Openclassrooms - Développeur d'applications front end - Projet 8

Etape 1 : Corrigez les bugs

Le premier bug, la faute de frappe, se trouve dans le fichier controller.js à la ligne 95. La partie du code suivant « Controller.prototype.adddItem » comprend une lettre « d » en trop qui créer un bug. La partie du code « Controller.prototype.adddItem » a donc été remplacé par « Controller.prototype.addItem ».

Le second bug introduit un conflit éventuel entre deux Ids identiques. En effet dans le fichier store.js, la boucle suivante, à la ligne 87, ne vérifie pas au préalable si l'identifiant passé n'est pas au préalable utilisé :

```
for (var i = 0; i < 6; i++) {
    newId += charset.charAt(Math.floor(Math.random() * charset.length));
}
```

Pour corriger ce bug nous avons remplacé la boucle par le code suivant :

```
let isUnique = false;

while (!isUnique){
  for (let i = 0; i < 6; i++) {
    newId += charset.charAt(Math.floor(Math.random() * charset.length));
  }
  isUnique = true;
  for (let i = 0; i < todos.length; i++) {
    if (todos[i].id == newId) {
      isUnique = false;
    }
  }
}
```

Combinés avec cela, nous ajoutons un « id » aux inputs situé dans le « main de l'application. Pour cela nous modifions la ligne 16 du fichier index.html.

```
<section class="main">
    <input class="toggle-all" id="toggle-all" type="checkbox">
    <label for="toggle-all">Mark all as complete</label>
    <ul class="todo-list"></ul>
</section>
```

Etape 2 : Ou sont les tests ?

TEST N°1 :
- describe : controller
- it : should show entries on start-up
- ligne : 62

TEST N°2 :
- describe : routing
- it : should show active entries
- ligne : 95

TEST N°3 :
- describe : routing
- it : should show completed entries
- ligne : 105

TEST N°4 :
- describe : routing
- it : should highlight "All" filter by default
- ligne : 158

TEST N°5 :
- describe : routing
- it : should highlight "Active" filter when switching to active view
- ligne : 168

TEST N°6 :
- describe : toggle all
- it : should toggle all todos to completed
- ligne : 179

TEST N°7 :
- describe : toggle all
- it : should update the view
- ligne : 192

TEST N°8 :
- describe : new todo
- it : should add a new todo to the model
- ligne : 207

TEST N°9 :
- describe : element removal
- it : should remove an entry from the model
- ligne : 255

Etape 3 : optimisez la performance

1. Analyse de l'application « todolistme ».

1.1 Présentation de l'application

L'application « todolisme » est une applation permettant à ses utilisateurs de gérer et d'organiser ses tâches. L'utilisateur peut en effet entrer dans l'application les différentes tâches qu'il souhaite accomplir qui apparaîtront dans la liste « to do » de l'application. Une fois la tâche réalisé, l'utilisateur peut cocher cette dernière qui apparaîtra dorénavant dans la liste « done ».

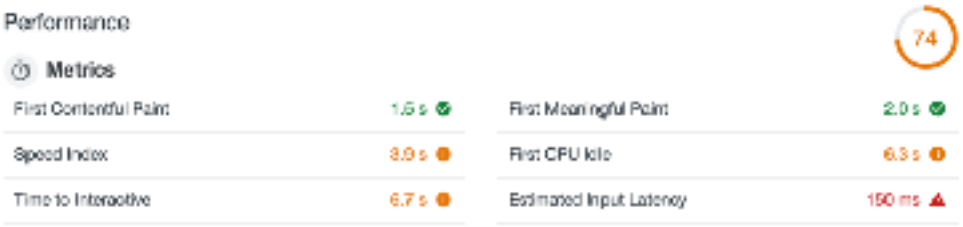
L'application est sophistiquée et assez complète puisqu'elle dispose d'autres fonctionnalités très utiles. En effet, l'application propose de gérer les tâches qui sont à réaliser au lendemain ou plus tard. Le système de drag and drop est également très utile et pratique pour à l'utilisation. Autre fonctionnalité, il est possible pour l'utilisateur d'organiser ses tâches par catégories ou par liste. Pour finir, un système de filtre est aussi proposé, avec divers critères possibles : ordre alphabétique, aléatoire ou bien par top 3.

