Documentation Technique

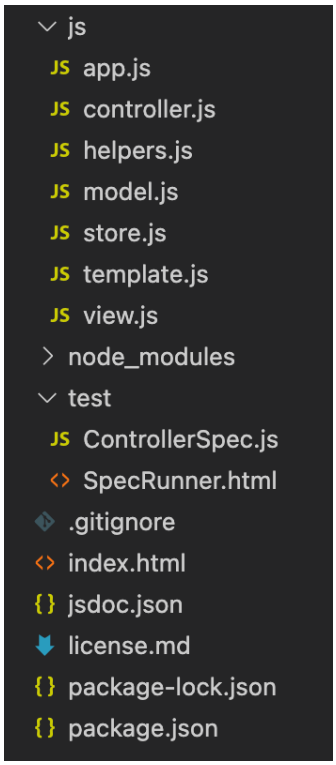
TodoMVC

1. Introduction

Cette application de ToDo List permet de créer une liste de tâches, de les modifier, de les filtrer et de les supprimer.
Ces tâches sont sauvegardées en local chez l'utilisateur, dans son navigateur.

L'application est développée en Vanilla Javascript et est organisée selon l'architecture MVC (Model – View – Controller).
Aucun framework, bibliothèque ou service tiers n'est utilisé.

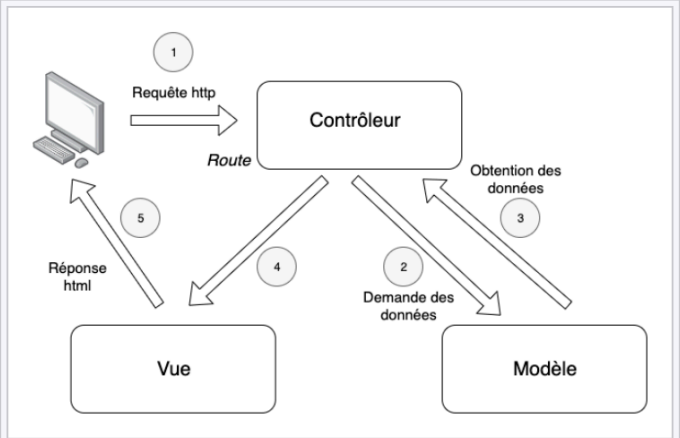
2. Architecture



App.js	Initialise l'application, instancie les classes
Controller.js	Regroupe les actions possibles de l'utilisateur pour les transmettre au modèle et à la vue.
Helpers.js	Regroupe les fonctions utilitaires (permet de se passer d'une librairie comme jQuery par exemple
Model.js	Crée une instance d'une tache et la lie au localStorage
Store.js	Crée et gère la base de données dans le localStorage
Template.js	Génère un template par défaut
View.js	Gère l'affichage du DOM
Node_module	Regroupe notamment le style css et Jasmine pour les tests
Test	Définit les tests qui seront effectués

Une documentation détaillée est présente sous format JSDoc, dans le dossier docs.

Le pattern MVC



Représentation des interactions entre le modèle, la vue et le contrôleur dans le cas d'une application web.

L'objectif de cette architecture est de séparer le code en 3 parties :

- Model : Il gère les données : création, mise à jour, suppression. Dans notre application, le model communique avec le store afin d'enregistrer les données dans le localStorage.

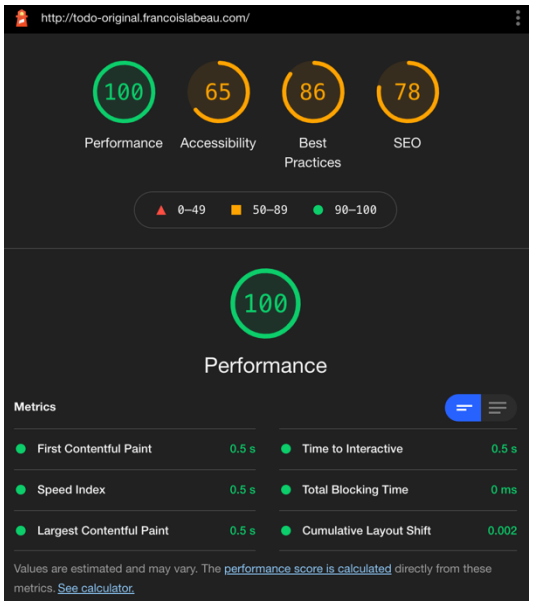
- View : Elle gère l'affichage du DOM et écoute les évènements (les actions de l'utilisateurs).

