Découverte

Le motif MVC et les modules ES6 (JavaScript)

Rédigé par

David ROUMANET
Professeur BTS SIO

Changement

Date	Révision
2024-03-04	Correction sur export (inversion des classes) et App.js (majuscule au début) partout + index.html module

Sommaire

A Introduction..... .1
A.1 Présentation MVC..... .1
A.2 Présentation module ES6..... .2
A.3 Prérequis..... .2
B Programmer un compteur..... .3
B.1 Structure physique..... .3
B.2 Les fichiers..... .4
C Compatibilité modules ES6..... .6
C.1 Création du nouveau projet..... .6
C.2 Modification des fichiers..... .6
D Évolution potentielle..... .8
E Projet multi compteur en Vue.js..... .9
E.1 Création du projet..... .9
E.2 Création des fichiers..... .9
E.3 Résultat..... .10
F Annexes..... .11
F.1 Fonction bind()..... .11
F.2 Framework Vue.js..... .11
F.3 Motif MVC..... .11
G Ce qu'il faut retenir…..... .12
G.1 MVC..... .12
G.2 Module ES6..... .12

Nomenclature :

- **Assimiler** : cours pur. Explication théorique et détaillée (globalement supérieur à 4 pages).
- **Décoder** : fiche de cours, généralement inférieure à 5 pages.
- **Découvrir** : Travaux dirigés. Faisable sans matériel.
- **Explorer** : Travaux pratiques. Nécessite du matériel ou des logiciels.
- **Mission** : Projet encadré ou partie d'un projet.
- **Voyager** : Projet en autonomie totale. Environnement ouvert : Vous êtes le capitaine !

A Introduction

Pour coder de manière plus efficace, il est nécessaire de structurer son code.

La norme dans ce domaine est une méthode pratiquée par de nombreux codeurs, projets après projets. Il s'agit du design pattern (motif) MVC, pour Modèle – Vue – Contrôleur.

L'objectif de cette activité de découverte est donc de générer une application Java capable de gérer des étudiants dans une classe et ceci, en utilisant le motif MVC.

A.1 Présentation MVC

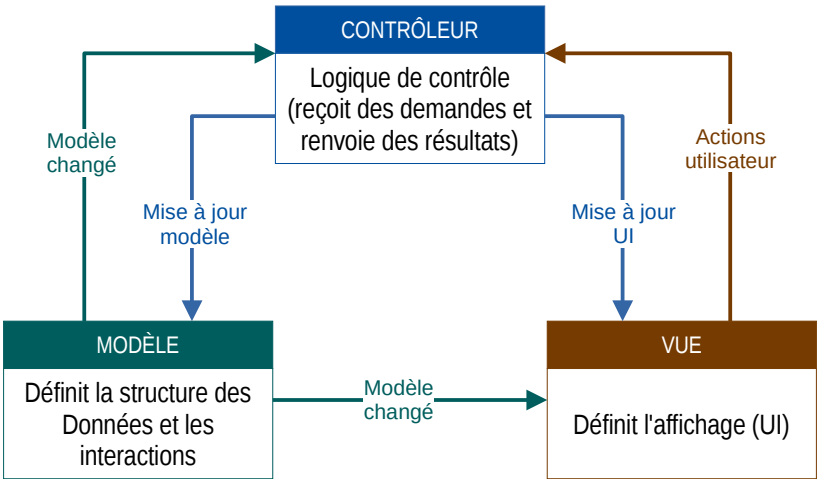
Dans une application, on constate souvent les mêmes interactions :

- Les actions sur les éléments présents dans une base de données
- L'affichage et le traitement des données pour la mise en forme de celles-ci
- Les interactions avec l'utilisateur

C'est devenu très visible dans les applications web, car le traitement est fait côté serveur (en PHP, ou par Node.JS) et l'affichage est réalisé sur le terminal de l'utilisateur (page HTML/JS dans le navigateur).

En séparant ces trois domaines, on obtient la séparation des rôles et ainsi, un code comportant plusieurs fichiers bien distincts mais, reliés logiquement ensemble.

- Modèle : ce sont les fichiers qui décrivent le format des données et permettent de les modifier.
- Vue : ce sont les fichiers qui reçoivent les données et exécutent un traitement de mise en forme.
- Contrôleur : ce sont les fichiers qui font le lien entre le·s modèle·s et la ou les vues



A.2 Présentation module ES6

JavaScript évolue régulièrement et permet désormais l'importation et l'exportation de modules.