1.2 Analyse de la performance

Audits de performance :

Afin de réaliser l'audit suivant, l'outil « Lighthouse » du dev tools du navigateur google chrome a été utilisé. Google Lighthouse permet de réaliser un audit de performance rapide d'un site internet ou d'une application web. L'audit prends en compte 5 éléments dans l'analyse qui apparaîtront ensuite avec un score associé. Pour cet audit, nous prendrons en compte uniquement les trois indicateurs suivants : performance, best practices, accessibility.

Performance :



L'audit de performance réalisée donne un score de 74 sur 100, ce qui peut être considéré comme un score moyen et perfectible. Si l'on regarde plus en détails l'analyse de performance nous pouvons y trouver divers informations.

Les temps obtenus pour le First Contentful Paint ainsi que pour le First Meaningful Paint considérés comme satisfaisants, bien que perfectibles.

Dans les points à améliorer, nous pouvons remarquer les trois indicateurs de performance suivants :
- Speed Index
- First CPU Idle
- Time to Interactive

Pour finir, les 150 millisecondes obtenues pour l'indicateur « Estimated Input Latency » sont considérées comme mauvais et sont un point à améliorer d'urgence. Pour résumer, l'audit de performance réalisé pour le site internet « http://todolistme.net/" nous montre que les performances du site internet sont plutôt moyennes et quelles peuvent être grandement améliorées.

Best Practices :

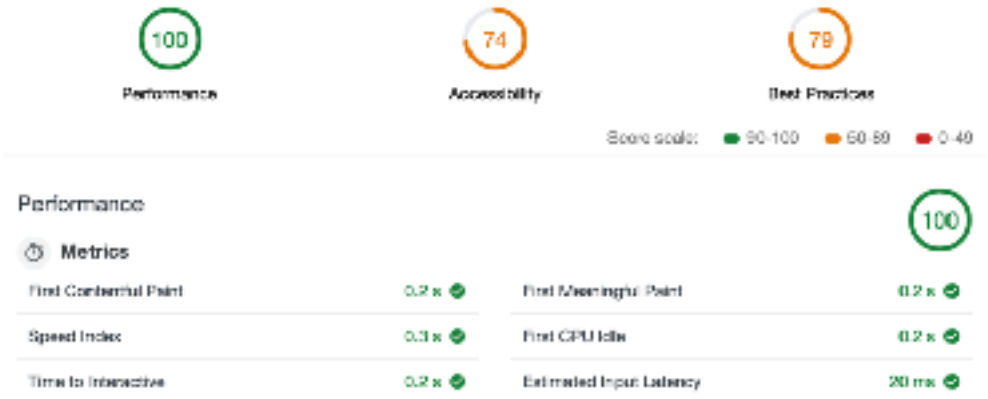
Concernant les Best Practices, les meilleures pratiques, le score obtenu est de 64. Ce score n'est pas satisfaisant et plusieurs améliorations sont à effectuer. Premièrement, l'utilisation du protocole HTTPS pour une plus grande sécurité ainsi qu'un meilleur référencement. Dans un second temps, utiliser le protocole HTTP/2.0 pour une meilleure gestion des flux. Le troisième point relevé est l'utilisation d'une librairie Javascript ayant une sécurité vulnérable. Quatrième point, plusieurs erreurs dans la console sont détectés. Pour finir, plusieurs images ont un ratio incorrect.

Accesibility :

Pour finir avec l'audit de performance, l'accessibilité. Ici le score obtenu est de 47, ce qui peut être considéré comme mauvais. Plusieurs points sont à revoir afin d'obtenir un meilleur résultat :
- les couleurs utilisées en background n'ont pas un contraste de couleurs suffisant
- les attributs sur les pages ne sont pas uniques
- les éléments <frame> ou <iframe> n'ont pas de titre
- les éléments images n'ont pas d'attributs [alt]
- les éléments de formulaire n'ont pas d'étiquette associée
- l'élément <html> n'a pas d'attribut [lang]

2. Comparaison des deux applications

A titre de comparaison, voici les résultats que nous avons obtenus pour l'audit de notre application « to-do-list-app ».



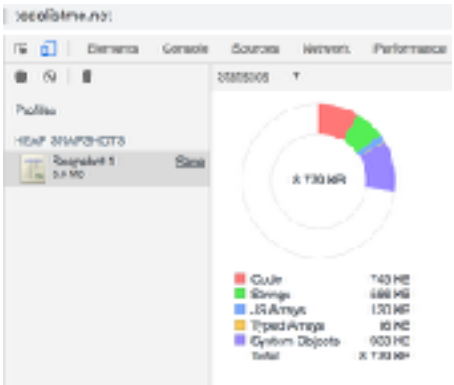
Notre application a obtenu un score de 100 à l'audit de performance. Un score donc bien meilleur que celui obtenus par le site concurrentiel qui était de 74. Si l'on regarde plus en détails, notre application a obtenu un meilleur résultat sur 5 de 6 indicateurs de performance, et les 6 résultats sont considérés comme satisfaisants selon Google Lighthouse.

