**Documentation fonctionnelle et technique**

1. **Présentation**

L’application **To-do list** a pour but de permettre facilement de créer une liste de tâches à effectuer par l’utilisateur.

Cette application permet **de créer, de modifier ou supprimer les tâches.**

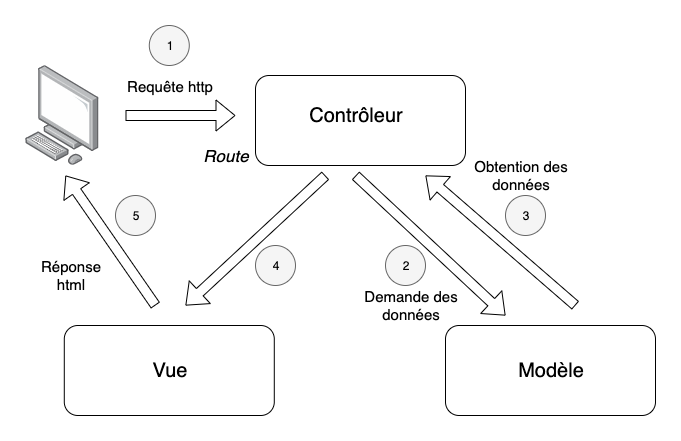
1. **Architecture**

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. **Structure du projet**
2. Arborescence du projet

Vous trouverez ci-dessous l’arborescence de l’application. Elle est composée de plusieurs fichiers nécessaire à son exécution :



Certains fichiers méritent notre attention comme :

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

1. Les fichiers Javascript

Le code de chaque fichier est inclus au sein d’une fonction IIFE (Expression de fonction invoquée 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.

* **Function Todo ligne 12** a le même comportement qu’une classe et charge tous les autres fichier Javascript : **storage », « model », « template », « view » et « controller ».**
* **Function setView() ligne 27** gère l’URL de la page.
* **$on(window, 'load', setView)** L’évènement load est émis lorsqu’une ressource et ses ressources dépendantes sont complètement chargées. **$on(window, 'hashchange', setView)** est déclenché lorsque le hash de l’url change.

1. Le fichier controller.js contient toutes les actions requises pour l'application. Il appelle les méthodes pour la modification des données.

Il joue l’intermédiaire entre le model (modèle) et la vue (view). Une fois la mise à jour effectuée, ces données sont envoyées à View qui va mettre à jour l'affichage et le DOM.

Function Controller (model, view) ligne 11 permet l’interaction entre les deux.

Les méthodes Controller.prototype… permettent la mise à jour des items.

1. Le fichier helpers.js fournit des raccourcies utilisés dans les fichiers principaux pour alléger le code dans sa globalité.

* **qs** Récupère les éléments grâce au sélecteur querySelector.
* **qsa** Récupère les éléments grâce au sélecteur querySelectorAll.
* **$on** Création d'un gestionnaire d'évènements avec la méthode -addEventListener.
* **$delegate** Délègue un gestionnaire d'évènements à un parent.
* **$parent** Récupère le parent d'un élément HTML.

1. Le fichier model.js définit le Model et ses méthodes utilisées lors de la manipulation des données

* **Function Model (storage)** Initialise le model et utilise le Stockage.
* **Model.prototype.create** créer une tâche todo
* **Model.prototype.read** Trouve et retourne un modèle en stockage.
* **Model.prototype.update** Met à jour une tâche.
* **Model.prototype.getCount** Compte toutes les taches.
* **Model.prototype.remove** supprime une tâche du stockage.

1. Le fichier store.js crée un nouvel objet pour stocker les données locales.

Function Store (name, callback) ligne 13

* **Store.prototype.find** Trouve des tâches).
* **Store.prototype.findAll** Retrouve toutes les données.
* **Store.prototype.remove** Supprime un item (tâche) du Store en fonction de son id.
* **Store.prototype.drop** Purge toutes les données et réinitialise le array à vide.

1. Le fichier Template Contient les fonctions qui génèrent le template HTML pour l'application.

Function Template() ligne 31 Initialise le template par défaut.

* **Template.prototype.show** Insérer en HTML les données stockées dans Data,
* **Template.prototype.itemCounter** compteur le nombre de tâches à terminer.
* **Template.prototype.clearCompletedButton** Affiche le bouton "clear completed" si plus de zéro tâches complétées.

1. Le fichier view définit l’objet View et les méthodes qui modifient à l’affichage.

Function View (template) ligne 15 Initialise l’affichage de l’application.

* **View.prototype.\_removeItem** Supprime la tâche en fonction de son id.
* **View.prototype.\_clearCompletedButton** Rend visible ou non le bouton clear completed
* **View.prototype.\_setFilter** Gère l'affichage de la page en fonction du choix de l'utilisateur (all, active, completed).
* **View.prototype.\_elementComplete**  Gère l'affichage des tâches marquées comme complétées.
* **View.prototype.\_editItem** Teste si un élément est en cours de modification et permet cette modification

1. Les fichiers restants
2. Test Jasmine

ControllerSpec.js contient les tests unitaires Jasmine.

SpecRunner.html est exécute et affiche les tests.