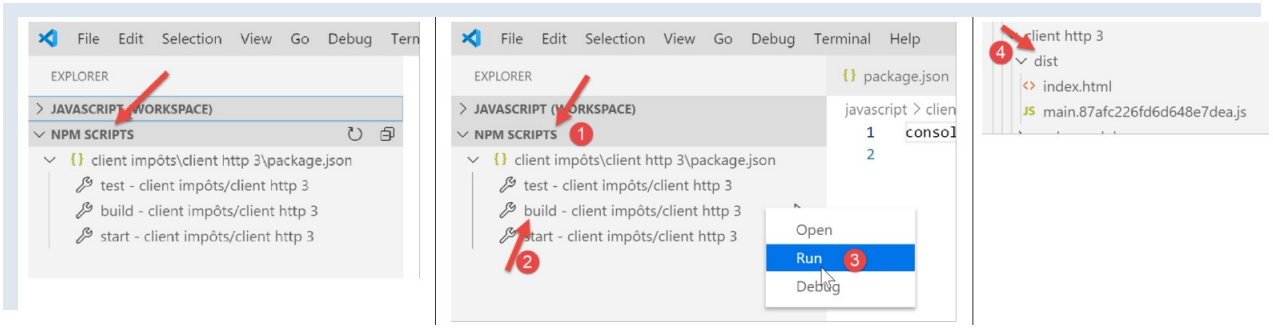
```
1.  console.log("Hello World from your main file!");
```

14.4.4 Compilation et exécution du projet

Le fichier **[package.json]** a trois tâches **[npm]** :

```
1.  "scripts": {
2.      "test": "echo \"Error: no test specified\" && exit 1",
3.      "build": "webpack",
4.      "start": "webpack-dev-server"
5.  },
```

Ces tâches sont comprises par **[VSCode]** qui les propose à l'exécution :



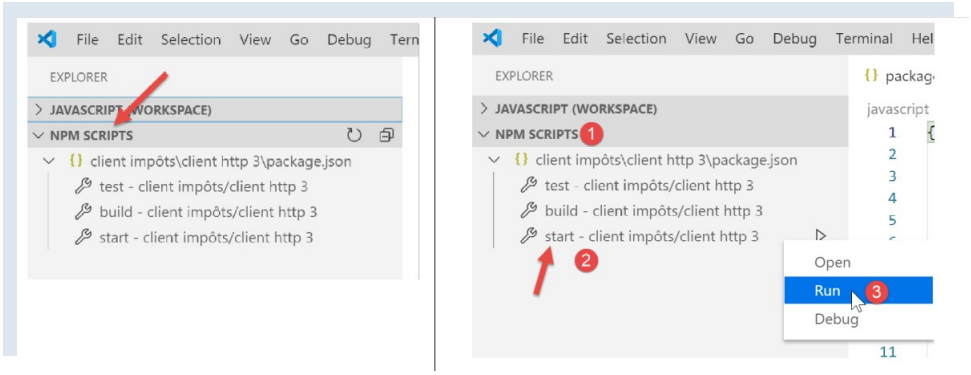
- en **[1-3]**, on compile le projet ;
- en **[4]** : le projet est compilé dans **[dist/main.hash.js]** et une page **[dist/index.html]** est créée ;

La page **[index.html]** générée est la suivante :

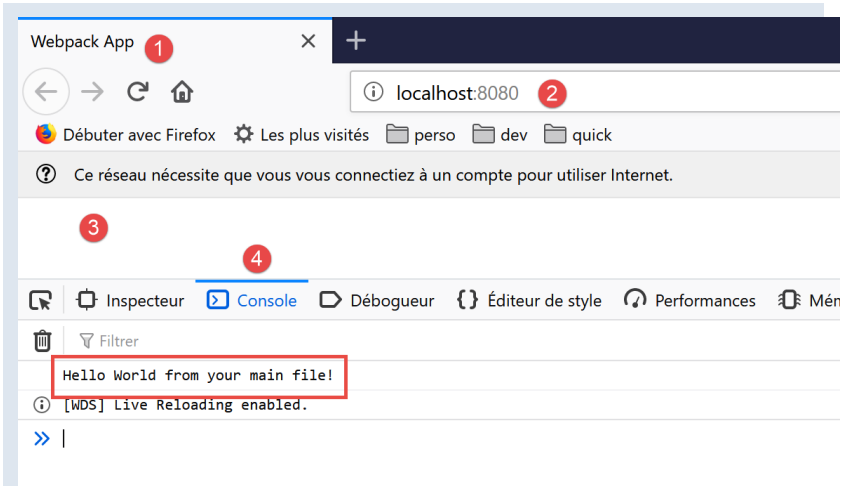
```
1.  <!DOCTYPE html>
2.  <html>
3.    <head>
4.      <meta charset="UTF-8">
5.      <title>Webpack App</title>
6.    </head>
7.    <body>
8.    <script type="text/javascript" src="main.87afc226fd6d648e7dea.js"></script></body>
9.  </html>
```

Cette page se contente donc d'encapsuler le fichier **[main.hash.js]** généré par **[webpack]**.

