

# PROJET 8



REPRENEZ ET AMELIOREZ UN PROJET  
EXISTANT

Nom du projet : To-do List

# SOMMAIRE

- I) Présentation du projet**
- II) Correction des bugs**
- III) Amélioration du code**
- IV) Test Jasmine**
- V) Usage non technique**
- VI) Documentation technique**
- VII) Audit de performance du concurrent**
- VIII) Audit de performance de TodoList**

## I) 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.

## II) CORRECTION DES BUGS

### Bug numéro 1 Faute de frappe



```
94  */
95  Controller.prototype.addItem = function (title) {
96      var self = this;
97
98      if (title.trim() === '') {
99          return;
100     }
101
102     self.model.create(title, function () {
103         self.view.render('clearNewTodo');
104         self._filter(true);
105     });
106 }
```

- 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 :

```
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 // If an ID was actually given, find the item and update each property
```

### Après :

```
80 // Generate an ID
81 // CORRIGER LE BUG
82 // Incrémenter le compteur.
83 var newId = this._compteur;
84 this._compteur++
85
86 // Si un ID a été donné, trouve l'élément et met à jour les propriétés
87 if (id) {
88     for (var i = 0; i < todos.length; i++) {
89         if (todos[i].id === id) {
90             for (var key in updateData) {
91                 todos[i][key] = updateData[key];
92             }
93             break;
94         }
95     }
96     localStorage[this._dbName] = JSON.stringify(data);
97     callback.call(this, todos);
98 } else {
99     /**
100     * Génère un identifiant unique
101     * @voir https://forum.alsacreations.com/topic-5-26755-1-Resolu-Comm
102     * @voir https://developer.mozilla.org/fr/docs/Web/JavaScript/Referenc
103     */
104     updateData.id = newId; // avec le compteur incrémentée
```

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 :**

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

**Après :**

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

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

### III) 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 :**

```
121 Controller.prototype.editItemSave = function (id, title) {
122     var self = this;
123
124     while (title[0] === " ") {
125         title = title.slice(1);
126     }
127
128     while (title[title.length-1] === " ") {
129         title = title.slice(0, -1);
130     }
131 }
```

**Après :**

```
125  Controller.prototype.editItemSave = function (id, title) {
126     var self = this;
127     // AMELIORATION
128     //La boucle while sur title[0] et title[title.length-1] permet de supprimer les espaces avant et après le titre
129     // la propriété trim permet de faire ca directement. Le code est plus performant.
130     // les deux boucles ne sont pas assez performante. Les blancs considérés sont les caractères d'espacement
131     title = title.trim();
132 }
```

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 :**

```
164
165 items.forEach(function(item) {
166     if (item.id === id) {
167         console.log("Element with ID: " + id + " has been removed.");
168     }
169 });
```

**Après :**

```
165 // AMELIORATION
166 // un console.log est utile uniquement en production et pas en phase d'exploitation
167
168 // items.forEach(function(item) {
169 //     if (item.id === id) {
170 //         console.log("Element with ID: " + id + " has been removed.");
171 //     }
172 // });
```

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** ligne 120. Les deux boucles **for** peuvent être améliorées.

### Avant :

```
120 Store.prototype.remove = function (id, callback) {
121     var data = JSON.parse(localStorage[this._dbName]);
122     var todos = data.todos;
123     var todoId;
124
125     for (var i = 0; i < todos.length; i++) {
126         if (todos[i].id == id) {
127             todoId = todos[i].id;
128         }
129     }
130
131     for (var i = 0; i < todos.length; i++) {
132         if (todos[i].id == todoId) {
133             todos.splice(i, 1);
134         }
135     }
```

### Après :

```
116 Store.prototype.remove = function (id, callback) {
117     var data = JSON.parse(localStorage[this._dbName]);
118     var todos = data.todos;
119     //var todoId;
120     // AMELIORATION
121     // la premiere boucle for servant simplement à attribuer à todoId
122     // la valeur de todos[i].id, celle-ci n'est pas nécessaire vu
123     // que la seconde boucle permet d'obtenir le meme resultat en utilisant directement l'id reçu.
124
125     for (var i = 0; i < todos.length; i++) {
126         if (todos[i].id == id) { // la meme chose pour deux boucles différentes
127             todos.splice(i, 1);
128         }
129     }
```

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.