Un module est un simple fichier Javascript, qui exporte ses méthodes et ses attributs. L'intérêt et de permettre une programmation modulaire – comprendre par blocs de modules – qui facilite le travail du développeur.

La norme ES6 permet de déclarer des modules, mais pour des raisons de sécurité, les navigateurs ne les acceptent pas toujours en mode local : cela implique donc l'usage constant d'un serveur pour ne pas rencontrer d'erreur CORS (Cross-origin resource sharing).

Une erreur CORS survient lorsqu'un fichier local appelle une ressource externe au site, ce qui ressemble à une attaque de type Cross Scripting. Exécuté sans serveur, une page web est considérée comme dangereuse et ses possibilités sont restreintes par le navigateur.

L'intérêt des modules à la norme ES6, est que l'ordre d'importation ne dépend plus du fichier HTML, mais du fichier JavaScript qui veut utiliser les modules.

Exemple de code :

```
import { MaClasse } from "./maClasse.js";

export class ClasseUtilisatrice {
  // Code pour la classe ClasseUtilisatrice
}
```

Il est donc utile d'installer initialement un outil LiveServer qui permet également le débogage dans Visual Studio Code. L'outil utilisé dans cet activité est celui de Microsoft : https://open-vsx.org/extension/ms-vscode/live-server

A.3 Prérequis

La bonne compréhension du MVC implique d'avoir déjà codé de manière basique, des fonctions (méthodes) nombreuses.

L'usage correct des modules ES6 nécessite de comprendre l'imbrication des codes et le fonctionnement des objets.

Si vous avez déjà connu le sentiment de ne pas pouvoir retravailler un code, parce qu'il est devenu trop complexe, le motif MVC est le moyen de réduire ce problème dans vos projets.

B Programmer un compteur

Nous allons créer un simple compteur en JavaScript et trois boutons pour interagir avec lui :

- Incrémenter
- Décrémenter
- Remise à zéro

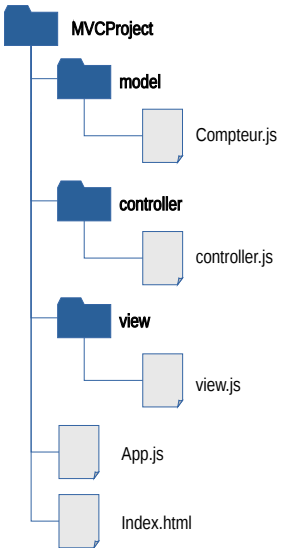
Pour le moment, il n'y aura pas de limitations (bornes de maximum et minimum).

B.1 Structure physique

La structure est la suivante :

Il faut créer un répertoire MVCProject qui contiendra

- un répertoire 'model'
- un répertoire 'controller'
- un répertoire 'view'
- un fichier 'index.html'
- un fichier 'App.js'



Si vous souhaitez ajouter de la mise en forme, il suffit de créer les répertoires 'css' et 'public' à la racine de MVCProject.

L'important est de comprendre que c'est le fichier App.js qui devient le coeur de l'application.

Dans le cycle NodeJS, nous verrons qu'il est possible de réaliser des importations de modules directement dans JavaScript, sans utiliser le fichier HTML pour les importer.

B.2 Les fichiers

Le fichier index.html définit ce que contiendra la vue.

index.html

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple MVC</title>
</head>
<body>
    <div id="app">
        <h1>Compteur : <span id="counter"></span></h1>
        <button id="increment">Incrémenter</button>
        <button id="decrement">Décrémenter</button>
        <button id="raz">Remise à zéro</button>
    </div>

    <script src="model/Compteur.js"></script>
    <script src="view/view.js"></script>
    <script src="controller/controller.js"></script>
    <script src="App.js"></script>
</body>
</html>
```

Le fichier App.js ne contient qu'une ligne : il relie le model et le contrôleur ensemble.

App.js

```
const controller = new Controller(model, view);
```

Le fichier Compteur.js contient le modèle de données et ses accesseurs/mutateurs.

Compteur.js

