Sommaire

**Nom du projet : To-do List**

projet 8

reprenez et ameliorez un projet  
existant

Sommaire

1. **Présentation du projet**
2. **Correction des bugs**
3. **Amélioration du code**
4. **Test Jasmine**
5. **Usage non technique**
6. **Documentation technique**
7. **Audit de performance du concurrent**
8. **Audit de performance de TodoList**
9. PRESENTATION DU PROJET

Ce projet consiste à reprendre un projet déjà existant, de l’analyser et de proposer des solutions pour l’optimiser.

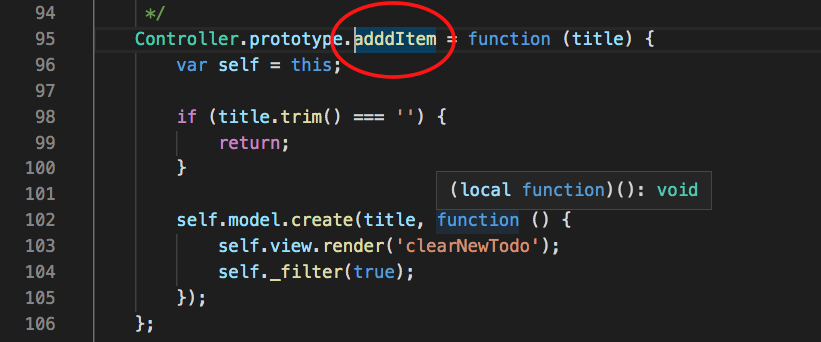
L’application **TodoList** a pour fonctionnalités de :

* Créer une tâche
* Modifiez la tâche
* Cocher ou Décocher la tâche comme terminée
* Effacer la tâche terminée
* Supprimer une tâche

L’application ne fonctionne pas encore correctement et n’est pas optimisé. Nous allons essayer de résoudre ces bugs.

1. correction des bugs

**Bug numéro 1 Faute de frappe**

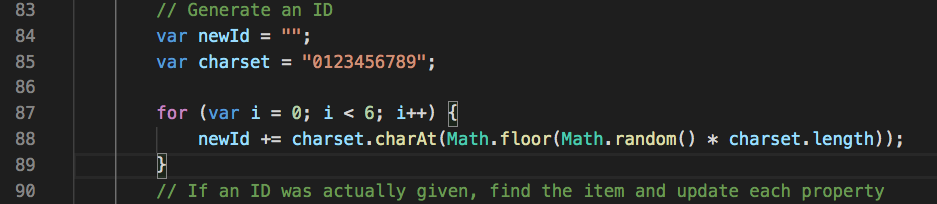


* Dans le fichier **controller.js** **Ligne 95**. Un caractère « d » est en trop dans le nom de la fonction « **addItem** ». Il empêchait à l’application d’ajouter une tâche.

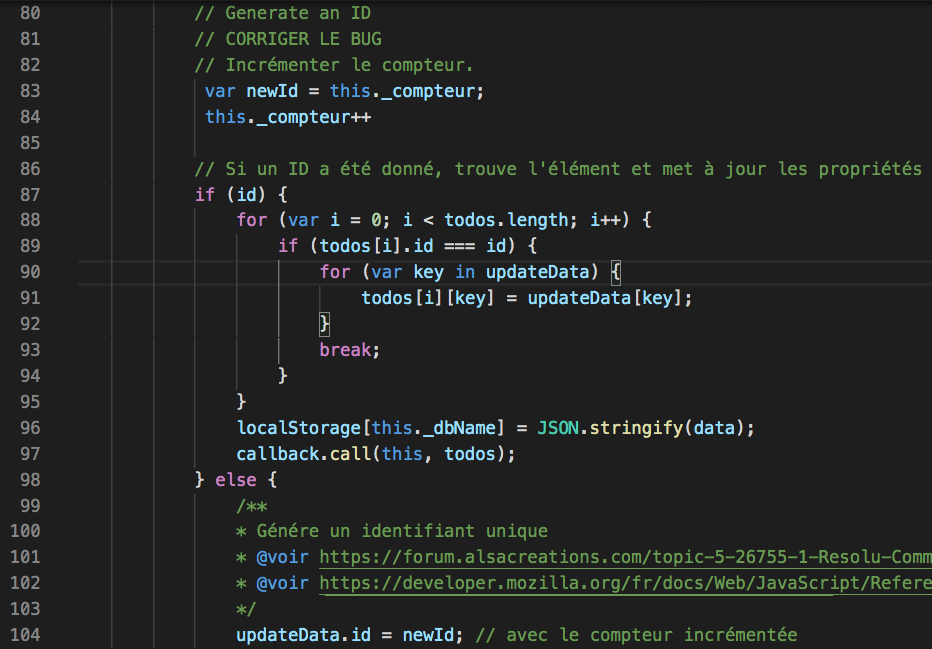
**Bug numéro 2 Conflit sur les IDs**

Dans le fichier store.js **Ligne 83**. Le bug introduit un conflit éventuel entre deux IDs identiques. La variable **newID** risque d’être strictement égale à la variable **id** dans un cas éventuel.