Le projet est exécuté par la tâche **[start]** :



La page **[dist/index.html]** est alors chargée sur un serveur, appartenant à la suite **[webpack]**, opérant sur le port 8080 de la machine locale et affichée par le navigateur par défaut de la machine :



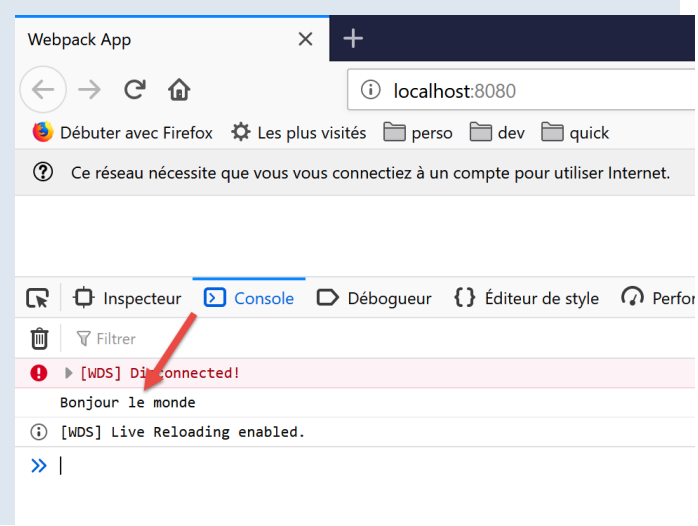
- en **[2]**, le port de service du serveur web de **[webpack]** ;
- en **[3]**, le corps de la page **[dist/index.html]** est vide ;
- en **[4]**, l'onglet **[console]** des outils de développement du navigateur, ici Firefox (F12) ;
- en **[5]**, le résultat de l'exécution du fichier **[src/index.js]**. On rappelle que le contenu de celui-ci était le suivant :

```
1.  console.log("Hello World from your main file!");
```

Maintenant, changeons ce contenu en la ligne suivante :

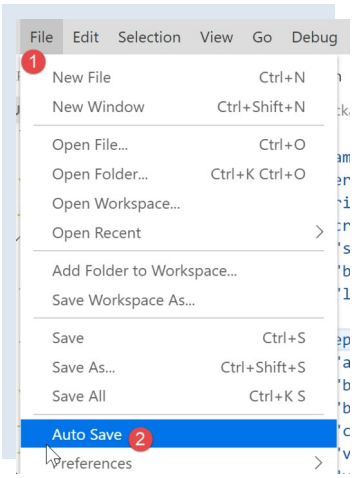
```
1.  console.log("Bonjour le monde");
```

Automatiquement (sans recompiler), de nouveaux fichiers **[main.js, index.html]** sont générés et le nouveau fichier **[index.html]** chargé dans le navigateur :



Il n'est pas nécessaire d'exécuter la tâche **[build]** avant la tâche **[start]** : cette dernière fait d'abord la compilation du projet. Elle ne stocke pas les produits de cette compilation dans le dossier **[dist]**. Pour s'en apercevoir, il suffit de supprimer ce dossier. On verra alors que la tâche **[start]** compile et exécute le projet sans créer le dossier **[dist]**. Elle semble stocker ses produits **[index.html, main.hash.js]** dans un dossier propre à **[webpackdev-server]**. Ce comportement est suffisant pour nos tests.

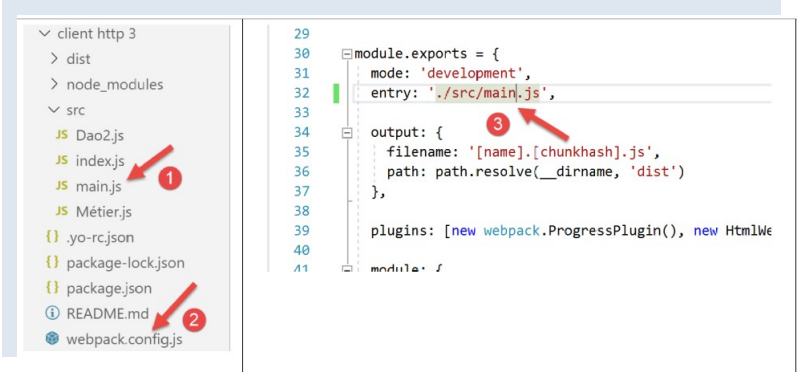
Lorsque le serveur de développement est lancé, toute modification sauvegardée d'un des fichiers du projet provoque une recompilation. Pour cette raison, nous inhibons le mode **[Auto Save]** de **[VSCode]**. En effet, nous ne voulons pas de recompilation dès qu'on tape des caractères dans un des fichiers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