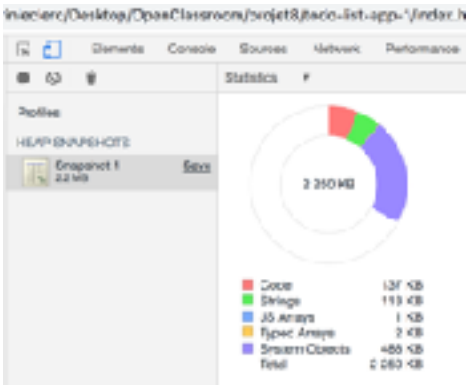
Concernant l'accessibilité, notre application a obtenu un score de 74, ce qui est 10 points de plus que l'application « todolisme ». Pour finir, nous obtenons un résultat de 74 concernant les bonnes pratiques à adoptés, ce qui est 32 points de plus que les 47 obtenus par l'application « todolisme ».

Les différences :

Différentes explications peuvent nous aider à mieux comprendre ces résultats. Tout d'abord, l'application « todolistme » semble plus sophistiquée et propose beaucoup plus de fonctionnalité que notre application. En effet, l'application concurrente propose des fonctionnalités telles que le glissez-déposez, le fait d'avoir plusieurs listes, de trier ses tâches ou encore d'enregistrer ses listes. Toutes ces fonctionnalités peuvent en partie expliquer le niveau de performance moins bon.

En se basant sur l'audit, nous remarquons également que notre application se base uniquement sur des fichiers Javascript, Html et Css et ne fait appel à aucun fichier multimédia tel que images ou encore des vidéos. Ceci explique en partie le fait que notre application ait un temps de chargement rapide, que peu de mémoire est utilisé et que le contenu s'affiche de manière relativement rapide voir instantanément.





Pour terminer la comparaison entre les deux applications, voici un tableau récapitulatif des points forts et faibles de chacune d'elles.

http://todolistme.net/	
Points forts	Points faibles
Application sophistiquée	Forte consommation de mémoire
Beaucoup de fonctionnalités	

todo-list-app	
Points forts	Points faibles
Chargement rapide	Peu de fonctionnalité
Faible consommation de mémoire	

Bilan :

Suite aux différentes analyses que nous avons effectuées, nous pouvons en conclure que notre application est bien plus performante que l'application concurrente « todolistme ». Cependant, cette dernière semble bien plus complète et sophistiquée au niveau des fonctionnalités et des possibilités offertes à l'utilisateur.

Dans l'hypothèse de scaler notre application, l'ajout de diverses fonctionnalités telles que le système de filtre, de rangement par catégorie ou encore la possibilité de remettre certaines tâches au lendemain serait un vrai plus.

Etape 4 : Améliorez le projet

1. Documentation fonctionnelle

Notre application a pour objectif de permettre à l'utilisateur de gérer ses tâches de manière simple et très efficace. L'application est construite et s'utilise autour de 4 actions principales qui sont les suivantes : ajouter une tâche, modifier une tâche, supprimer une tâche, filtrer les tâches (active ou completed).

Voici comment se présente l'application à son lancement :

1.1 Ajouter une tâche

Afin d'ajouter une tâche à notre application, il suffit simplement de cliquer sur l'air de texte ou est écrit « What needs to be done », d'écrire le nom de notre tâche, et d'appuyer sur la touche « entrée ».

Une fois la tâche ajouté, cette dernière apparaît dans liste des tâches.

