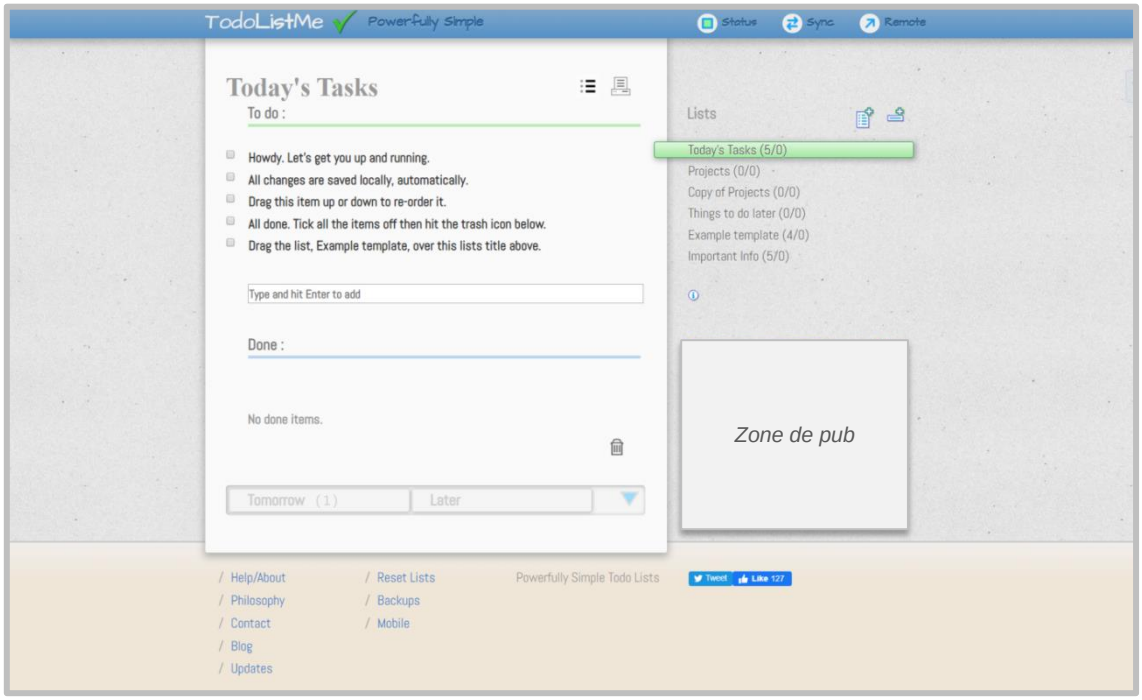
Audit de performance

1. L'application TodoListMe

Interface utilisateur



On observe une interface graphique plus développée et bien plus chargée. L'ensemble des éléments présents rendent la page moins intuitive aux premiers abords pour l'utilisateur. Cela s'explique par la présence d'un nombre de fonctionnalités plus élevé.

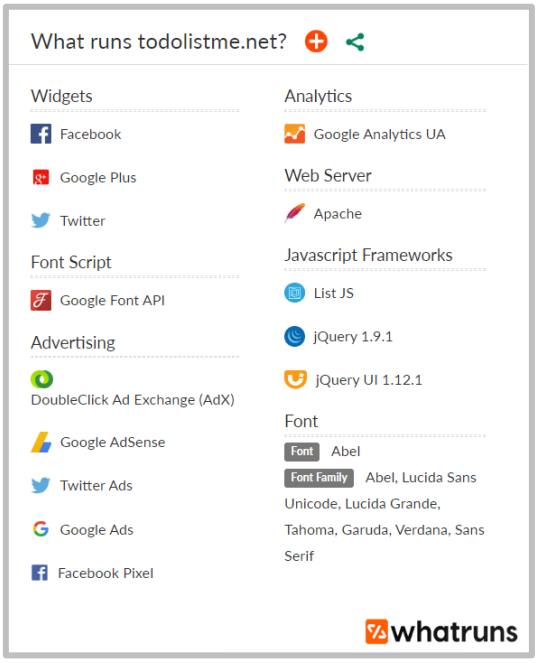
Fonctionnalités

TodoListMe présente des fonctionnalités de base d'un site de gestion des tâches que l'on retrouve également dans notre application. Le site propose également des fonctionnalités plus poussées avec un système de gestion plus avancé :

