Projet 8 - Améliorer un projet existant

Présentation

Dans ce projet, une base de code déjà existante nous était fournie.

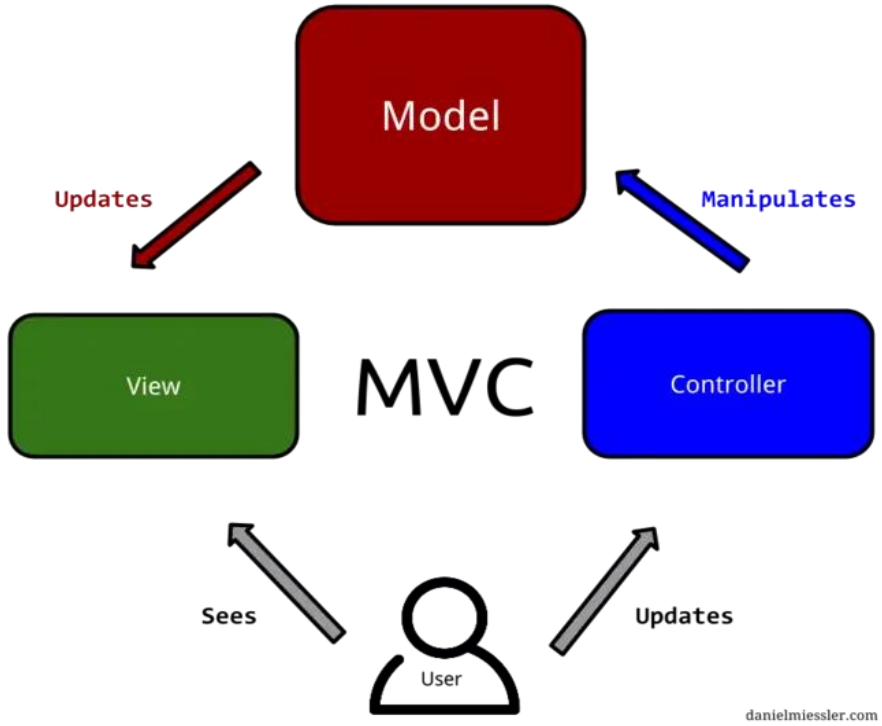
L'architecture de l'application est basée sur MVC, c'est-à-dire Model, View and Controller.

Les principales fonctionnalités de l'application sont les suivantes :

- Possibilité de créer et d'ajouter une tâche.
- Modifiez les tâches existantes.
- Marquer / Décocher tâches comme terminé.
- Effacer les tâches terminées.
- Supprimer les tâches.

L'idée de base de MVC Architecture est qu'un utilisateur voit une « View » lorsqu'il saisit l'URL correcte. Les routes sont gérées par les « Controller » pour rendre la « View », certains « Controller » aident également à déclencher la manipulation dans les « Model », puis la « View » est mise à jour avec les modifications des données.

L'image ci-dessous peut mieux définir le flux.



Les principaux objectifs de ce projet étaient :

- Saisir la base du code
- Trouver et corriger les bugs
- Ecrire des tests dans Jasmine
- Optimiser les performances de l'application
- Analyser la performance d'un site concurrent.

Structure du projet

L'arborescence de base du système de fichiers du projet ressemble à ceci :

```
.
├── index.html
├── js
├── license.md
├── node_modules
├── package.json
├── package-lock.json
└── test

3 directories, 5 files
```

Le répertoire contient quelques fichiers importants comme :

- `index.html` : Ceci est la page de destination de notre application ToDo. Tous les CSS et JS sont chargés.
- `license.md` : Ce fichier contient les détails de la licence du code.
- `package.json` : Ce fichier contient les informations du package, sa version, son auteur et tous les packages installés utilisés dans le projet.
- `package-lock.json` : Ce fichier contient la configuration verrouillée des modules node_modules.

Répertoire

/ js /

```
js
├── app.js
├── controller.js
├── helpers.js
├── model.js
├── store.js
├── template.js
└── view.js

0 directories, 7 files
```

- `app.js` : Il s'agit du fichier qui crée une nouvelle instance de tous les composants MVC requis comme « storage », « model », « template », « view » et « controller ».

- `controller.js` : Ce fichier est responsable de la partie contrôleur du modèle MVC. Il contient toutes les actions requises pour l'application.

- `helpers.js` : Ce fichier comprend des fonctions pour les interactions avec le DOM telles que des modèles de requête DOM complexes.