**Avant :**



**Après :**



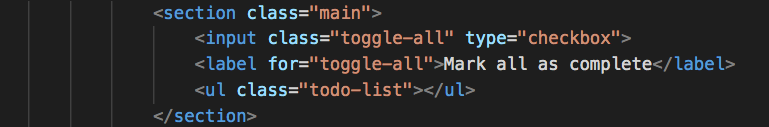
Pour Éviter les conflits entre les **ID’s** des **Todo** :

J’ai créé un **compteur** associé à la variable **newID** qui s’incrémente à chaque fois s’il n’y a pas de id disponible. Nous mettons à jour le newId avec la valeur du compteur incrémenté qui sera unique.

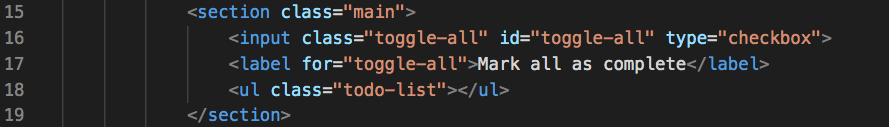
**Bug numéro 3 (facultatif)**

Dans le fichier **index.html ligne 15**. Il manque un id à la balise input.

**Avant :**



**Après :**



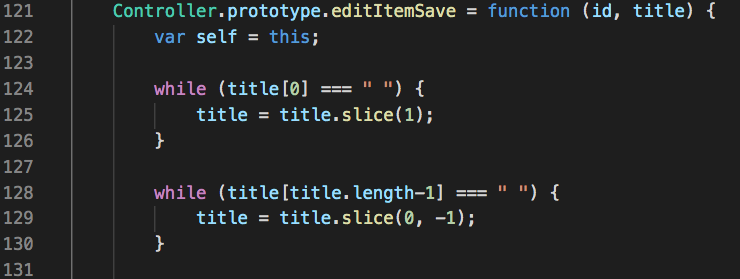
Par convention L'attribut "for" de la balise <label> doit contenir l'id de son input.

1. amelioration du code

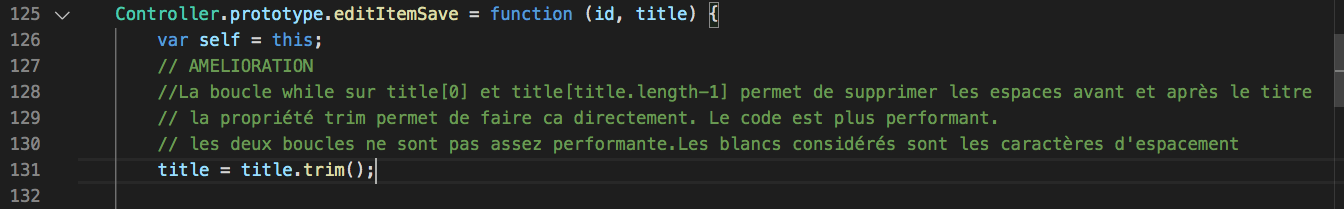
Le code n’est pas optimisé, nous pouvons y ajouter des améliorations, même s'il ne s'agit pas de bugs proprement dit :

* Dans le fichier **controller.js ligne 121** les deux boucles **while**.

**Avant :**



**Après :**

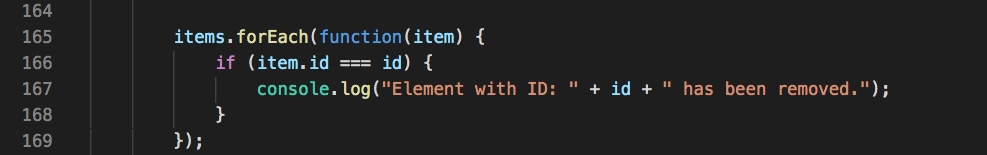


Les deux boucles permettaient de supprimer les espaces au début et à la fin du titre.

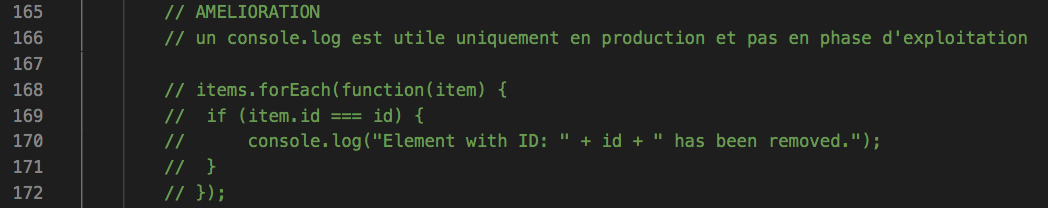
La propriété **trim()** permet de faire la même chose en une seule ligne de code.

* Dans le fichier **controller.js ligne 165** la boucle **forEach** n’est pas nécessaire.