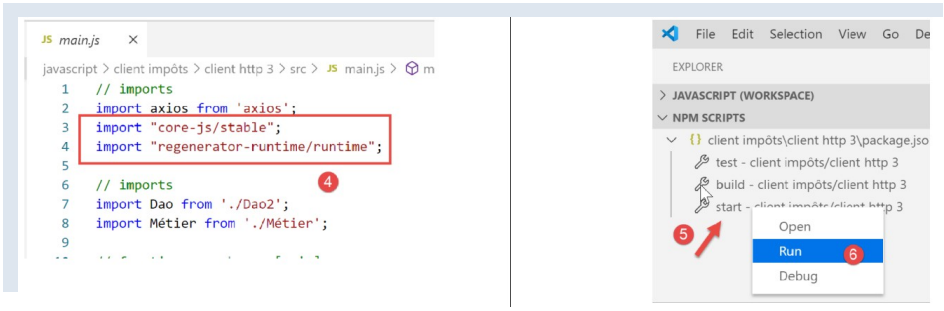
- en **[2]**, l'option **[Auto Save]** ne doit pas être cochée ;

14.4.5 Tests du client Javascript du serveur de calcul de l'impôt

Pour tester le client Javascript du serveur de calcul de l'impôt, il faut désigner **[main.js] [1]** comme le point d'entrée du projet dans le fichier **[webpack.config.js] [2-3]** :



N'oublions pas que le script **[main.js]** doit inclure deux imports supplémentaires par rapport à sa version dans **[Client http 2]** :



Par ailleurs, nous avons légèrement modifié le code pour gérer les erreurs que peut envoyer le serveur :

```
1.  // imports
2.  import axios from 'axios';
3.  import "core-js/stable";
4.  import "regenerator-runtime/runtime";
5.
6.  // imports
7.  import Dao from './Dao2';
8.  import Métier from './Métier';
9.
10. // fonction asynchrone [main]
11. async function main() {
12.   // configuration axios
13.   axios.defaults.timeout = 2000;
14.   axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';
15.   // instanciation couche [dao]
16.   const dao = new Dao(axios);
17.   // requêtes HTTP
18.   let taxAdminData;
19.   try {
20.     // init session
21.     log("-----------init-session");
22.     let response = await dao.initSession();
23.     log(response);
24.     if (response.état != 700) {
25.       throw new Error(JSON.stringify(response.réponse));
26.     }
27.     // authentification
28.     log("-----------authentifier-utilisateur");
29.     response = await dao.authentifierUtilisateur("admin", "admin");
30.     log(response);
31.     if (response.état != 200) {
32.       throw new Error(JSON.stringify(response.réponse));
33.     }
34.     // données fiscales
35.     log("-----------get-admindata");
```

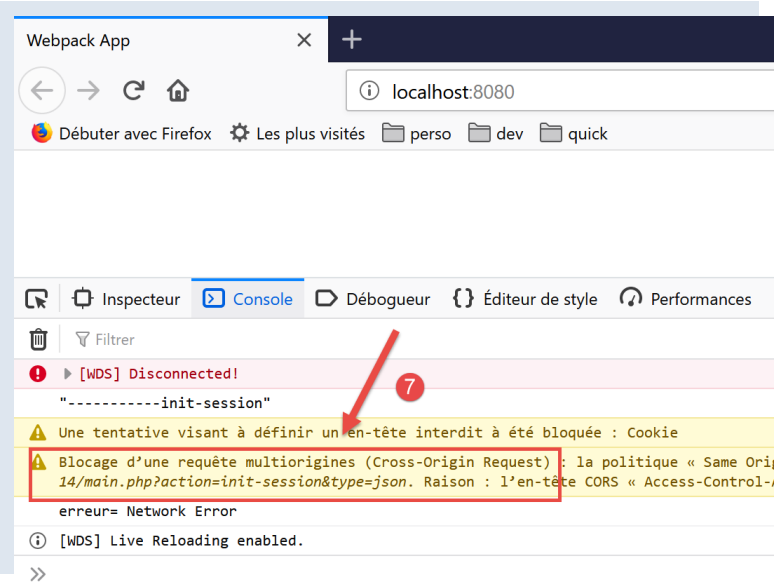
```
36.     response = await dao.getAdminData();
37.     log(response);
38.     if (response.état != 1000) {
39.       throw new Error(JSON.stringify(response.réponse));
40.     }
41.     taxAdminData = response.réponse;
42.   } catch (error) {
43.     // on logue l'erreur
44.     console.log("erreur=", error.message);
45.     // fin
46.     return;
47.   }
48.
49.   // instanciation couche [métier]
50.   const métier = new Métier(taxAdminData);
51.
52.   // calculs d'impôt
53.   log("-----------calculer-impot x 3");
54.   const simulations = [];
55.   simulations.push(métier.calculerImpot("oui", 2, 45000));
56.   simulations.push(métier.calculerImpot("non", 2, 45000));
57.   simulations.push(métier.calculerImpot("non", 1, 30000));
58.   // liste des simulations
59.   log("-----------liste-des-simulations");
60.   log(simulations);
61.   // suppression d'une simulation
62.   log("-----------suppression simulation n° 1");
63.   simulations.splice(1, 1);
64.   log(simulations);
65. }
66.
67. // log jSON
68. function log(object) {
69.   console.log(JSON.stringify(object, null, 2));
70. }
71.
72. // exécution
73. main();
```

Commentaires

- aux lignes **[24-26]**, **[31-33]**, **[38-40]**, on teste le code **[response.état]** envoyé dans la réponse jSON du serveur. Si ce code dénote une erreur, une exception est lancée avec pour message d'erreur la chaîne jSON de la réponse du serveur **[response.réponse]** ;

Ceci fait, nous exécutons le projet **[5-6]**.