- `model.js` : Ce fichier est responsable de la partie du modèle MVC. Il contient des fonctions qui gèrent les données de l'application et toutes les choses liées au modèle.

- `store.js` : Ce fichier agit comme la fausse base de données de l'application. Le stockage utilise localStorage qui crée un fichier temporaire sur le PC de l'utilisateur et agit comme une fausse base de données persistante, mais en vidant le cache, cette base de données disparaîtra.

- `template.js` : Ce fichier contient des modèles pour les éléments de liste, les caractères d'échappement HTML et d'autres modèles DOM qui aident notre application.

- `view.js` : Ce fichier contient des fonctionnalités qui nous permettent d'afficher les données à l'utilisateur et de visualiser l'application.

/ node_modules /

```
node_modules
├── jasmine-core
├── todomvc-app-css
└── todomvc-common

3 directories, 0 files
```

- `jasmine-core` : Nous permet d'écrire des tests automatisés avec le framework de test Jasmine qui teste les fonctionnalités de notre application.

- `todomvc-app-css` : Ce package qui contient les styles CSS de l'application héritée de l'application TodoMVC.

- `todomvc-common` : Il contient les fichiers communs nécessaires pour amorcer le projet TodoMVC.

/test/

```
test
├── ControllerSpec.js
└── SpecRunner.html

0 directories, 2 files
```

- `ControllerSpec.js` : Ce fichier contient tous les tests unitaires qui ont été écrits pour cette application à l'aide de Jasmine Framework. Les tests sont écrits dans des suites, puis chaque suite contient des tests.

- `SpecRunner.html` : Ceci est la première page qui montre les résultats des tests automatisés qui ont été exécutés sur notre projet. Le contenu de la page en fonction des tests écrits dans le fichier ControllerSpec.js.

Maintenant en marche vers l'objectif du projet

Bug 1 (Typo `/js/controller.js:95`)
Un caractère « d » en trop dans le nom de la fonction « addItem » ne permettais pas d'ajouter de nouvelles tâches.

Avant :

```
95    Controller.prototype.adddItem = function (title) {
96      var self = this;
97
98      if (title.trim() === '') {
99        return;
100     }
```

Après :

```
95    Controller.prototype.addItem = function (title) {
96      var self = this;
97
98      if (title.trim() === '') {
99        return;
100     }
```

Bug 2 (ID conflit possible @ `/js/store.js:83`)
Les ID générés pour la nouvelle tâche n'ont pas été vérifiés pour leur caractère unique, ce qui entraînera une duplication de l'ID en HTML.

Avant :

```
83      // Generate an ID
84        var newId = "";
85        var charset = "0123456789";
86
87          for (var i = 0; i < 6; i++) {
88          newId += charset.charAt(Math.floor(Math.random() * charset.length));
89      }
90
```

Après :

```
84      // Generate an ID
85      var newId = "";
86      var charset = "0123456789";
87      var idUnique = false;
88      while (idUnique == false) {
89          for (var i = 0; i < 6; i++) {
90              newId += charset.charAt(Math.floor(Math.random() * charset.length));
91          }
92          idUnique = true;
93          todos.forEach(todo => {
94              if (todo.id === newId) {
95                  idUnique = false;
96              }
97          });
98      }
```

2. Écriture des tests dans Jasmine

La base du code existant contenait des commentaires aux emplacement des tests à rajouter :

- Doit afficher des entrées au démarrage :

```
66    it('should show entries on start-up', function () {
67      // new test here
68      var todo = { title: "my todo" };
69      setUpModel([todo]);
70
71      subject.setView("");
72
73      expect(view.render).toHaveBeenCalledWith("showEntries", [todo]);
74    });
```

- Doit afficher les entrées actives :

```
95      it('should show active entries', function () {
96        // new test here
97        var todo = { title: "my todo", completed: false };
98        setUpModel([todo]);
99
100       subject.setView("#/active");
101
102       expect(model.read).toHaveBeenCalledWith(
103         { completed: false },
104         jasmine.any(Function)
105       );
106
107       expect(view.render).toHaveBeenCalledWith("showEntries", [todo]);
108     });
```

- Doit montrer les entrées terminées

```
110     it('should show completed entries', function () {
111       // new test here
112       var todo = { title: "my todo", completed: true };
113       setUpModel([todo]);
114
115       subject.setView("#/completed");
116
117       expect(model.read).toHaveBeenCalledWith(
118         { completed: true },
119         jasmine.any(Function)
120       );
121
122       expect(view.render).toHaveBeenCalledWith("showEntries", [todo]);
123     });
124   });
```