**Avant :**



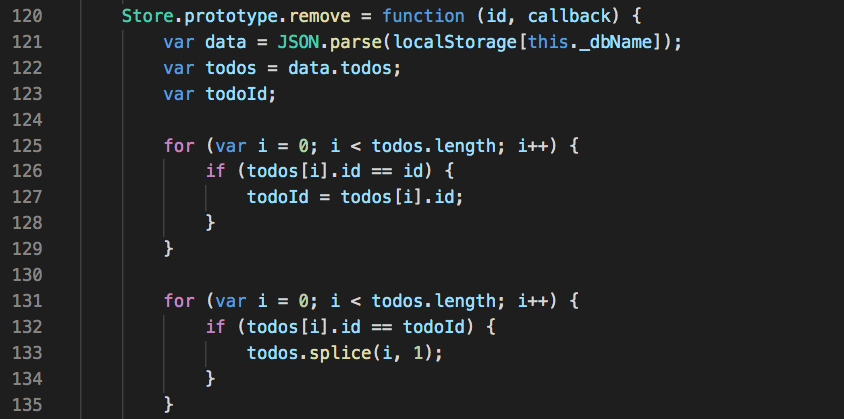
**Après :**



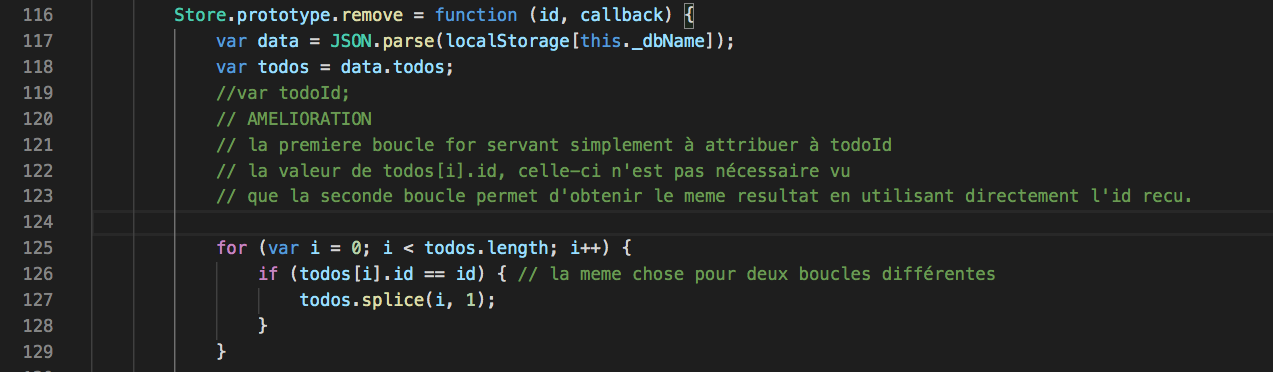
Un console.log est utile uniquement en phase de production et non en phase d'exploitation. La boucle **forEach** est donc inutile.

* Dans le fichier **store.js ligne120**. Les deux boucles **for** peuvent être améliorées.

**Avant :**



**Après :**



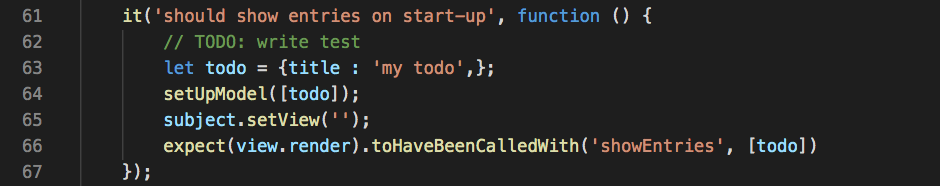
La Fusion des deux conditions **If** en une seule, le premier **if** servait à attribuer à la variable **todoId** un id, bien que le deuxième **if** fait la même chose.

1. test jasmine

Les **Test unitaires** vérifient qu’un élément précis fonctionne correctement.

Une suite de test commence par un appel à la fonction Jasmine **(describe)** avec deux paramètres : **une chaine de caractère** et **une fonction**. La chaine est un titre qui définit ce qui est testé. La fonction est un bloc de code contenant le test lui-même.

* Teste les entrées au démarrage **Ligne 61** :



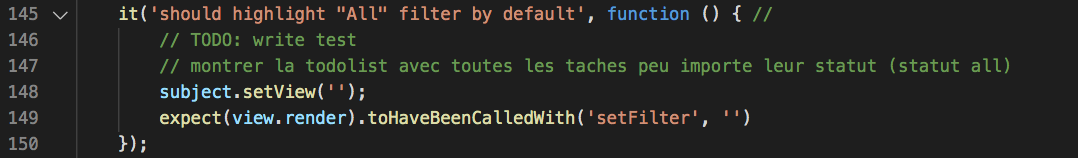
* Teste les entrées avec le statut actif **Ligne 85**  :



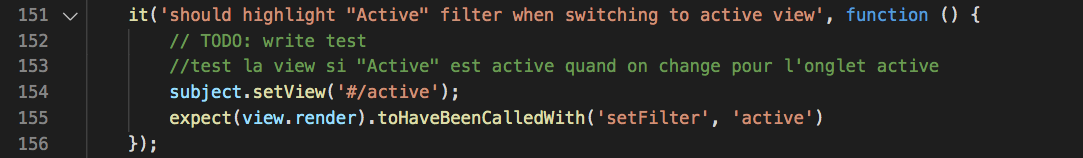
* Teste les entrées avec le statut terminées **Ligne 94** :