- Controller : il traite les actions de l'utilisateurs et envoie ces informations au modèle et la vue.

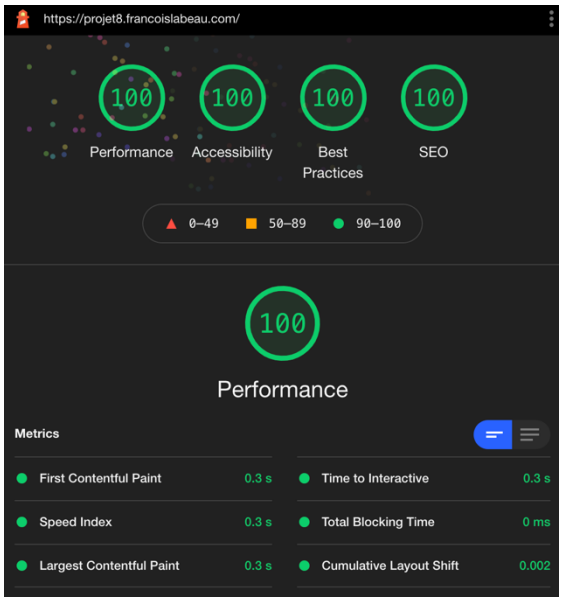
3. Audit

Afin de réaliser l'audit de notre application, l'application originale et la version optimisée ont été hébergées sur internet.

Avant



Après



Même avec la version d'origine, on peut voir que la performance est déjà très bonne car l'application est très légère. Il n'y a aucun service tiers, pas d'images et peu de requêtes.

Néanmoins, certaines améliorations ont été apportées.

3.1 Correction des erreurs

- Faute de frappe dans le nommage de la fonction addItem dans controller.js ligne 100. (Il y avait 3 d : adddItem)
- Problème d'Id dans store.js qui pouvait engendrer la création 2 Id identiques.
- La flèche « toggle.all » ne fonctionnait pas. Il n'y avait pas d'Id sur l'input mais une classe. Pour associer un élément <label> avec un élément <input>, il faut fournir un identifiant à l'élément <input> sous la forme d'un attribut id. Ensuite on peut renseigner l'attribut for de l'élément <label> avec la valeur de cet identifiant.
- Dans la console, nous avions 2 erreurs :
    - Un appel à learn.json dans les scripts d'index.html. Ce fichier n'étant pas présent et ne perturbant pas le bon fonctionnement de l'application, j'ai supprimé ce script.
    - Un défaut de favicon : j'ai rajouté un <link rel='icon' href=``/> dans le head du index.html.