## IV) 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 chaîne de caractère** et **une fonction**. La chaîne 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** :

```
61     it('should show entries on start-up', function () {
62         // TODO: write test
63         let todo = {title : 'my todo',};
64         setUpModel([todo]);
65         subject.setView('');
66         expect(view.render).toHaveBeenCalled('showEntries', [todo])
67     });
```

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

```
85     it('should show active entries', function () {
86         // TODO: write test
87         // montrer la todolist avec des taches à faire (statut active)
88         let todo = {title: 'my todo', completed: false};
89         setUpModel([todo]);
90         subject.setView('#/active');
91         expect(view.render).toHaveBeenCalled('showEntries', [todo]);
92     });
```

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

```
94     it('should show completed entries', function () {
95         // TODO: write tes
96         // montrer la todolist avec des taches complété (statut completed)
97         let todo = {title: 'my todo', completed: true};
98         setUpModel([todo]);
99         subject.setView('#/completed');
100        expect(view.render).toHaveBeenCalled('showEntries', [todo]);
101    });
```

- Teste la vue avec le filtre 'Tous' par défaut **Ligne 145** :

```
145    it('should highlight "All" filter by default', function () { //
146        // TODO: write test
147        // montrer la todolist avec toutes les taches peu importe leur statut (statut all)
148        subject.setView('');
149        expect(view.render).toHaveBeenCalled('setFilter', '')
150    });
```

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



```

151  it('should highlight "Active" filter when switching to active view', function () {
152      // TODO: write test
153      //test la view si "Active" est active quand on change pour l'onglet active
154      subject.setView('#/active');
155      expect(view.render).toHaveBeenCalled('setFilter', 'active')
156  });

```

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

```

166  describe('toggle all', function () {
167      it('should toggle all todos to completed', function () {
168          // TODO: write test
169          // basculer tous les todos vers terminés
170          let todo = [{id: 42, title: 'Premiere tâche', completed: false},
171                    {id: 43, title: 'Seconde tâche', completed: false}];
172          setUpModel(todo);
173
174          subject.setView('');
175
176          view.trigger('toggleAll', {completed: true})
177
178          expect(model.update).toHaveBeenCalled(42, {completed: true}, jasmine.any(Function));
179          expect(model.update).toHaveBeenCalled(43, {completed: true}, jasmine.any(Function));
180
181      });

```

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

```

183  it('should update the view', function () {
184      // TODO: write test
185      // mettre à jour la view
186      let todo = [{id: 42, title: 'Premiere tâche', completed: false},
187                {id: 43, title: 'Seconde tâche', completed: false}];
188      setUpModel(todo);
189
190      subject.setView('')
191      view.trigger('toggleAll', {completed: true})
192
193      expect(view.render).toHaveBeenCalled('elementComplete', {id: 42, completed: true});
194      expect(view.render).toHaveBeenCalled('elementComplete', {id: 43, completed: true});
195
196  });

```

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

```

199  describe('new todo', function () {
200      it('should add a new todo to the model', function () {
201          // TODO: write test
202          // ajouter une nouvelle tâche au modèle
203          setUpModel([]);
204          subject.setView('');
205          view.trigger('newTodo', 'a new todo');
206          expect(model.create).toHaveBeenCalled('a new todo', jasmine.any(Function));
207      });

```

- Teste la suppression d'une tâche dans le modèle **Ligne 244** :

```

244  describe('element removal', function () {
245      it('should remove an entry from the model', function () {
246          // TODO: write test
247          // supprimer une entrée du modèle
248          let todo = {id: 42, title: 'my todo', completed: true};
249          setUpModel([todo]);
250          subject.setView('');
251          view.trigger('itemRemove', {id: 42});
252
253          expect(model.remove).toHaveBeenCalled(42, jasmine.any(Function));
254      });

```

## Tests supplémentaires

- Teste l'onglet 'Complétés' quand on switch sur le filtre 'completed' **Ligne 157**

```

157  it('should highlight "Completed" filter when switching to completed view', function () {
158      //TEST RAJOUTEE
159      // Teste si le bouton 'Completed' est activé lorsque l'on passe en vue 'completed'
160      var todo = {id: 42, title: "my todo", completed: true};
161      setUpModel([todo]);
162      subject.setView("#/completed");
163      expect(view.render).toHaveBeenCalledWith("setFilter", "completed");
164  });