La page **[index.html]** est alors générée et chargée dans le navigateur :



- en **[7]**, on voit que l'action **[init-session]** n'a pu aller à son terme à cause d'un problème **[CORS]** (Cross-Origin Resource Sharing) ;

Le problème CORS vient de la relation client / serveur :

- notre client Javascript a été téléchargée sur la machine **[http://localhost:8080]**;
- le serveur de calcul d'impôt s'exécute sur la machine **[http://localhost:80]**;
- le client et le serveur ne sont pas alors dans les mêmes domaines (même machine mais pas même port);
- le navigateur qui exécute le client Javascript chargé à partir de la machine **[http://localhost:8080]** bloque toute requête qui n'a pas pour cible **[http://localhost:80]**. C'est une mesure de sécurité. Aussi bloque-t-il la requête du client vers le serveur qui opère sur la machine **[http://localhost:80]**;

En fait, le navigateur ne bloque pas totalement la requête. Il attend en fait que le serveur lui 'dise' qu'il accepte les requêtes inter-domaines. S'il obtient cette autorisation, le navigateur transmettra alors la requête inter-domaines.

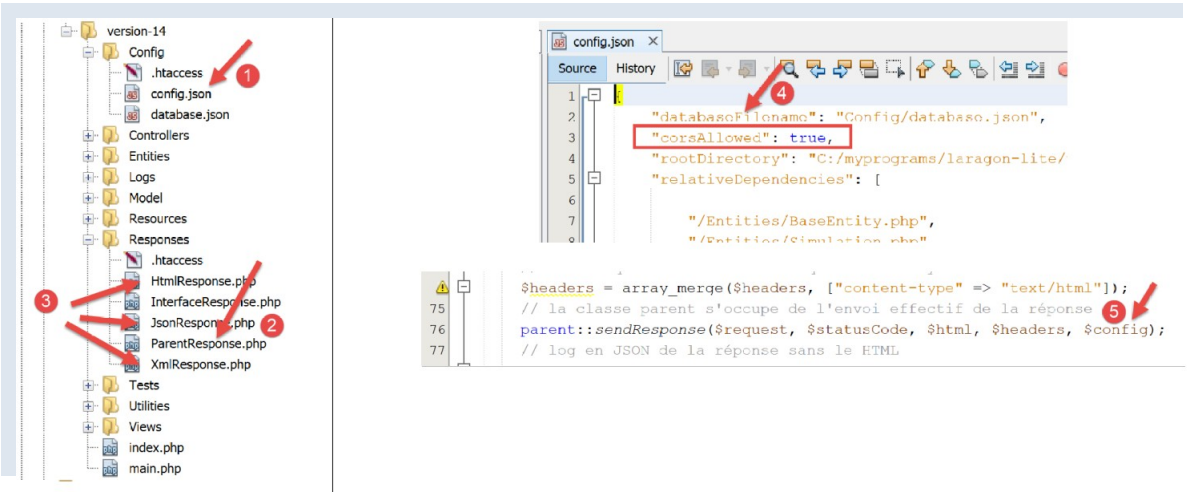
Le serveur donne son autorisation en envoyant des entêtes HTTP particuliers :

```
1.  Access-Control-Allow-Origin: http://localhost:8080
2.  Access-Control-Allow-Headers: Accept, Content-Type
3.  Access-Control-Allow-Methods: GET, POST
4.  Access-Control-Allow-Credentials: true
```

- ligne 1 : le client Javascript opère sur le domaine **[http://localhost:8080]**. Le serveur doit explicitement répondre qu'il accepte ce domaine ;
- ligne 2 : le client Javascript va utiliser dans ses requêtes les entêtes HTTP **[Accept, Content-Type]** :
  - **[Accept]** : cet entête est envoyé dans toute requête ;
  - **[Content-Type]** : cet entête est utilisé dans les opérations POST pour indiquer le type des paramètres du POST ;

  Le serveur doit explicitement accepter ces deux entêtes HTTP ;
- ligne 3 : le client Javascript va utiliser des requêtes GET et POST. Le serveur doit explicitement accepter ces deux types de requêtes ;
- ligne 4 : le client Javascript va envoyer des cookies de session. Le serveur les accepte avec l'entête de la ligne 4 ;

Il nous faut donc modifier le serveur. Nous faisons cela dans **[Netbeans]**. Le problème des CORS est un problème rencontré uniquement en mode développement. En production, le client et le serveur travailleront dans le même domaine **[http://localhost:80]** et il n'y aura pas de problème CORS. Il nous faut donc un moyen d'autoriser ou pas les requêtes CORS par configuration du serveur.