```
class Model {
    #counter;
    #maximum;
    #minimum;

    constructor() {
        this.#counter = 0;
        this.#maximum = 20;
        this.#minimum = -20
    }
    incrementCounter() {
        this.#counter++;
    }
    decrementCounter() {
        this.#counter--;
    }
    RAZCounter() {
        this.#counter = 0;
    }
    getCounter() {
        return this.#counter;
    }
}

const model = new Model();
```

Il est important de noter qu'il n'y a aucun affichage dans le modèle. Nous utilisons des attributs privés (variables précédées du signe `#`).

Le fichier view.js contient les interactions avec la vue : Chaque élément de la vue est associé à un événement (les bind… sont des liens avec une interaction) tandis qu'une seule méthode affiche le compteur (displayCounter(counter)).

View.js

```
class View {
    constructor() {
        this.counterElem = document.getElementById('counter');
        this.incrementBtn = document.getElementById('increment');
        this.decrementBtn = document.getElementById('decrement');
        this.razBtn = document.getElementById('raz');
    }
    displayCounter(counter) {
        this.counterElem.textContent = counter;
    }
    bindIncrementEvent(handler) {
        this.incrementBtn.addEventListener('click', handler);
    }
    bindDecrementEvent(handler) {
        this.decrementBtn.addEventListener('click', handler);
    }
    bindRAZEvent(handler) {
        this.razBtn.addEventListener('click', handler);
    }
}

const view = new View();
```

Enfin, le contrôleur gère le traitement des actions de l'utilisateur. Il utilise les événements de la vue pour recevoir des informations.

controller.js

```
class Controller {
    constructor(model, view) {
        this.model = model;
        this.view = view;

        this.view.bindIncrementEvent(this.handleIncrement.bind(this));
        this.view.bindDecrementEvent(this.handleDecrement.bind(this));
        this.view.bindRAZEvent(this.handleRAZ.bind(this));

        this.updateView();
    }
    handleIncrement() {
        this.model.incrementCounter();
        this.updateView();
    }
    handleDecrement() {
        this.model.decrementCounter();
        this.updateView();
    }
    handleRAZ() {
        this.model.RAZCounter();
        this.updateView();
    }

    updateView() {
        this.view.displayCounter(this.model.getCounter());
    }
}
```

C Compatibilité modules ES6

Le code précédent s'appuie énormément sur le fichier HTML pour utiliser les différentes fonctions présentes dans les différents fichiers.

Pour utiliser plus facilement le code et le rendre modulaire, il est utile d'utiliser les modules. Dans cette activité, nous ne parlerons pas des modules dans Node.js (il existait une norme appelée CommonJS pour les déclarer), mais uniquement des modules en version ES6 (Node.js est également capable de les utiliser).

C.1 Création du nouveau projet

Copiez le contenu du répertoire MVCProject dans un nouveau répertoire MVC2Project, **à l'intérieur d'un serveur web (par exemple www de votre WAMP)**.

C.2 Modification des fichiers

Nous allons dans un premier temps supprimer les scripts inutiles du fichier index.html

index.html

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple MVC</title>
</head>
<body>
    <div id="app">
        <h1>Compteur : <span id="counter"></span></h1>
        <button id="increment">Incrémenter</button>
        <button id="decrement">Décrémenter</button>
        <button id="raz">Remise à zéro</button>
    </div>

    <script src="model/compteur.js"></script>
    <script src="view/view.js"></script>
    <script src="controller/controller.js"></script>
    <script type="module" src="App.js"></script>
</body>
</html>
```

En effet, le point d'entrée de notre module sera App.js, qui va avoir beaucoup plus de lignes...

App.js

```
// Importation des classes Compteur, View et Controller
import { Compteur } from './model/Compteur.js';
import { View } from './view/view.js';
import { Controller } from './controller/controller.js';

// Création des instances de Compteur et View
const compteur = new Compteur();
const view = new View();

// Création de l'instance de Controller en lui passant les instances de Compteur et View
const controller = new Controller(compteur, view);
```