```

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

31 specs, 0 failures

finished in 0.907s

```
controller
  should show entries on start-up

routing
  should show all entries without a route
  should show all entries without "all" route
  should show active entries
  should show completed entries

  should show the content block when todos exists
  should hide the content block when no todos exists
  should check the toggle all button, if all todos are completed
  should set the "clear completed" button
  should highlight "All" filter by default
  should highlight "Active" filter when switching to active view
  should highlight "Completed" filter when switching to completed view

toggle all
  should toggle all todos to completed
  should update the view

new todo
  should add a new todo to the model
  should add a new todo to the view
  should clear the input field when a new todo is added

element removal
  should remove an entry from the model
  should remove an entry from the view
  should update the element count

remove completed
  should remove a completed entry from the model
  should remove a completed entry from the view

element complete toggle
  should update the model
  should update the view

edit item
  should switch to edit mode
  should leave edit mode on done
  should persist the changes on done
  should remove the element from the model when persisting an empty title
  should remove the element from the view when persisting an empty title
  should leave edit mode on cancel
  should not persist the changes on cancel
```

## V) DOCUMENTATION NON TECHNIQUE

Notre application se présente de la façon suivante.



### a) 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**

## b) 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.



## c) 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.



## d) Voir l'état des todos

Le bouton **All** nous montre l'ensemble des Todos quel que 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 terminés.

## VI) DOCUMENTATION TECHNIQUE

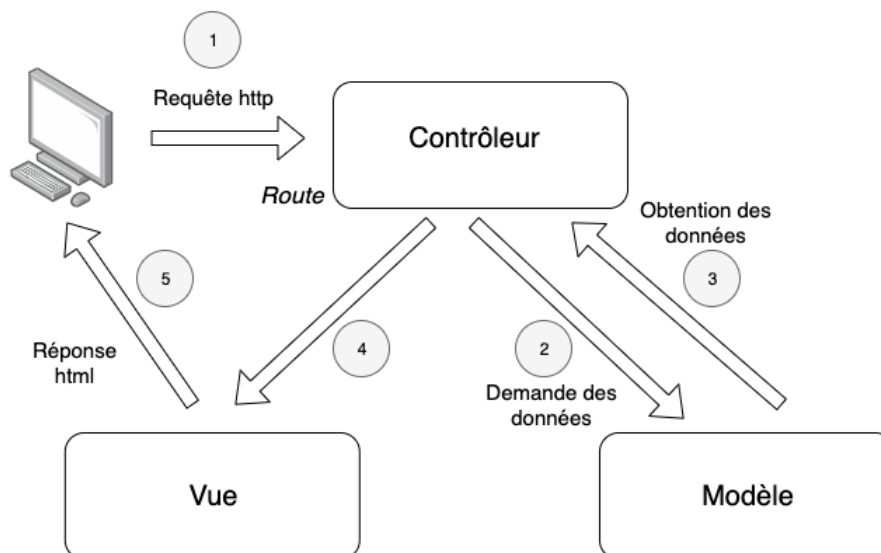
(Voir document « Documentation fonctionnelle et technique »)

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 :

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

3 directories, 5 files

Certains fichiers méritent des explications :

- Le fichier **license.md** contient les détails de la licence et de permission MIT
- Le fichier **package.json** est installé à la racine du projet en tant que **gestionnaire de paquet** 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.