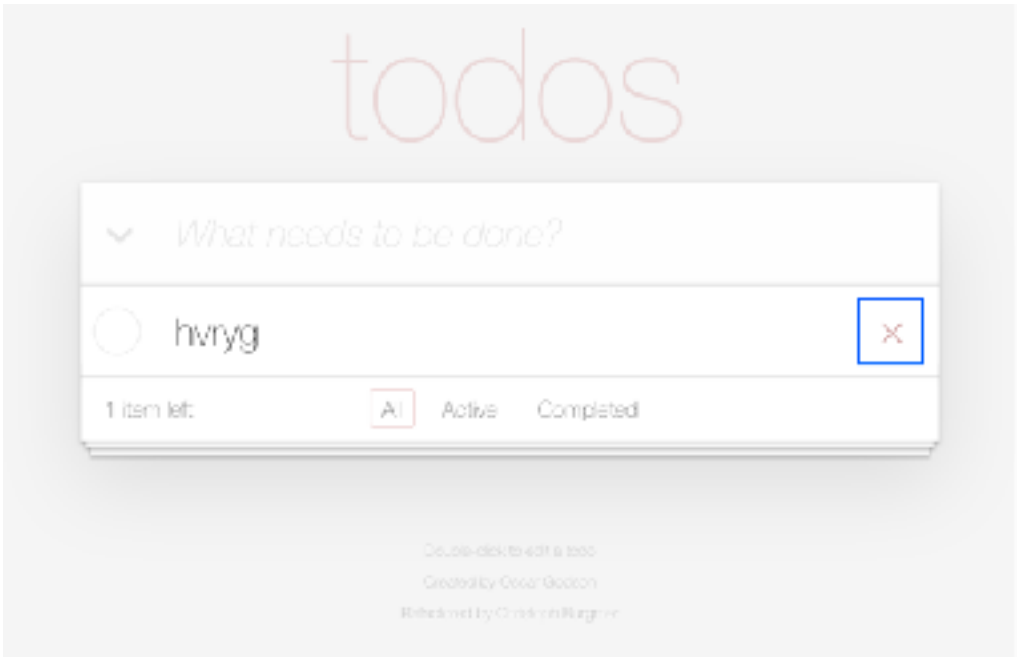
1.2 Modifier une tâche

Afin de modifier une tâche, il suffit simplement de double-cliquer sur celle que nous souhaitons modifier. Dans l'exemple-ci nous souhaitons modifier notre tâche « première tâche ». Une fois le double-clique effectué, la tâche est entourée d'une ombre grise, nous pouvons dorénavant modifier le nom de la tâche.

Une fois que nous avons modifié le nom, il suffit de cliquer sur la touche « entrée », et la modification est prise en compte.

1.3 Supprimer une tâche

Pour supprimer une tâche sur l'application, deux méthodes sont possibles. La première consiste à cliquer sur la croix rouge qui s'affiche au survol d'une tâche. Une fois le clique effectué, la tâche est directement supprimée.



La seconde méthode, consiste à cocher la tâche comme effectué. Une fois que la tâche fait partie de la liste des tâche effectuées, un bouton « Clear completed » apparaît, cliquer une fois dessus. Cela aura pour effet de supprimer toutes les tâches déjà effectuées.

1.4 Filtrer les tâches

Pour terminer, nous allons voir comment filtrer les tâches. Trois filtres sont possibles et proposés. Le premier filtre, « all », qui comme son nom l'indique affiche l'ensemble des tâches de l'application. Le second filtre, « active » affiche uniquement les tâches pas encore effectués. Pour finir, le troisième et dernier filtre, « completed » affichent uniquement les tâches déjà effectuées.

Filtre all :

Filtre « active » :

Filtre « completed » :

2. Documentation technique

L'application fonctionne avec un ensemble de fichiers javascript qui sont les suivants :
- app.js
- controller.js
- helpers.js
- model.js
- store.js

- template.js
- view.js

Une description de chacun de ces fichiers et de leur composition est effectué dans la suite de ce document. Il est important de noter que l'application ne fait appel à aucun framework ou librairie externe.

L'architecture MVC :

L'application est construite selon le modèle MVC, qui signifie Modèle - Vue - Contrôleur. Le but du modèle MVC est de séparer la logique du code en trois parties, que l'on retrouve dans des fichiers distincts :
- Modèle : gère les données de l'application. Son rôle est de récupérer les informations, de les organiser et de les assembler pour qu'elle puisse ensuite être traitée par le contrôleur.
- Vue : gère l'affichage de l'application. Elle reçoit également les interactions de l'utilisateur dont elle va déléguer la gestion au contrôleur.
- Contrôleur : assure le lien entre le modèle et la vue. Il s'occupe de la réception des données et des interactions avec l'utilisateur puis les traites en modifiant le modèle.

Les avantages d'une architecture MVC sont multiples. Tout d'abord, la conception se veut plus claire et efficace grâce à la séparation des données de la vue et du contrôleur. Cela permet également une plus grande souplesse pour organiser le développement entre différents développeurs. Pour finir, cela rend la maintenance et l'évolution d'une application plus simple et plus rapide.