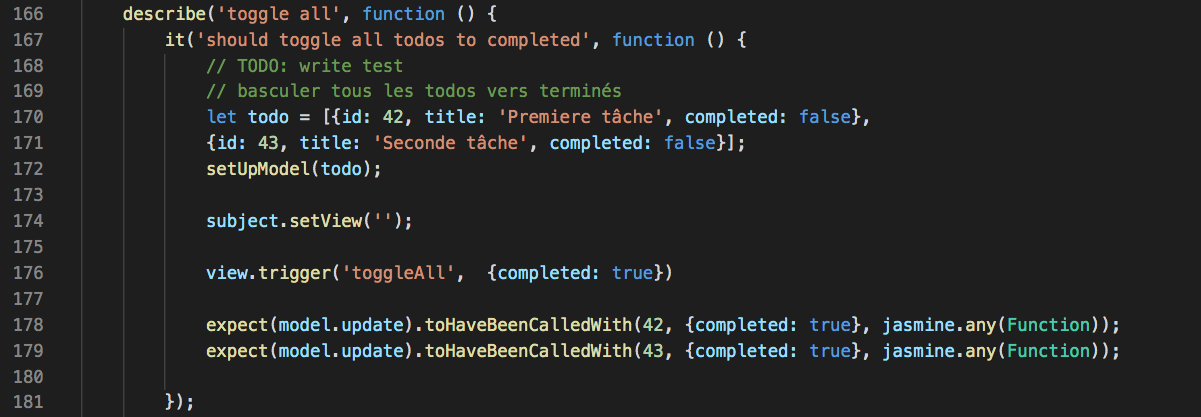
* Teste la vue avec le filtre ‘Tous’ par défaut **Ligne 145** :



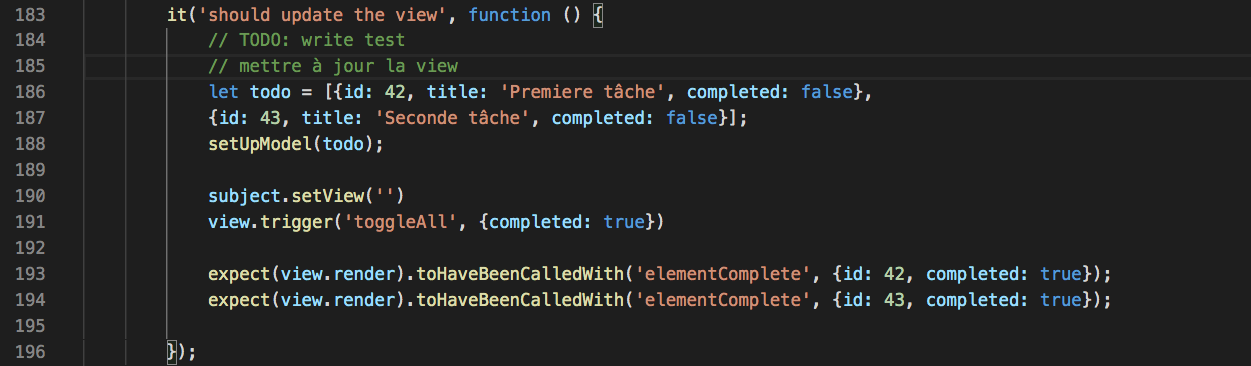
* Teste la vue quand le filtre ‘Actif’ quand on switch sur l’onglet ‘actif’ **Ligne 151** :



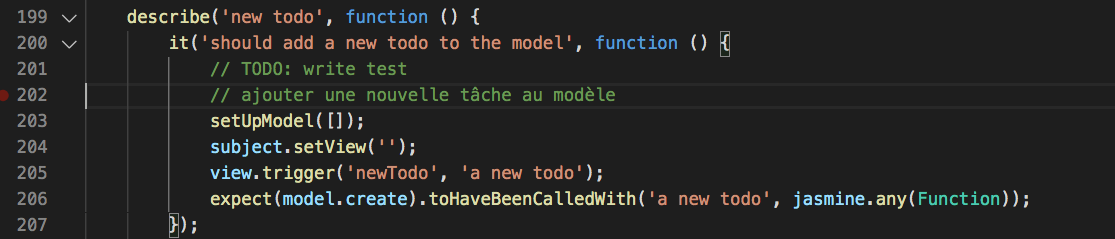
* Teste la mise à jour vers les taches terminées **Ligne 166** :



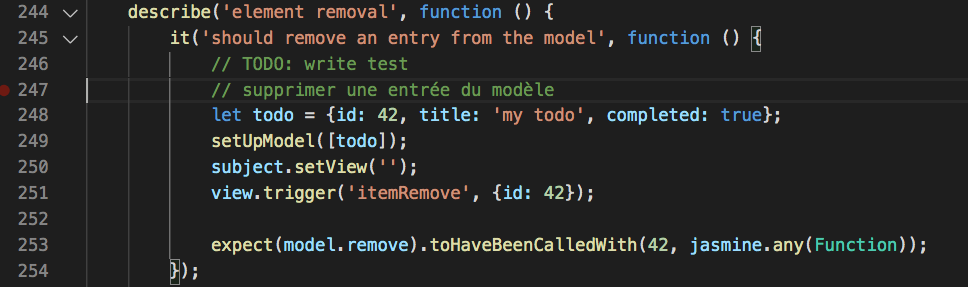
* Teste la mise à jour de la vue **Ligne 183** :



* Teste l’ajout d’une nouvelle tâche au modèle **Ligne 199** :

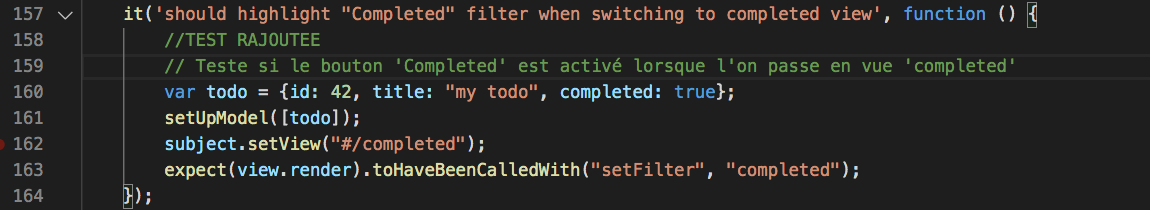


* Teste la suppression d’une tache dans le modèle **Ligne 244** :