Les autres fichiers vont subir des modifications mineures, notamment par l'ajout du mot-clé `export` sur les classes déclarées :

view.js

```
export class View {
    // code déjà existant
}
```

Compteur.js

```
export class Compteur {
    // code déjà existant
}
```

controller.js

```
import { Compteur } from "../model/Compteur.js";

export class Controller {
    constructor(compteur, view) {
        this.compteur = compteur;
        this.view = view;

        this.view.bindIncrementEvent(this.handleIncrement.bind(this));
        this.view.bindDecrementEvent(this.handleDecrement.bind(this));
        this.view.bindRAZEvent(this.handleRAZ.bind(this));

        this.updateView();
    }

    handleIncrement() {
        this.compteur.incrementCounter();
        console.log("increment");
        this.updateView();
    }
    handleDecrement() {
        this.compteur.decrementCounter();
        console.log("decrement");
        this.updateView();
    }
    handleRAZ() {
        this.compteur.RAZCounter();
        this.updateView();
    }

    updateView() {
        this.view.displayCounter(this.compteur.getCounter());
    }
}
```

L'importation du module Compteur dans le contrôleur, ne sert que pour l'usage des méthodes incrementCounter(), decrementCounter() et RAZCounteur().

Note : le formalisme des chemins avec `./` ou `../` est obligatoire en JavaScript et il n'est pas autorisé de commencer un chemin avec `/`.

L'application devrait fonctionner comme ceci (pensez à enregistrer vos fichiers avant de la tester).

D Évolution potentielle...

Nous pourrions avoir une application qui aurait plusieurs compteurs affichés.

Un modèle supplémentaire serait ListeCompteur (une classe gérant une liste de Compteur), avec son contrôleur ListeController (qui lirait les boutons associés aux bons compteurs) et une vue ListeView qui permettrait d'afficher les compteurs dans la liste.

Cela signifie qu'il faut faire évoluer la vue View, en modifiant son constructeur : actuellement, il n'a aucun paramètre et retrouve les boutons codés "en dur" dans les premières lignes :

```
    constructor() {
        this.counterElem = document.getElementById('counter');
        this.incrementBtn = document.getElementById('increment');
        this.decrementBtn = document.getElementById('decrement');
        this.razBtn = document.getElementById('raz');
    }
```

par

```
    constructor(counter, btInc, btDec, btRAZ) {
        this.counterElem = document.getElementById(counter);
        this.incrementBtn = document.getElementById(btInc);
        this.decrementBtn = document.getElementById(btDec);
        this.razBtn = document.getElementById(btRAZ);
    }
```

Et il faudrait probablement que le contrôleur de liste puisse générer les emplacements d'affichage d'un compteur, avec des attributs différents.

Cependant, la complexité de gestion deviendrait vite importante... c'est pourquoi des frameworks ont été développés pour résoudre ceci : reactJS, vueJS, AngularJS, etc.

Dans l'exemple suivant, nous allons tester un code en Vue.JS pour mieux comprendre les modules et valider que le code est plus simple.

E Projet multi compteur en Vue.js

E.1 Création du projet

Dans un nouveau répertoire appelé "MultiCountProject", créer 3 fichiers :

- Un fichier index.html
- Un fichier App.js
- Un fichier Compteur.js (dans un répertoire 'model')

E.2 Création des fichiers

Voici les fichiers.

index.html

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Multi compteur en Vue.js</title>
</head>
<body>
  <div id="app">
    <div v-for="(compteur, index) in compteurs" :key="index">
      <h1>Compteur {{ index + 1 }} : {{ compteur.valeur }}</h1>
      <button @click="incrementer(index)">Incrémenter</button>
      <button @click="decrementer(index)">Décrémenter</button>
      <button @click="RAZ(index)">RAZ</button>
    </div>
  </div>
  <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
  <script type="module" src="App.js"></script>
</body>
</html>
```

Compteur.js

```
class Compteur {
    constructor() {
      this.valeur = 0;
    }

    incrementerCompteur() {
      this.valeur++;
    }
    decrementerCompteur() {
      this.valeur--;
    }
    RAZCompteur() {
      this.valeur = 0;
    }
  }