Les modifications du serveur se font à trois endroits :

- **[1, 4]** : dans le fichier de configuration **[config.json]** pour y mettre un booléen qui contrôlera l'acceptation ou non des requêtes inter-domaines ;

- **[2]** : dans la classe **[ParentResponse]** qui envoie la réponse au client Javascript. C'est elle qui enverra les entêtes CORS attendus par le navigateur client ;
- **[3]** : dans les classes **[HtmlResponse, JsonResponse, XmlResponse]** qui génèrent les réponses pour respectivement les sessions **[html, json, xml]**. Ces classes doivent passer à leur classe parente **[2]**, le booléen **[corsAllowed]** trouvé en **[4]**. Cela se fait en **[5]**, en passant le tableau image du fichier jSON **[2]** ;

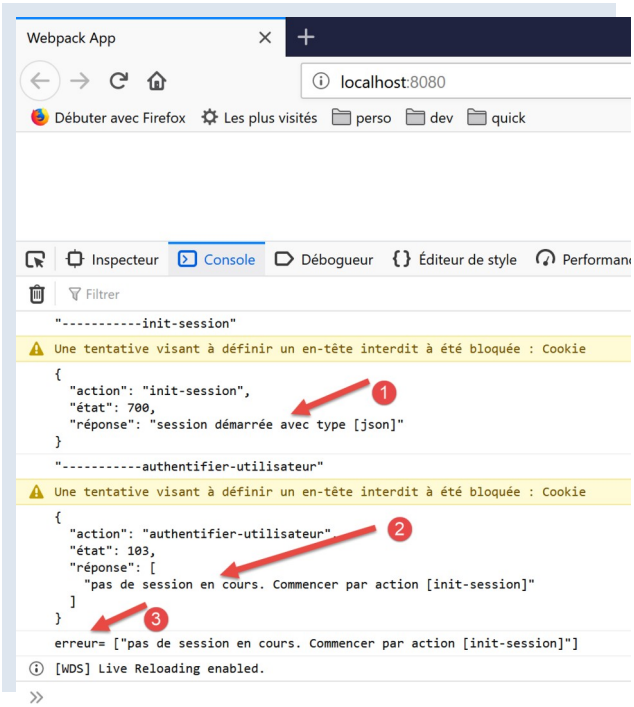
La classe **[ParentResponse] [2]** évolue de la façon suivante :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
6.  use Symfony\Component\HttpFoundation\Response;
7.  use Symfony\Component\HttpFoundation\Request;
8.
9.  class ParentResponse {
10.
11.   // int $statusCode : le code HTTP de statut de la réponse
12.   // string $content : le corps de la réponse à envoyer
13.   // selon les cas, c'est une chaîne JSON, XML, HTML
14.   // array $headers : les entêtes HTTP à ajouter à la réponse
15.
16.   public function sendResponse(
17.     Request $request,
18.     int $statusCode,
19.     string $content,
20.     array $headers,
21.     array $config): void {
22.
23.     // préparation de la réponse texte du serveur
24.     $response = new Response();
25.     $response->setCharset("utf-8");
26.     // code de statut
27.     $response->setStatusCode($statusCode);
28.     // headers pour les requêtes inter-domaines
29.     if ($config['corsAllowed']) {
30.       $origin = $request->headers->get("origin");
31.       if (strpos($origin, "http://localhost") === 0) {
32.         $headers = array_merge($headers,
33.           ["Access-Control-Allow-Origin" => $origin,
34.             "Access-Control-Allow-Headers" => "Accept, Content-Type",
35.             "Access-Control-Allow-Methods" => "GET, POST",
36.             "Access-Control-Allow-Credentials" => "true"
37.         ]);
38.       }
39.     }
40.     foreach ($headers as $text => $value) {
41.       $response->headers->set($text, $value);
42.     }
43.     // cas particulier de la méthode [OPTIONS]
44.     // seuls les entêtes sont importants dans ce cas
45.     $method = strtolower($request->getMethod());
46.     if ($method === "options") {
47.       $content = "";
48.       $response->setStatusCode(Response::HTTP_OK);
49.     }
50.     // on envoie la réponse
51.     $response->setContent($content);
52.     $response->send();
53.   }
54.
55. }
```