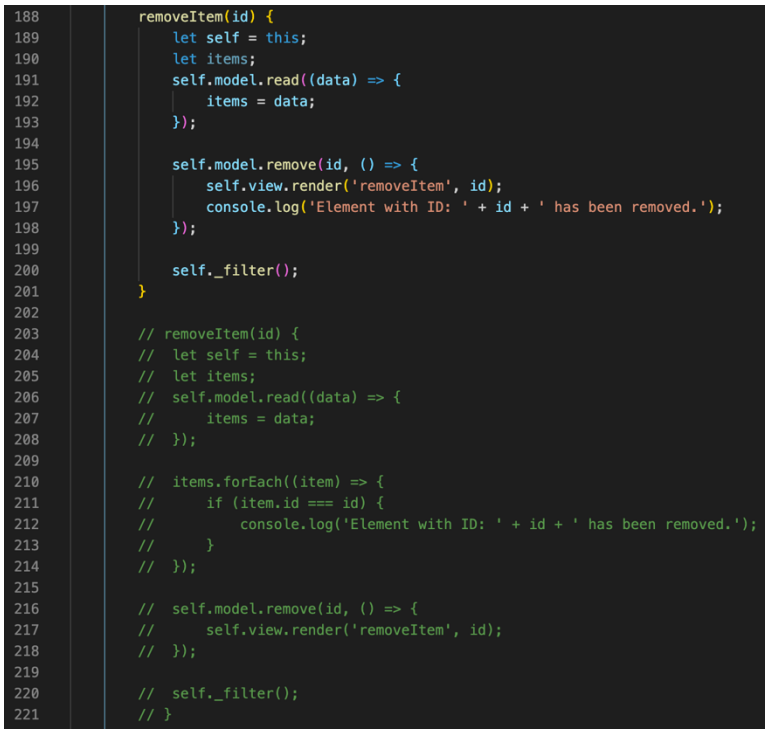
3.2 Améliorations

Utilisation du langage ES6 pour passer des « fonctions constructrices » aux « classes », des fonctions fléchées. Ceci nous donne une meilleure visibilité dans le code.

- Controller.js :
  - Ligne 134 : utilisation de trim() pour supprimer les blancs ou espaces à la place de while() et slice()

```
134     editItemSave(id, title) {
135         let self = this;
136 
137         if (title.length === 0) {
138             self.removeItem(id);
139             return;
140         }
141         title = title.trim();
142         self.model.update(id, { title: title }, () => {
143             self.view.render('editItemDone', { id: id, title: title });
144         });
145     }
146 
147     // editItemSave(id, title) {
148     //  let self = this;
149 
150     //  while (title[0] === ' ') {
151     //      title = title.slice(1);
152     //  }
153 
154     //  while (title[title.length - 1] === ' ') {
155     //      title = title.slice(0, -1);
156     //  }
157 
158     //  if (title.length !== 0) {
159     //      self.model.update(id, { title: title }, () => {
160     //          self.view.render('editItemDone', { id: id, title: title });
161     //      });
162     //  } else {
163     //      self.removeItem(id);
164     //  }
165     // }
```

  - Ligne 188 : suppression de la boucle forEach car on a déjà l'id en paramètre de remove(). On affiche simplement le console.log à ce moment-là.



```
188     removeItem(id) {
189         let self = this;
190         let items;
191         self.model.read((data) => {
192             items = data;
193         });
194 
195         self.model.remove(id, () => {
196             self.view.render('removeItem', id);
197             console.log('Element with ID: ' + id + ' has been removed.');
198         });
199 
200         self._filter();
201     }
202 
203     // removeItem(id) {
204     //  let self = this;
205     //  let items;
206     //  self.model.read((data) => {
207     //      items = data;
208     //  });
209 
210     //  items.forEach((item) => {
211     //      if (item.id === id) {
212     //          console.log('Element with ID: ' + id + ' has been removed.');
213     //      }
214     //  });
215 
216     //  self.model.remove(id, () => {
217     //      self.view.render('removeItem', id);
218     //  });
219 
220     //  self._filter();
221     // }
```

- Store.js :
    - Ligne 84 : Au lieu de générer un ID qui a 6 caractères entre les nombres 0 à 9, on assigne un ID avec Date.now(), qui sera forcément unique (Nbre de millisecondes écoulées depuis le 1er janvier 1970)

```
84      save(updateData, callback, id) {
85          let data = JSON.parse(localStorage[this._dbName]);
86          let todos = data.todos;
87
88          callback = callback || function () {};
89
90          // Generate an ID
91          // let newId = '';
92          // let charset = '0123456789';
93
94          // for (let i = 0; i < 6; i++) {
95          //  newId += charset.charAt(Math.floor(Math.random() * charset.length));
96          // }
97
98          // If an ID was actually given, find the item and update each property
99          if (id) {
100             for (let i = 0; i < todos.length; i++) {
101                 if (todos[i].id === id) {
102                     for (let key in updateData) {
103                         todos[i][key] = updateData[key];
104                     }
105                     break;
106                 }
107             }
108
109             localStorage[this._d let todos: any ringify(data);
110             callback.call(this, todos);
111         } else {
112             // Assign an ID
113             updateData.id = Date.now();
114
115             todos.push(updateData);
116             localStorage[this._dbName] = JSON.stringify(data);
117             callback.call(this, [updateData]);
118         }
119     }
120
```

