Fonctionnement technique

Voici le détail des fonctions utilisées dans ce projet :

1 - app.js

On a un objet **Todo** qui nous sert à configurer une nouvelle to do list en lançant chacun des autres fichiers avec en paramètre le nom que l'on veut lui donner.

La fonction **setView** nous sert à contrôler la vue (ce qui est affiché à l'utilisateur).

2 - controller.js

L'objet **Controller** qui prend l'objet **model** et l'objet **view** en paramètre va faire office d'intermédiaire en tant que contrôleur grâce à un certain nombre de méthodes :

- **addItem** qui sert à ajouter un objet todo avec un titre
- **editItem** pour pouvoir éditer le titre d'un todo grâce à son identifiant
- **editItemSave** qui sauvegarde l'édition du titre grâce à l'identifiant
- **editItemCancel** pour annuler les modifications du titre toujours avec l'identifiant
- **removeItem** qui nous permet d'effacer un todo grâce à l'identifiant
- **toggleComplete** qui fait passer le paramètre completed de l'état false à true

- **removeCompletedItems** permet d'effacer le(s) todo(s) dont le paramètre completed est égal à true

- **toggleAll** qui compte le nombre de todo(s) dont la case a été cochée pour pouvoir les ranger dans les bons filtres (Active et Completed) quand on va trier nos todos

Grâce à ces méthodes, nous allons pouvoir utiliser les fonctions suivantes :

- **setView** qui charge et initialise la vue selon le paramètre qui va nous rediriger vers les trois vues (All, Active et Completed)

- **showAll** va aller chercher l'ensemble des objets au chargement pour pouvoir les afficher
- **showActive** passe l'état des todos à 'Active'
- **showCompleted** passe l'état des todos à 'Completed'
- **removeCompletedItems** qui supprime les todos dont l'état completed est égal à true

- **_updateCount** va compter les todos 'Active' et les 'Completed' pour pouvoir mettre la page à jour

- **_filter** filtre à nouveau les todos en se basant sur l'onglet qui est actif

- **_updateFilterState** met à jour l'état du filtre selon l'onglet sélectionné

3 - helpers.js

Dans ce script, nous avons accès à des fonctions pour nous aider à faire certaines choses plus rapidement en reprenant l'élément **window** :

Récupérer les éléments grâce aux sélecteurs CSS :
- **window.qs**
- **window.qsa**

Ajouter un addEventListener qui regroupe plusieurs paramètres :
- **window.$on**

Attacher un gestionnaire à l'événement pour tous les éléments qui correspondent au sélecteur :
- **window.$delegate**

Rechercher le parent de l'élément avec le nom de balise donné :
- **window.$parent**

4 - model.js

Grâce à l'objet Model, on va pouvoir créer une nouvelle instance de model et le raccorder au stockage :

- **create** qui va nous permettre de faire un nouveau model pour les todos en créant un objet qui va pouvoir être sauvegardé

- **read** retrouve et retourne un model dans le stockage, si rien ne correspond alors il retourne tout les éléments, on peut rechercher grâce à l'identifiant, une chaîne de caractères ou encore un objet

- **update** va mettre le model à jour grâce à l'identifiant, les données et la fonction de callback une fois que tout est prêt

- **remove** nous permet de supprimer le model que l'on souhaite grâce à l'identifiant et en lançant une fonction de rappel pour la mise à jour

- **removeAll** supprime l'ensemble des données et effectue l'actualisation grâce à une fonction de callback

- **getCount** range et nous donne l'ensemble des todos par 'Active', 'Completed' et le total

5 - store.js

L'objet **Store** va mettre en place le stockage côté client et il créera une collection vide si il n'y a pas de collection existante grâce à un nom et à un callback pour mettre les données à jour car on n'utilise pas d'appels AJAX dans ce projet

- **find** nous permet de trouver une donnée grâce à un objet
- **findAll** retourne l'ensemble de la collection

- **save** sauvegarde la donnée que l'on souhaite dans la base de données, si la donnée n'existe pas, elle est créée sinon elle est mise à jour

- **remove** efface une donnée de la base grâce à son identifiant
- **drop** réinitialise la base en effaçant toutes les données

6 - template.js

L'objet **Template** construit un design pour tout les todos

- **show** crée un élément <li> qui contient nos données todo dans l'HTML pour le placer dans l'application

- **itemCounter** Nous affiche un compteur indiquant le nombre de todos qu'il reste à réaliser

- **clearCompletedButton** met à jour le bouton 'Clear completed' en fonction des todos qu'il reste à faire (le bouton apparaît si un todo est 'Completed' sinon il disparaît)

7 - view.js

L'objet **View** va faire abstraction du DOM, il a deux points d'entrées :

- **bind**, qui va lier l'événement avec le gestionnaire qui correspond
- **render**, va nous permettre d'exécuter les commandes comme effacer ou éditer un todo

à cela, on ajoute des sélecteurs qui vont être utilisés dans les fonctions suivantes :

- **_removeItem** qui efface une donnée par rapport à son identifiant
- **_clearCompletedButton**, affiche ou non le bouton 'Clear Completed'
- **_setFilter** qui trie les éléments par rapport à leurs classes
- **_elementComplete** va nous dire il il y a au moins un élément de la liste qui est 'Completed'
- **_editItem** pour pouvoir éditer notre todo en fonction de son identifiant
- **_editItemDone** qui va enregistrer les modifications et mettre l'élément concerné à jour

Ensuite nous allons voir des fonctions qui seront utilisées dans le bind :

- **_itemId** permet de retrouver un identifiant dans un
- **_bindItemEditDone** va lier un élément qui a été modifié
- **_bindItemEditCancel** annule la liaison d'un élément qui vient d'être modifié