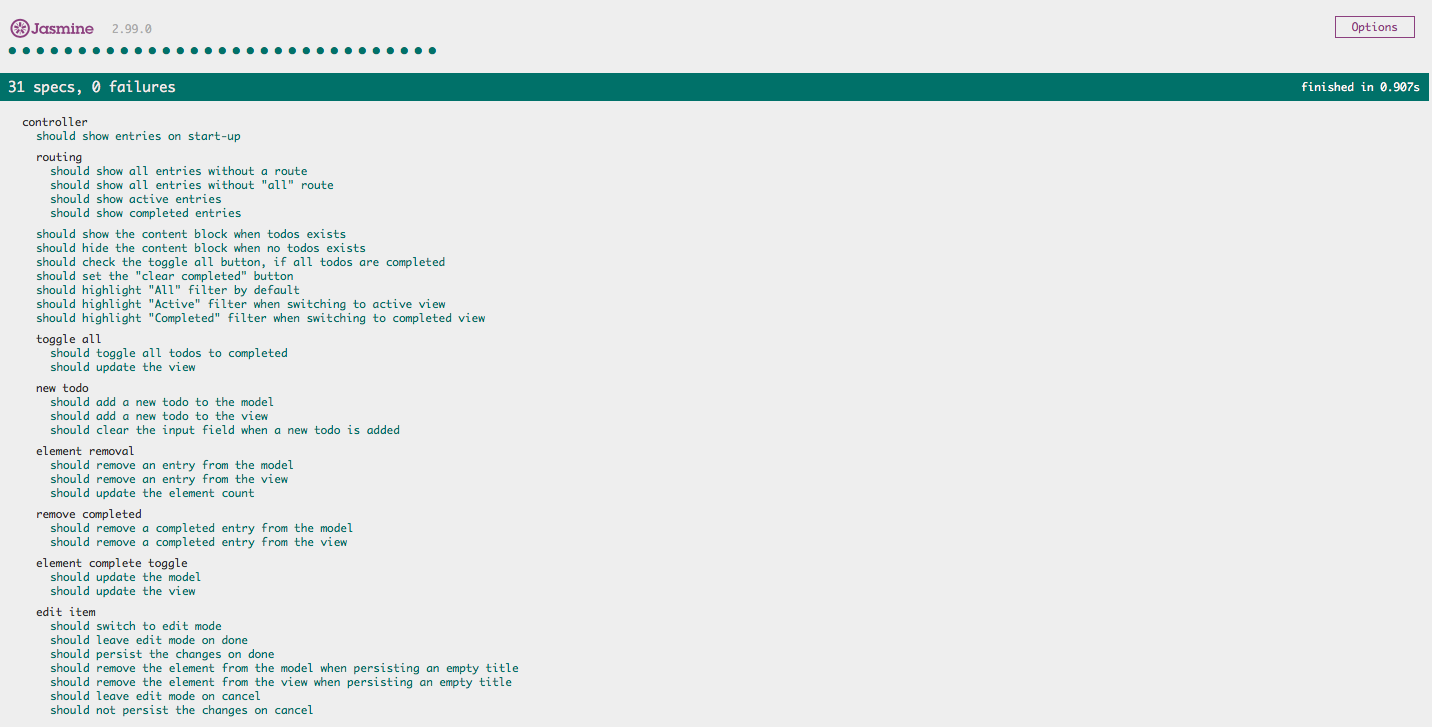


Tests supplémentaires

* Teste l’onglet ‘Complétés’ quand on switch sur le filtre ‘completed’ **Ligne 157**

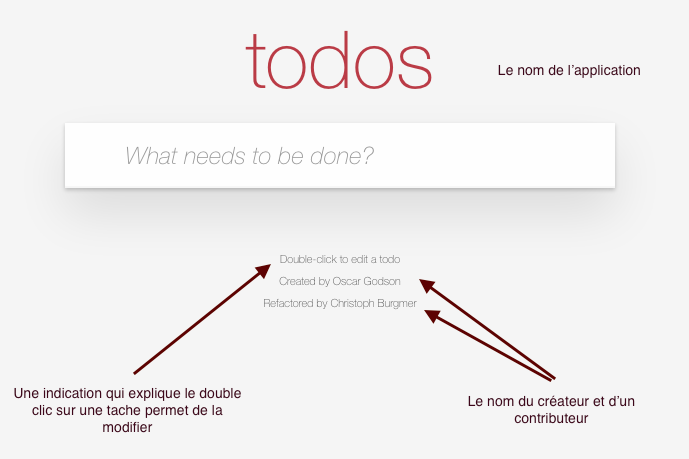


Le fichier **SpecRunner.html** propre à Jasmine montre la validité des tests réalisés fonction de leurs catégories.



1. documentation non technique

Notre application se présente de la façon suivante.