  export default Compteur;
```

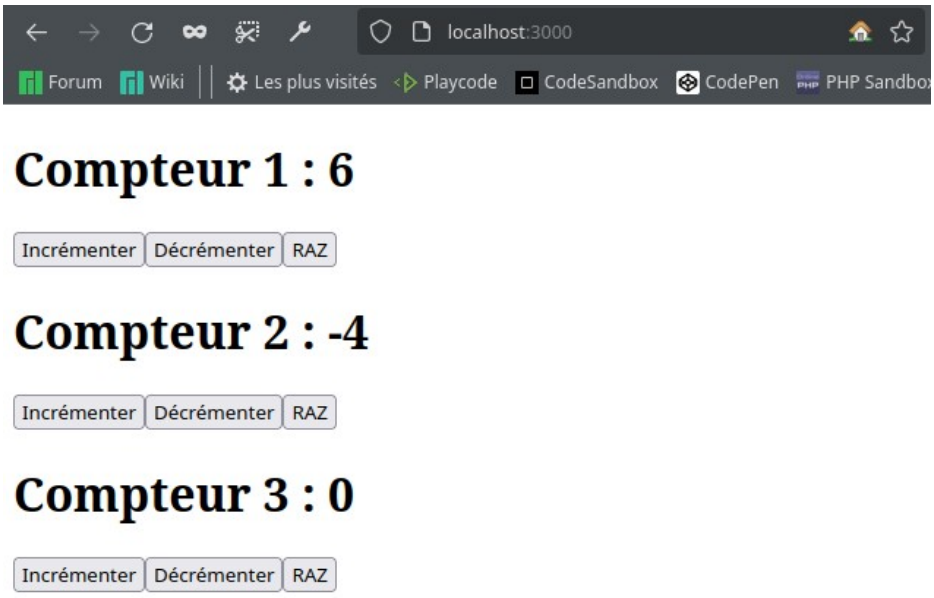
App.js

```
import Compteur from './model/Compteur.js';

const app = Vue.createApp({
  data() {
    return {
      compteurs: [new Compteur(), new Compteur(), new Compteur()],
    };
  },
  methods: {
    incrementer(index) {
      this.compteurs[index].incrementerCompteur();
    },
    decrementer(index) {
      this.compteurs[index].decrementerCompteur();
    },
    RAZ(index) {
      this.compteurs[index].RAZCompteur();
    },
  },
});

app.mount('#app');
```

E.3 Résultat

Comme vous pouvez le voir, Vue.js n'impose pas un MVC pour fonctionner (c'est donc à vous d'être structuré), mais sa syntaxe particulière dans le fichier index.html permet de transformer la vue pour utiliser des contrôleurs simples.

Ici, nous avons maintenant trois compteurs indépendants, qui sont gérés comme des objets.



F Annexes

F.1 Fonction bind()

En JavaScript, la fonction bind() permet d'associer une action avec son propre objet (plutôt qu'un objet général, comme `Windows` ou `Document`).

Voir https://www.delftstack.com/fr/howto/javascript/bind-method-javascript/ pour plus de détails ;

Ainsi, plutôt que de définir pour chaque objet, une méthode identique dans chacun d'un, on définit une seule fonction qui prendra le contexte de l'objet.

F.2 Framework Vue.js

Il s'agit d'un framework simplifiant les interactions entre l'utilisateur et l'application locale. Rien n'est transmis vers un éventuel serveur.

https://vuejs.org/

F.3 Motif MVC

Il n'existe pas que le motif MVC, mais également MVVC (Model-View View-Model) et MPV (Model – Presenter – View), chacun apportant des variations dans les interactions (les flux).

https://welovedevs.com/fr/articles/mvc/

https://www.baeldung.com/mvc-vs-mvp-pattern

G Ce qu’il faut retenir…

G.1 MVC

Le motif MVC (Modèle – Vue – Contrôleur) est un standard très utilisé pour rédiger du code. Son intérêt est de créer trois fichiers par objet à traiter.

Dans une application contenant des médecins, des patients, des médicaments, on sépare ainsi, de manière logique, les trois flux.

Il est ainsi plus facile de travailler en mode projet, avec plusieurs intervenants sur la même application : on distinguera aussi plus facilement les développeurs back-end (qui travailleront sur les modèles) et les développeurs front-end (qui travailleront plutôt sur les vues).

G.2 Module ES6

L'utilisation des modules ES6 nécessitent un serveur capable de gérer les types "modules", sinon il peut y avoir un message d'erreur :

```
Le chargement du module à l’adresse « http://127.0.0.1:5500/app.js » a été bloqué en raison d’un type MIME interdit (« text/html »)
```

L'intérêt des modules est de permettre une meilleure indépendance des fichiers utilisés dans une application. L'objectif étant que chaque bibliothèque/librairie utilise ses propres modules, sans que les développeurs n'aient à gérer les collisions de noms, de fichiers, de variables et fonctions.