## 2) 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
```

- a) Le fichier **app.js** est le point d'entrée de l'application et le moteur de l'application.
- b) Le fichier **controller.js** contient toutes les actions requises pour l'application. Il appelle les méthodes pour la modification des données.
- c) Le fichier **helpers.js** fournit des raccourcis utilisés dans les fichiers principaux pour alléger le code dans sa globalité.
- d) Le fichier **model.js** définit le Model et ses méthodes utilisées lors de la manipulation des données
- e) Le fichier **store.js** crée un nouvel objet pour stocker les données locales.
- f) Le fichier **template.js** contient les fonctions qui génèrent le template HTML pour l'application.
- g) Le fichier **view.js** définit l'objet View et les méthodes qui modifient à l'affichage.

## 3) Les fichiers restants

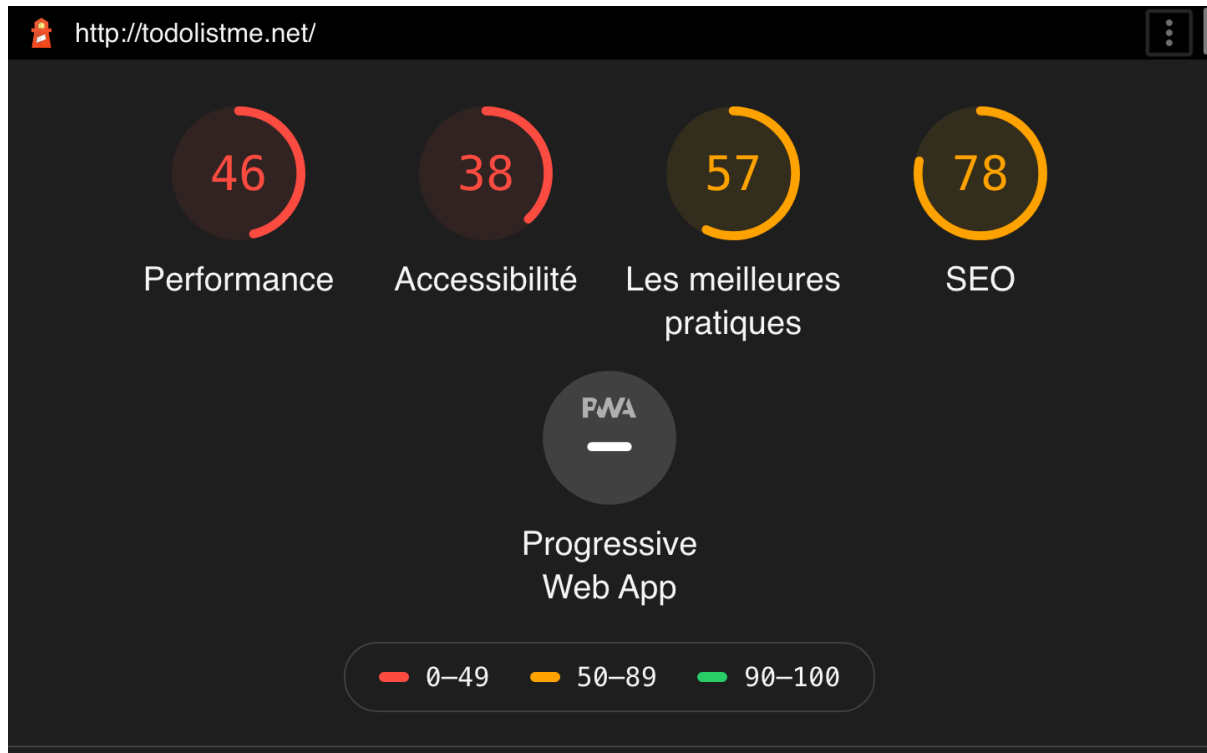
**ControllerSpec.js** Ce fichier contient tous les tests unitaires qui ont été écrit pour cette application à l'aide de Jasmine Framework.

**SpecRunner.html** est la 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.

## VII) 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.

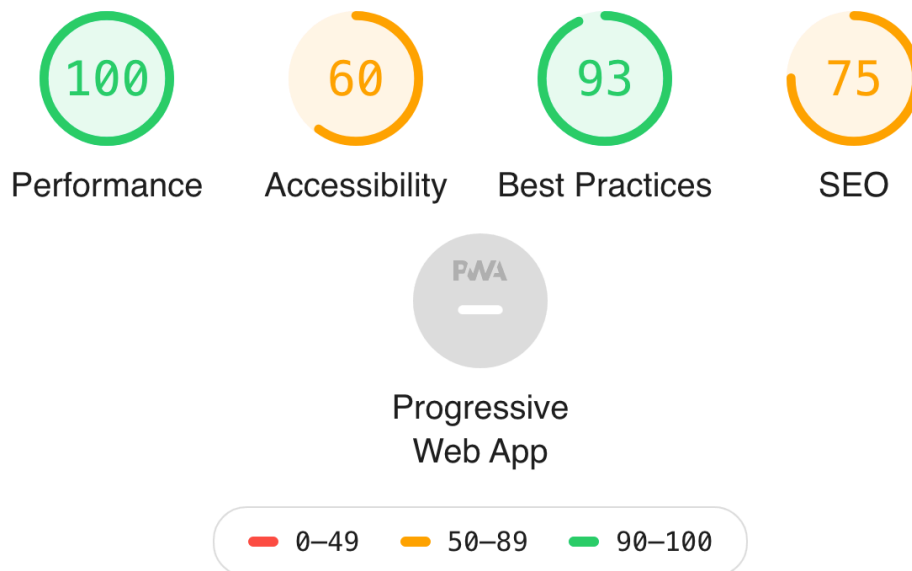
**Les meilleures pratiques** :

- 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.



## VIII) 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**