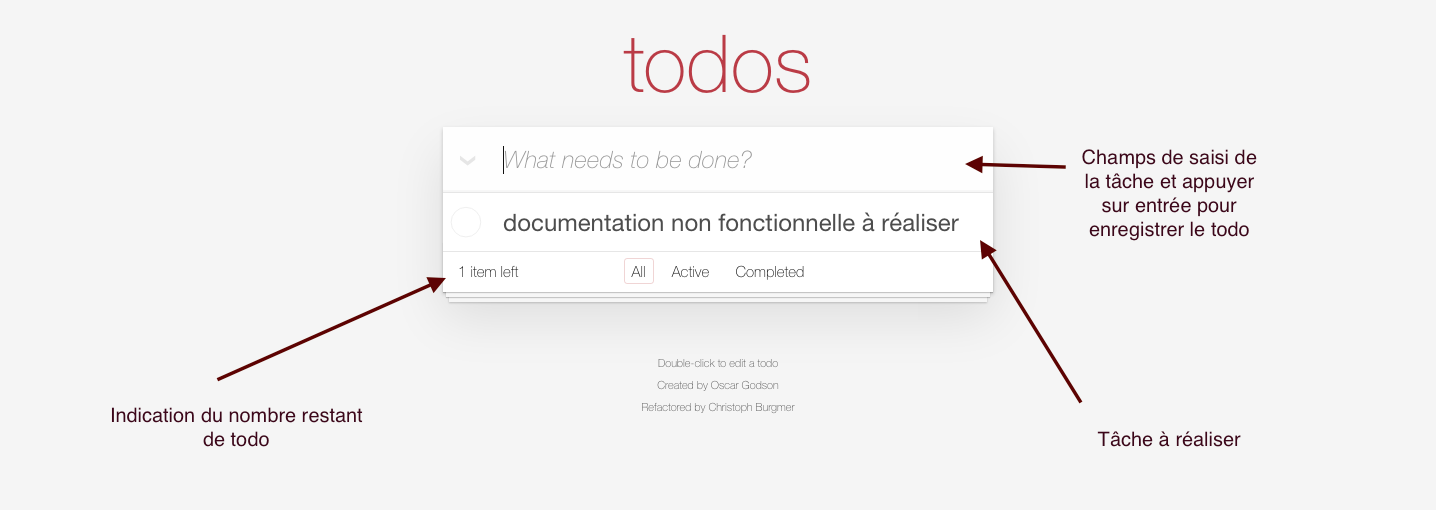


1. **Créer un todo**

Pour créer un Todo cliquer dans « what needs to be done ? » et rédiger votre todo.

Appuyer sur ***entrée*** pour enregistrer le todo.

Vous avez créé votre premier todo.

****

Plusieurs éléments sont apparus :

* Une indication du nombre de todos restant
* Un onglet **all**
* Un onglet **active**
* Un onglet **completed**

1. **Modifier un todo**

Pour modifier un todo, il vous faut double cliquer sur le todo à modifier.

Taper sur *Entrée* pour valider la modification ou sur *échap* pour annuler.

****

1. **Supprimer un todo**

Nous pouvons supprimer le todo. Pour se faire lorsque l’on survole un todo avec notre souris une croix à droite apparaît.

Cliquer sur la croix supprime définitivement le todo.

****

1. **Voir l’état des todos**

Le bouton **All** nous montre l’ensemble des Todos quelques soit leur état.

Le bouton **Active** nous montre les Todos en cours.

Le bouton **Completed** quant à lui nous montre les todos que nous avons terminé.

1. DOCUMENTATION TECHNIQUE

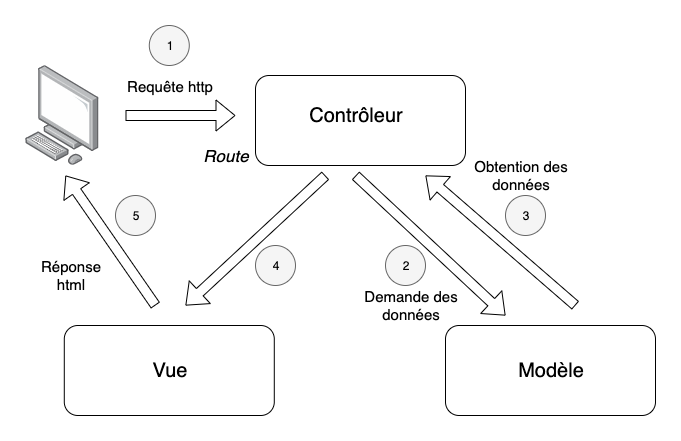
**(Voir document « Documentation fonctionnelle et technique »)**

L’application **« To-do list »** est architecturé sur le pattern design **Modèle-vue-contrôleur** très populaires pour les applications web.

Le motif est composé de trois types de modules ayant trois responsabilités différentes : les modèles, les vues et les contrôleurs.

* **Un modèle (Model)** contient les données à afficher.
* **Une vue (View)** contient la présentation de l'interface graphique.
* **Un contrôleur (Controller**) contient la logique concernant les actions effectuées par l'utilisateur.

Ci-dessous le schéma représentant le pattern design MVC :



1. Arborescence du projet

L’application est composée de 3 dossiers et 5 fichiers nécessaire à son exécution :



Certains fichiers méritent des explications :

* Le fichier **license.md** contient les détails de la licence et de permission MIT
* Le fichier **package.json** estinstallé à la racine du projet en tant que **gestionnaire de packet** npm pour installer les dépendances.
* Le fichier **package-lock.json** contient la configuration verrouillée des modules qui est installé automatiquement avec le gestionnaire de paquet.