- ligne 29 : on regarde si on doit gérer les requêtes inter-domaines. Si oui, on va générer les entêtes HTTP CORS (lignes 33-37) même si la requête courante n'est pas une requête inter-domaines. Dans ce dernier cas, les entêtes CORS seront inutiles et resteront inexploités par le client ;
- ligne 30 : dans une requête inter-domaines, le navigateur client qui interroge le serveur envoie un entête HTTP **[Origin: http://localhost:8080]** (dans le cas précis de notre client Javascript). Ligne 30, on récupère cet entête HTTP dans la requête **[$request]** ;

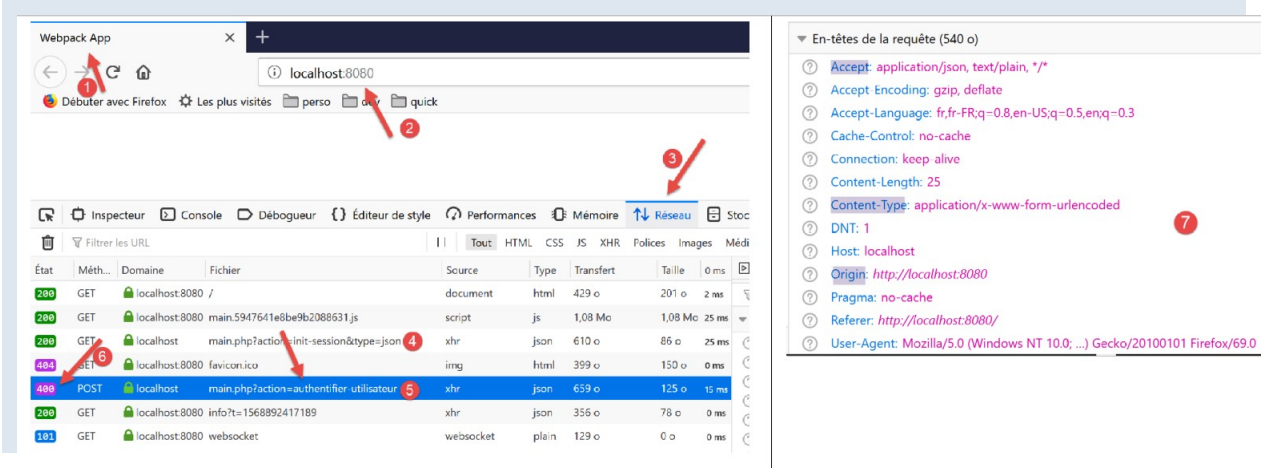
- ligne 31 : on n'acceptera des requêtes inter-domaines provenant uniquement de la machine `[http://localhost]`. On rappelle que ces requêtes n'ont lieu qu'en mode développement du projet ;
- lignes 32-36 : on ajoute les entêtes CORS aux entêtes déjà présents dans le tableau `[$headers]` ;
- lignes 45-49 : la façon dont le navigateur client demande les autorisations CORS peut différer selon le client exécuté. Il arrive parfois que le navigateur client demande ces autorisations avec une commande HTTP `[OPTIONS]`. C'est une nouveauté pour notre serveur qui a été construit pour servir uniquement les commandes `[GET, POST]`. Dans le cas d'une commande `[OPTIONS]`, le serveur génère actuellement une réponse d'erreur. Lignes 46-49, nous corrigeons cela au dernier moment : si ligne 46, nous constatons que la commande courante est une commande `[OPTIONS]`, alors on génère pour le client :
  - lignes 47, 51 : une réponse `[$content]` vide ;
  - ligne 48 : un code de statut de 200 indiquant que la commande est réussie. La seule chose importante pour cette commande est l'envoi des entêtes CORS des lignes 33-36. C'est ce qu'attend le navigateur client ;

Une fois le serveur ainsi corrigé, le client Javascript s'exécute mieux mais fait apparaître une nouvelle erreur :



- en `[1]`, la session jSON est correctement initialisée ;
- en `[2]`, l'action `[authentifier-utilisateur]` échoue : le serveur indique qu'il n'y a pas de session en cours. Cela signifie que le client Javascript ne lui pas renvoyé correctement le cookie de session qu'il a envoyé lors de l'action `[init-session]` ;

Examinons les échanges réseau qui ont eu lieu :



- en **[4]**, la requête **[init-session]**. Elle s'est bien déroulée avec un code 200 pour le statut de la réponse ;
- en **[5]**, la requête **[authentifier-utilisateur]**. Celle-ci échoue avec un code 400 (Bad Request) **[6]** pour le statut de la réponse ;

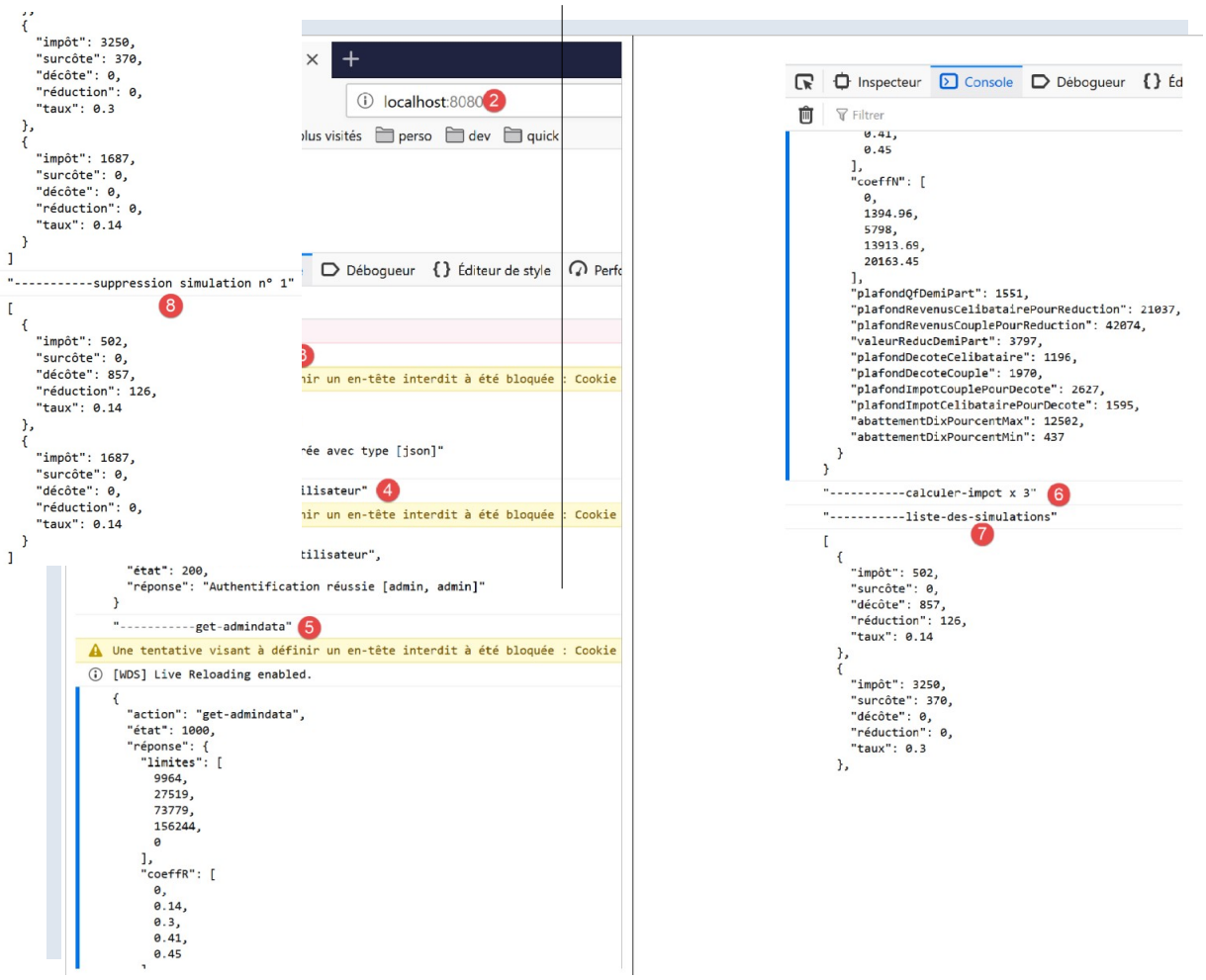
Si on examine les entêtes HTTP **[7]** de la requête **[5]**, on peut voir que le client Javascript n'a pas envoyé l'entête HTTP **[Cookie]** qui lui aurait permis de renvoyer le cookie de session envoyé initialement par le serveur. C'est la raison pour laquelle celui-ci déclare qu'il n'y a pas de session.

Pour que le client envoie le cookie de session, il faut ajouter une configuration à l'objet **[axios]** :