Dans le cas de notre application, la classe Model présente dans le fichier model.js correspond au modèle de l'application. La classe Model va définir les différentes méthodes de création et de mises à jour des tâches en mémoire. La mémoire elle est gérée par la classe Store, présente dans le fichier store.js, qui occupe le rôle de base de données de l'application. La mémoire de notre application est géré par le biais du stockage local du navigateur et non pas par une véritable base de données. La vue de l'application est quant à elle géré par la classe View présente dans le fichier view.js. Son rôle est de traiter les différents événements de l'application, de transmettre l'ajout et les modifications des tâches et bien évidemment de mettre à jour la vue de l'application. Pour cela, la classe View fait appel à la classe Template présent dans le fichier template.js. La classe Contrôler, présente dans le fichier Controller.js a pour rôle de faire le lien entre le modèle et la vue de l'application. Elle va donner les instructions au modèle puis demander à la vue de mettre à l'application à jour. La classe Todo présente dans le fichier app.js, s'occupe de créer une instance de chacune des classes présentée précédemment (Store, Model, Template, View et Controller). Pour finir, le fichier helper.js à deux objectifs. Le premier est de déclarer un ensemble de méthodes récurrentes dans l'application. Le second est de permettre à la personne utilisant techniquement l'application de mieux comprendre certaines portions de codes des autres fichiers.

Ci-dessous, le rôle de chacune des fonctions et méthodes présentes dans les différents fichiers javascript de notre application.

app.js :
- Todo : création d'une nouvelle todo (tâche)
- setView : mise à jour de la vue de l'application au chargement et en fonction du changement d'url

controller.js :
- Controller : fait le lien entre le modèle et la vue
- Controller.prototype.setView : charge et initialise la vue
- Controller.prototype.showAll : récupère les todos à partir du modèle et les affiche via la vue
- Controller.prototype.showActive : récupère les todos active du modèle et les affiche grâce à la vue
- Controller.prototype.showCompleted : récupère les todos complété du modèle et les affiche grâce à la vue

- Controller.prototype.addItem : ajout d'une nouvelle todo
- Controller.prototype.editItem : édite le titre d'une todo
- Controller.prototype.editItemSave : enregistre le titre modifiée d'une todo
- Controller.prototype.editItemCancel : annule la modification du titre d'une todo
- Controller.prototype.removeItem : supprime une todo
- Controller.prototype.removeCompletedItems : suppression des todos complétées
- Controller.prototype.toggleComplete : modifie le statut d'une todo complété
- Controller.prototype.toggleAll : modifie le statut de toutes les todos
- Controller.prototype.updateCount : mise à jour du nombre de todo
- Controller.prototype.filter : filtre les todos selon la sélection (All - Active - Completed)
- Controller.prototype.updateFilterState : mise à jour du filtre

model.js :
- Model.prototype.create : création d'une nouvelle todo
- Model.prototype.read : recherche et retourne une todo
- Model.prototype.update : mise à jour des données d'une todo
- Model.prototype.remove : suppression d'une todo dans le modèle de stockage
- Model.prototype.removeAll : supprime l'ensemble des données du modèle de stockage
- Model.prototypeGetCount : retourne le nombre de todos contenue dans le modèle de stockage

template.js :
- Template.prototype.show : insertion de la todo dans le template HTML par défaut
- Template.prototype.itemCounter : affiche le nombre de todo active
- Template.prototype.clearCompletedButton : mise à jour du bouton « clear completed »

store.js :
- Store.prototype.find : trouve une todo par le biais d'une requête
- Store.prototype.findAll : trouve toutes les todos
- Store.prototype.save : enregistre la todo dans le modèle de stockage
- Store.prototype.remove : supprime une todo du modèle de stockage
- Store.prototype.drop : supprime toutes les todos du modèle de stockage

view.js :
- View.prototype._removeItem : retire de la vue une todo
- View.prototype._clearCompletedButton : affiche ou cache le bouton « clear completed »
- View.prototype._setFilter : mise en valeur du bouton de filtre sélectionné
- View.prototype._elementComplete : affiche une todo comme complété ou non
- View.prototype._editItem : affichage du mode édition de la todo sélectionné
- View.prototype._editItemDone : quitte le mode édition de la todo
- View.prototype._render : exécute la commande passée en paramètre et appel la méthode privée correspondante
- View.prototype._itemId : récupération de l'ID d'une todo
- View.prototype.bindItemEditDone : quitte le mode édition
- View.prototype.bindItemEditCancel : quitte le mode édition en appuyant sur la touche « escape »
- View.prototype.bind : fait le lien entre éléments de la vue et du controller