1. Les fichiers Javascript

Le code de chaque fichier Javascript est inclus au sein d’une fonction IIFE (fonction auto invoqué et immédiatement).

* js
* ├── app.js
* ├── controller.js
* ├── helpers.js
* ├── model.js
* ├── store.js
* ├── template.js
* └── view.js
* 0 directories, 7 files

1. Le fichier **app.js** est le point d’entrée de l’application et le moteur de l’application.
2. Le fichier **controller.js** contient toutes les actions requises pour l'application. Il appelle les méthodes pour la modification des données.
3. Le fichier **helpers.js** fournit des raccourcies utilisés dans les fichiers principaux pour alléger le code dans sa globalité.
4. Le fichier **model.js** définit le Model et ses méthodes utilisées lors de la manipulation des données
5. Le fichier **store.js** crée un nouvel objet pour stocker les données locales.
6. Le fichier **template.js** contient les fonctions qui génèrent le template HTML pour l'application.
7. Le fichier **view.js** définit l’objet View et les méthodes qui modifient à l’affichage.
8. Les fichiers restants

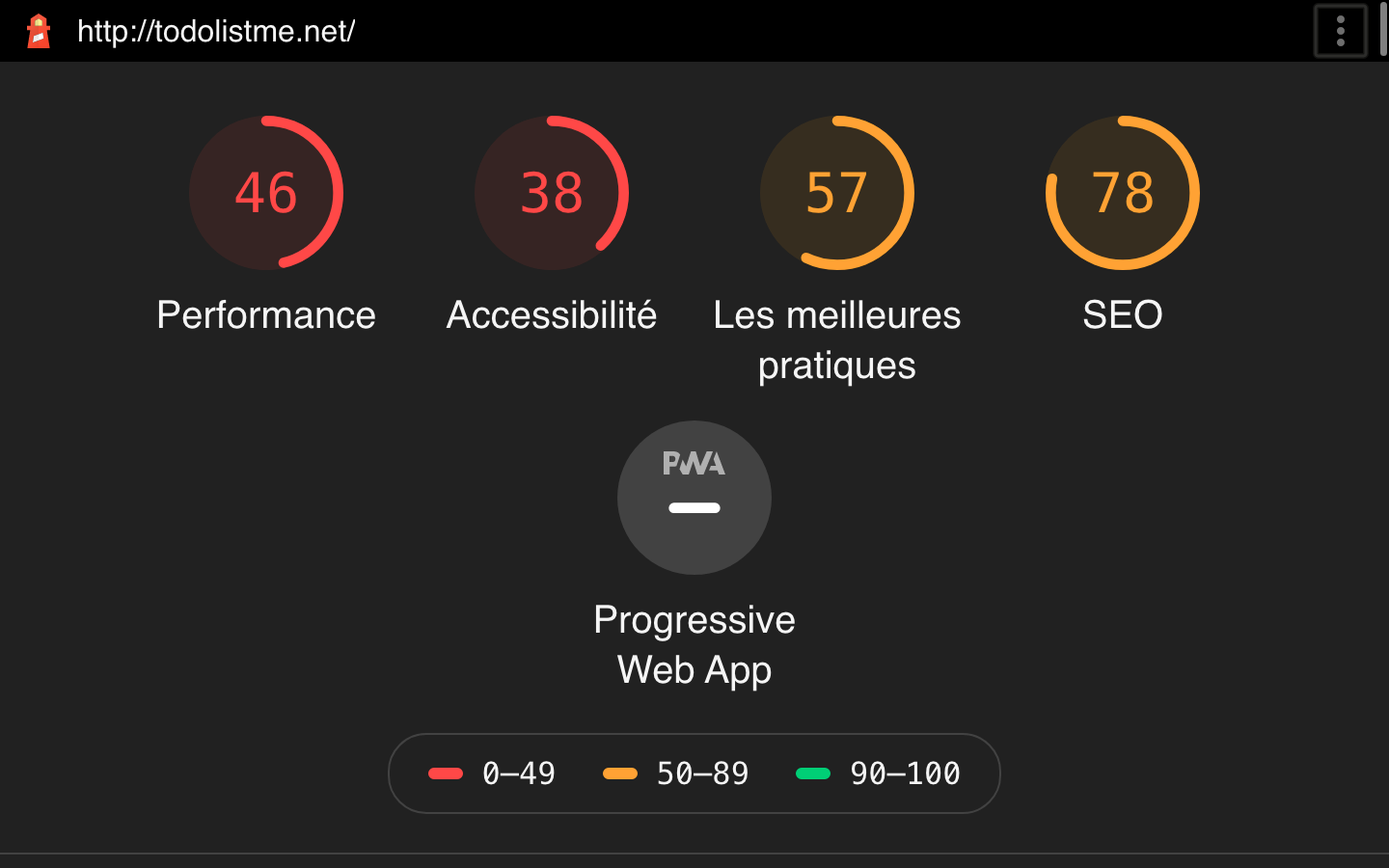
**ControllerSpec.js** Ce fichier contient tous les tests unitaires qui ont été́ écrit pour cette application à l'aide de Jasmine Framework.

**SpecRunner.html** est la page qui montre les résultats des tests automatisés qui ont été́ exécutés sur notre projet. Le contenu de la page en fonction des tests écrits dans le fichier ControllerSpec.js.