```
1.  // imports
2.  import axios from 'axios';
3.  import "core-js/stable";
4.  import "regenerator-runtime/runtime";
5.
6.  // imports
7.  import Dao from './Dao2';
8.  import Métier from './Métier';
9.
10. // fonction asynchrone [main]
11. async function main() {
12.   // configuration axios
13.   axios.defaults.timeout = 2000;
14.   axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';
15.   axios.defaults.withCredentials = true;
16.   // instanciation couche [dao]
17.   const dao = new Dao(axios);
18.   // requêtes HTTP
19.   let taxAdminData;
20. ...
```

La ligne 15 demande à ce que les cookies soient inclus dans les entêtes HTTP de la requête **[axios]**. Remarquons que cela n'avait pas été nécessaire dans l'environnement **[node.js]**. Il y a donc des différences de code entre les deux environnements.

Une fois cette erreur corrigée, le client Javascript se déroule normalement :



14.5 Amélioration du client HTTP 3

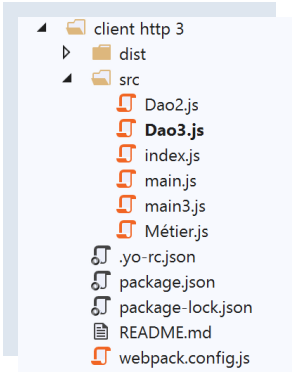
Lorsque la classe [Dao2] précédente s'exécute au sein d'un navigateur, la gestion du cookie de session est inutile. En effet, c'est le navigateur qui héberge la couche [dao] qui gère le cookie de session : il renvoie automatiquement tout cookie que le serveur lui envoie. Du coup la classe [Dao2] peut être réécrite en la classe [Dao3] suivante :

```
1.  "use strict";
2.
3.  // imports
4.  import qs from "qs";
5.
6.  class Dao3 {
7.    // constructeur
8.    constructor(axios) {
9.      this.axios = axios;
10.   }
11.
12.   // init session
13.   async initSession() {
14.     // options de la requête HHTP [get /main.php?action=init-session&type=json]
15.     const options = {
16.       method: "GET",
17.       // paramètres de l'URL
18.       params: {
19.         action: "init-session",
20.         type: "json"
21.       }
22.     };
23.     // exécution de la requête HTTP
24.     return await this.getRemoteData(options);
```