- Devrait afficher le bloc de contenu lorsque des tâches existent

```
126     it('should show the content block when todos exists', function () {
127       setUpModel([{title: 'my todo', completed: true}]);
128
129       subject.setView('');
130
131       expect(view.render).toHaveBeenCalledWith('contentBlockVisibility', {
132         visible: true
133       });
134     });
```

- Devrait mettre en évidence le filtre "Tous" par défaut

```
168     it('should highlight "All" filter by default', function () {
169       // new test here
170       setUpModel([]);
171
172       subject.setView("");
173
174       expect(view.render).toHaveBeenCalledWith("setFilter", "");
175     });
```

- Devrait basculer tous les tâches vers terminés

```
187       beforeEach(function() {
188         var todos = [
189           { title: "one todo", completed: false, id: 42 },
190           { title: "another todo", completed: false, id: 43 }
191         ];
192         setUpModel(todos);
193
194         subject.setView("");
195
196         view.trigger("toggleAll", { completed: true });
197       });
198
199       it('should toggle all todos to completed', function () {
200         // new test here
201         expect(model.update).toHaveBeenCalledWith(
202           42,
203           { completed: true },
204           jasmine.any(Function)
205         );
206         expect(model.update).toHaveBeenCalledWith(
207           43,
208           { completed: true },
209           jasmine.any(Function)
210         );
211       });
```

- Devrait mettre à jour la vue

```
213       it('should update the view', function () {
214         // new test here
215         expect(view.render).toHaveBeenCalledWith("elementComplete", {
216           id: 42,
217           completed: true
218         });
219         expect(view.render).toHaveBeenCalledWith("elementComplete", {
220           id: 43,
221           completed: true
222         });
223       });
```

- Devrait ajouter une nouvelle tâche au modèle

```
227       beforeEach(function() {
228         setUpModel([]);
229         subject.setView("");
230       });
231
232       it('should add a new todo to the model', function () {
233         // new test here
234         view.trigger("newTodo", "a new todo");
235
236         expect(model.create).toHaveBeenCalledWith(
237           "a new todo",
238           jasmine.any(Function)
239         );
240       });
```

- Devrait supprimer une entrée du modèle

```
272       beforeEach(function() {
273         setUpModel([{ title: "my todo", completed: true, id: 37 }]);
274         subject.setView("");
275         view.trigger("itemRemove", { id: 37 });
276       });
277
278       it('should remove an entry from the model', function () {
279         // new test here
280         expect(model.remove).toHaveBeenCalledWith(37, jasmine.any(Function));
281       });
```

3. Optimisations

Dans `/test/ControllerSpec.js` de nombreuses suites de tests avaient des lignes répétées, j'ai donc utilisé beforeEach ce qui a permis de sauver près de 40 lignes de code.

J'ai également modifié une boucle pouvant être optimisée dans `/js/controller.js` :

Avant :

```
158     Controller.prototype.removeItem = function (id) {
159       var self = this;
160       var items;
161       self.model.read(function(data) {
162         items = data;
163       });
164
165       items.forEach(function(item) {
166         if (item.id === id) {
167           console.log("Element with ID: " + id + " has been removed.");
168         }
169       });
170
171       self.model.remove(id, function () {
172         self.view.render('removeItem', id);
173       });
174
175       self._filter();
176     };
```

Après :

```
159     Controller.prototype.removeItem = function(id) {
160         var self = this;
161         var items;
162         self.model.read(function(data) {
163           items = data;
164         });
165
166         // this stops as soon as element with ID is found
167         var found = items.find(function(item) {
168           return item.id === id;
169         });
170
171         if (found) {
172           self.model.remove(id, function() {
173           self.view.render("removeItem", id);
174         });
175
176           self._filter();
177         }
178       };
```

4. Audits de performance d'une application concurrent

Présentation

Le site concurrent http://todolistme.net/ est un ensemble complet de fonctionnalités au nom d'un gestionnaire de tâches.

Il offre des fonctionnalités comme la programmation des tâches, l'impression de toutes les tâches, la synchronisation des tâches avec le compte et l'ouverture d'une vue compatible téléphone.

En outre, la visualisation et l'UX de l'application est beaucoup plus avancée avec la fonctionnalité de glisser-déposer, modals / dialogs, fenêtres pop up.