1. **AUDIT DE PERFORMANCE DU CONCURRENT**

**(Voir document « audit de performance site concurrent » complet)**

Audit de performance de l’application concurrente.



**Performance** : les chiffres reçus montrent que l’application n’a pas la moyenne en matière de performance.

* Points faibles : Elle est jugée trop lente. Les temps d’affichage sont trop long.
* Axe d’amélioration : Compression des images, compresser le code, supprimer le CSS.

**Accessibilité :**L’application pose problème au niveau de l’accessibilité.

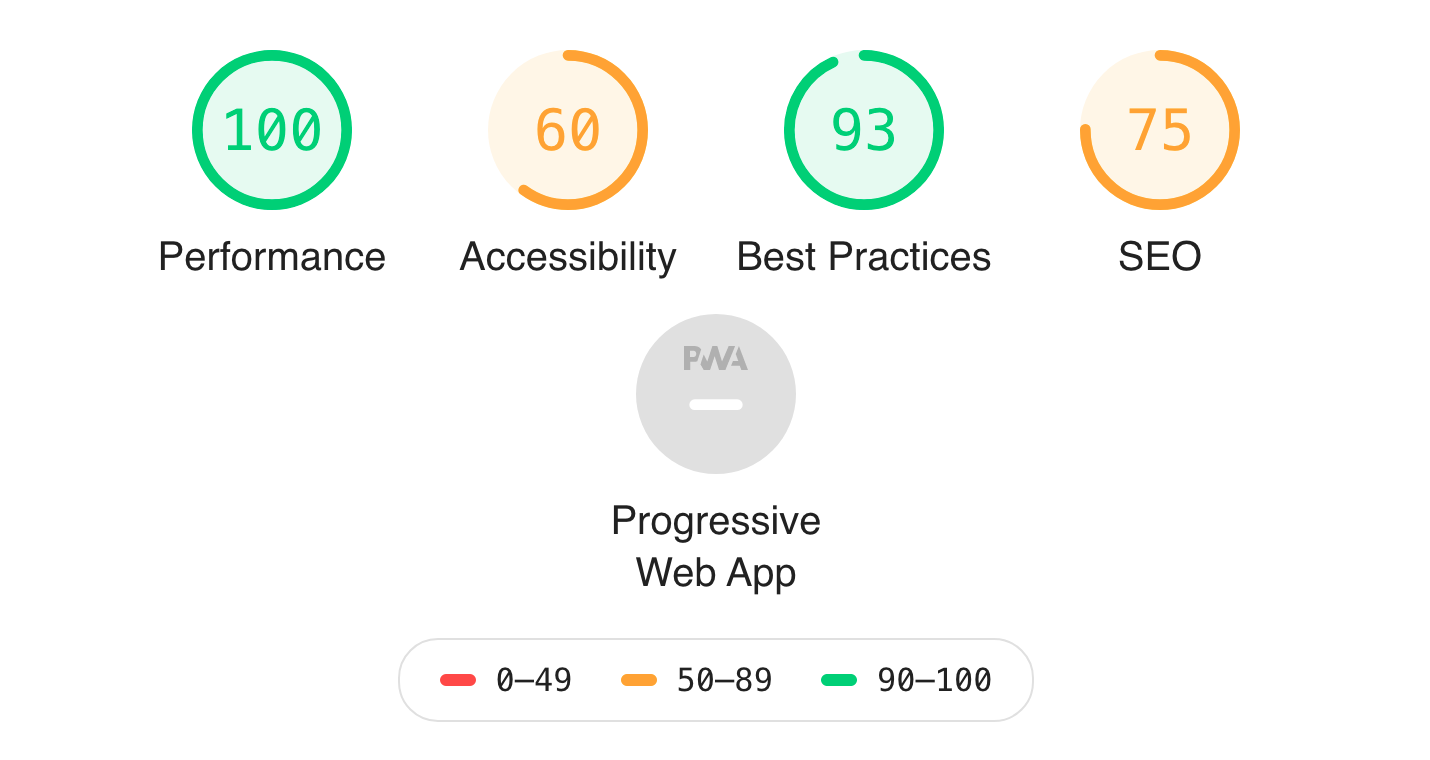
* Points faibles : La chartre graphique, problème pour les non-voyants, les images n’ont pas d’attribut.

**Les meilleures pratiques :**

* Points faibles : Une erreur dans la console, les liens ne sont pas sécurisés par la norme HTTPS.

**SEO :** L’application ne dispose pas de balise <meta name="viewport"> pour optimiser votre application pour les écrans mobiles.

1. **AUDIT DE PERFORMANCE DE TODOLIST**



**Conclusion**

Les outils devTools nous montre que les performances de l’application « **to-do list** » sont plus que correcte. L’application est rapide mais reste perfectible sur le référencement naturel et l’accessibilité.

L’audit de performance réaliser sur notre site concurrent nous permet d’analyser les faiblesses de l’application et en faire une force pour la nôtre.

Axes d’améliorations en vue d’un « **scaling** » de l’application :

**1. Améliorer le UX design et ajouter des fonctionnalités comme la catégorie de liste et la temporalité**

**2. En compressant les images pour réduire leur poids,**

**3. En minifiant les fichiers JavaScript grâce à Uglify, Minify ou Packer par exemple,**

**4. En nettoyant le CSS inutile présent dans le code.**

**5. Nous pourrions améliorer les contrastes et permettre un tri taches.**

**6. Travailler sur le référencement naturel**