    - Ligne 128 : On supprime la variable todoId pour regrouper les 2 boucles en une seule

```
128     remove(id, callback) {
129         let data = JSON.parse(localStorage[this._dbName]);
130         let todos = data.todos;
131         // let todoId;
132
133         for (let i = 0; i < todos.length; i++) {
134             if (todos[i].id === id) {
135                 todos.splice(i, 1);
136             }
137         }
138
139         // for (let i = 0; i < todos.length; i++) {
140         //  if (todos[i].id == id) {
141         //      todoId = todos[i].id;
142         //  }
143         // }
144
145         // for (let i = 0; i < todos.length; i++) {
146         //  if (todos[i].id == todoId) {
147         //      todos.splice(i, 1);
148         //  }
149         // }
150
151         localStorage[this._dbName] = JSON.stringify(data);
152         callback.call(this, todos);
153     }
```

- View.js :
    - Ligne 222 : Utilisation de l'instruction switch à la place des conditions if…else. Cela rend le code plus clair et plus rapide à exécuter. Si la valeur de notre variable est égale à celle du « case », alors on exécute le code qui est à l'intérieur.

```
222     bind(event, handler) {
223         let self = this;
224
225         switch (event) {
226             case 'newTodo':
227                 $on(self.$newTodo, 'change', function () {
228                     handler(self.$newTodo.value);
229                 });
230                 break;
231             case 'removeCompleted':
232                 $on(self.$clearCompleted, 'click', function () {
233                     handler();
234                 });
235                 break;
236             case 'toggleAll':
237                 $on(self.$toggleAll, 'click', function () {
238                     handler({ completed: this.checked });
239                 });
240                 break;
241             case 'itemEdit':
242                 $delegate(self.$todoList, 'li label', 'dblclick', function () {
243                     handler({ id: self._itemId(this) });
244                 });
245                 break;
246             case 'itemRemove':
247                 $delegate(self.$todoList, '.destroy', 'click', function () {
248                     handler({ id: self._itemId(this) });
249                 });
250                 break;
251             case 'itemToggle':
252                 $delegate(self.$todoList, '.toggle', 'click', function () {
253                     handler({
254                         id: self._itemId(this),
255                         completed: this.checked,
256                     });
257                 });
258                 break;
259             case 'itemEditDone':
260                 self._bindItemEditDone(handler);
261                 break;
262             case 'itemEditCancel':
263                 self._bindItemEditCancel(handler);
264                 break;
265         }
266
267         // if (event === 'newTodo') {
268         //  $on(self.$newTodo, 'change', function () {
269         //      handler(self.$newTodo.value);
270         //  });
271         // } else if (event === 'removeCompleted') {
272         //  $on(self.$clearCompleted, 'click', function () {
273         //      handler();
274         //  });
275         // } else if (event === 'toggleAll') {
```

Aussi, pour améliorer l'accessibilité et SEO, de petites modifications ont été effectuées :

- Ajout d'une balise <meta name = 'viewport' with initial-scale/>
- Ajout d'une balise <meta description />
- Le contraste a été amélioré

COMPLETER AVEC RAPPORT AUDIT DAREBOOST

4. Axes d'améliorations

- Utiliser un bundler type Webpack ou Parcel.
- Ajouter un compte utilisateur pour une utilisation multiplateformes afin de retrouver ses tâches sur n'importe quel support ou navigateur.
- Ajouter une date à nos tâches, synchronisation avec son calendrier (Google, Outlook…)