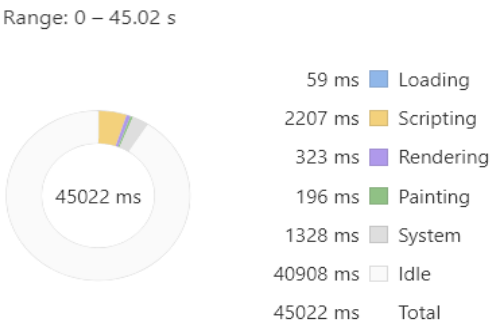
Le CSS me semble bien écrit, mais les requêtes média query auraient pu être améliorées comme lorsque nous faisons la taille de l'écran au carré, il y a des boutons qui sont rognés à partir de l'écran, il est également difficile d'utiliser toutes les fonctionnalités sur un petit site Web.

Le site Web est équipé d'infobulles et d'événements de survol de la souris pour rendre les boutons descriptifs.

Cette application utilise également localStorage comme fausse base de données pour chaque opération sur le site Web.

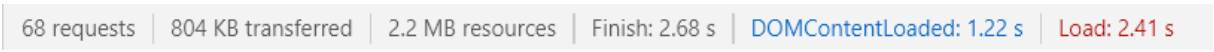
Outils d'audits avec le navigateur GoogleChrome

- Charge de l'application (en ligne)
    - Temps de script : 2207ms
    - Temps de rendu : 323 ms

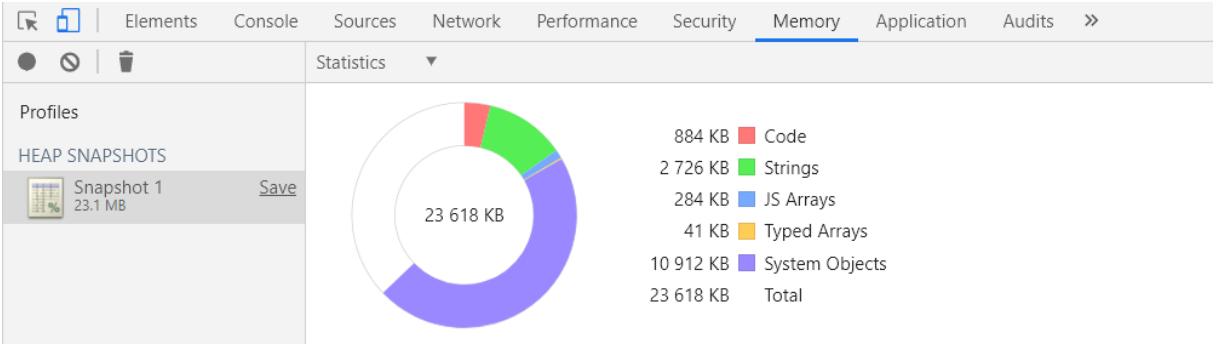


Notable :
L'application récupère jQuery à partir d'un CDN.

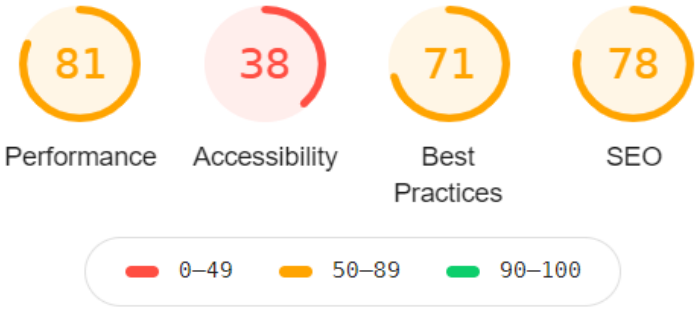
- DOMContentLoaded
    - Temps de chargement : 1.22 secondes



- Mémoire d'application
    - Utilisation de la mémoire : 23 Mo



Global



Conclusion en vue d'amélioration

L'application de notre concurrent me semble excessive pour la tâche qui doit être effectuée ici. La dépendance CDN et jQuery nuisent à la performance de l'application. Outre la performance, l'application offre une mauvaise accessibilité, des « Best Practices » manquantes, et un SEO optimisable.
Des normes qui ne sont pas respectés sont repéré et listé par le navigateur et doivent être appliqué.

En vue de l'amélioration de notre application, l'évolution de l'environnement UX et l'ajouts de fonctionnalités me semble être une étape tout à fait pertinente pour la suite.
Au niveau du code en lui-même, nous avons utilisé Vanilla JS avec une syntaxe ES6 et une implémentation légère de MVC qui est tout à fait adapté.