```
25.   }
26.
27.   async authentifierUtilisateur(user, password) {
28.     // options de la requête HHTP [post /main.php?action=authentifier-utilisateur]
29.     const options = {
30.       method: "POST",
31.       headers: {
32.         "Content-type": "application/x-www-form-urlencoded"
33.       },
34.       // corps du POST
35.       data: qs.stringify({
36.         user: user,
37.         password: password
38.       }),
39.       // paramètres de l'URL
40.       params: {
41.         action: "authentifier-utilisateur"
42.       }
43.     };
44.     // exécution de la requête HTTP
45.     return await this.getRemoteData(options);
46.   }
47.
48.   async getAdminData() {
49.     // options de la requête HHTP  [get /main.php?action=get-admindata]
50.     const options = {
51.       method: "GET",
52.       // paramètres de l'URL
53.       params: {
54.         action: "get-admindata"
55.       }
56.     };
57.     // exécution de la requête HTTP
58.     const data = await this.getRemoteData(options);
59.     // résultat
60.     return data;
61.   }
62.
63.   async getRemoteData(options) {
64.     // exécution de la requête HTTP
65.     let response;
66.     try {
67.       // requête asynchrone
68.       response = await this.axios.request("main.php", options);
69.     } catch (error) {
70.       // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
71.       if (error.response) {
72.         // la réponse du serveur est dans [error.response]
73.         response = error.response;
74.       } else {
75.         // on relance l'erreur
76.         throw error;
77.       }
78.     }
79.     // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-
    même)
80.     // la réponse du serveur est dans [response.data]
81.     return response.data;
82.   }
83. }
84.
85. // export de la classe
86. export default Dao3;
```

Tout ce qui avait trait à la gestion du cookie de gestion a disparu.

Nous modifions le projet précédent de la façon suivante :



Dans le dossier [src], nous avons ajouté deux fichiers :

- la classe [Dao3] que nous venons de présenter ;
- le fichier [main3] chargée de lancer la nouvelle version ;

Le fichier [main3] reste identique au fichier [main] de la version précédente mais il utilise désormais la classe [Dao3] :

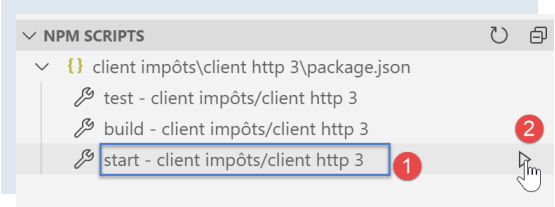
```
1.  // imports
2.  import axios from "axios";
3.  import "core-js/stable";
4.  import "regenerator-runtime/runtime";
5.
6.  // imports
7.  import Dao from "./Dao3";
8.  import Métier from "./Métier";
9.
10. // fonction asynchrone [main]
11. async function main() {
12.   // configuration axios
13.   axios.defaults.timeout = 2000;
14.   axios.defaults.baseURL =
15.     "http://localhost/php7/scripts-web/impots/version-14";
16.   axios.defaults.withCredentials = true;
17.   // instanciation couche [dao]
18.   const dao = new Dao(axios);
19.   // requêtes HTTP
20.   ...
21. }
22.
23. // log jSON
24. function log(object) {
25.   console.log(JSON.stringify(object, null, 2));
26. }
27.
28. // exécution
29. main();
```

Le fichier [webpack.config] est modifié pour exécuter maintenant, le script [main3] :

```
1.  /* eslint-disable */
2.
3.  const path = require("path");
4.  const webpack = require("webpack");
5.
6.  /*
7.   * SplitChunksPlugin is enabled by default and replaced
8.   * deprecated CommonsChunkPlugin. It automatically identifies modules which
9.   * should be splitted of chunk by heuristics using module duplication count and
10.  * module category (i. e. node_modules). And splits the chunks…
11.  *
12.  * It is safe to remove "splitChunks" from the generated configuration
13.  * and was added as an educational example.
14.  *
15.  * https://webpack.js.org/plugins/split-chunks-plugin/
16.  *
```

```
17.  */
18.
19. const HtmlWebpackPlugin = require("html-webpack-plugin");
20.
21. /*
22.  * We've enabled HtmlWebpackPlugin for you! This generates a html
23.  * page for you when you compile webpack, which will make you start
24.  * developing and prototyping faster.
25.  *
26.  * https://github.com/jantimon/html-webpack-plugin
27.  *
28.  */
29.
30. module.exports = {
31.   mode: "development",
32.   //entry: "./src/mainjs",
33.   entry: "./src/main3.js",
34.   output: {
35.     filename: "[name].[chunkhash].js",
36.     path: path.resolve(__dirname, "dist")
37.   },
38.
39.   plugins: [new webpack.ProgressPlugin(), new HtmlWebpackPlugin()],
40. ...
41. };
```

Ceci fait, on exécute le projet après avoir lancé le serveur de calcul de l'impôt :



Les résultats obtenus dans la console du navigateur sont identiques à ceux de la version précédente.

14.6 Conclusion

Nous avons désormais désormais tous les outils pour développer le code Javascript d'une application web. Nous pouvons :

- utiliser le code ECMAScript le plus récent ;
- tester des éléments isolés de ce code dans un environnement **[node.js]** plus simple pour le débogage et les tests ;
- porter ensuite ce code dans un navigateur grâce aux outils **[babel]** et **[webpack]** ;