- Gestion des listes :
  - Création de plusieurs listes de tâches
  - Création de catégories pour stocker des listes
- Tri des tâches et ajout d'une date :
  - Possibilité de donner une temporalité à une tâche
  - Possibilité de classer les tâches selon différents critères (alphabétiques, Date, Aléatoire)
  - Système de drag and drop pour déplacer, planifier ou compléter une tâche
- Expérience utilisateur :
  - Barre de progression pour visualiser le nombre de tâches restantes et terminées
  - Version imprimable dans une nouvelle fenêtre avec une interface adaptée à l'impression
- Synchronisation des données :
  - Création d'un compte utilisateur (permet d'envoyer les listes sur un serveur et de les récupérer depuis un autre périphérique, avec un historique sur les listes de deux semaines)
- Réseaux sociaux et publicité :
  - Partage Twitter d'un lien vers le site
  - Bouton « like » de Facebook de la page de l'application
  - Publicité

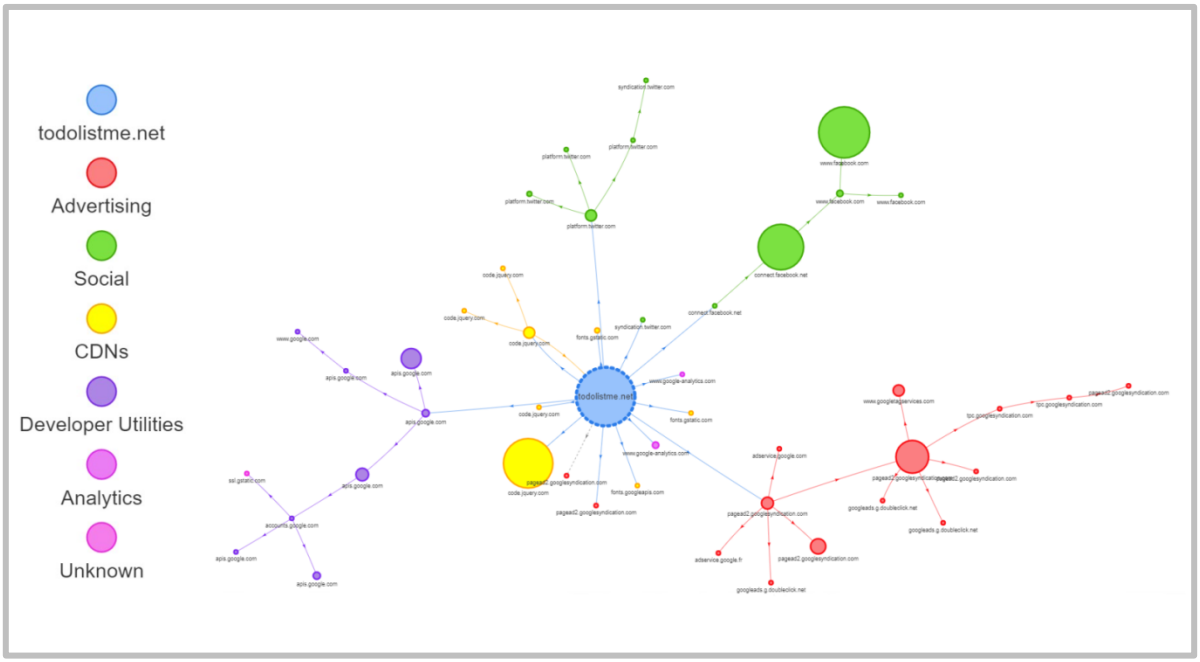
Technologies externes

L'utilisation de WhatsRun, nous a permis de pouvoir analyser les technologies utilisées par le site ainsi que les éléments présents.



On observe ainsi très rapidement qu'il y a de nombreux éléments qui interviennent pour tout ce qui concerne la publicité et on note également que la version utilisée de jQuery est 1.9.1 (dernière version stable → 3.5.0).

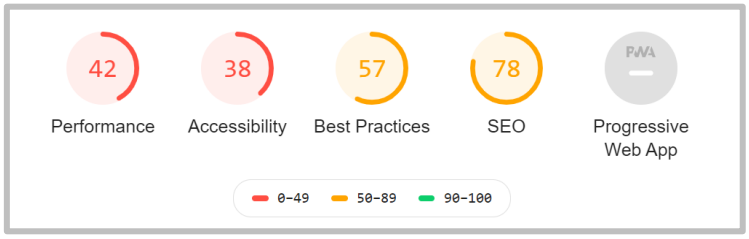
La **request map** permet une visualisation interactive des données. Elle montre l'ensemble des interactions avec les domaines tiers auxquels l'URL fait des demandes. Les gros nœuds signifient beaucoup d'octets, les nœuds distants signifient un TTFB (= Time to first byte) élevé (réactivité du serveur) et les lignes épaisses signifient beaucoup de demandes.



On observe bien les interactions. On remarque que les plus importantes sont la publicité (rouge) et les réseaux sociaux (vert).

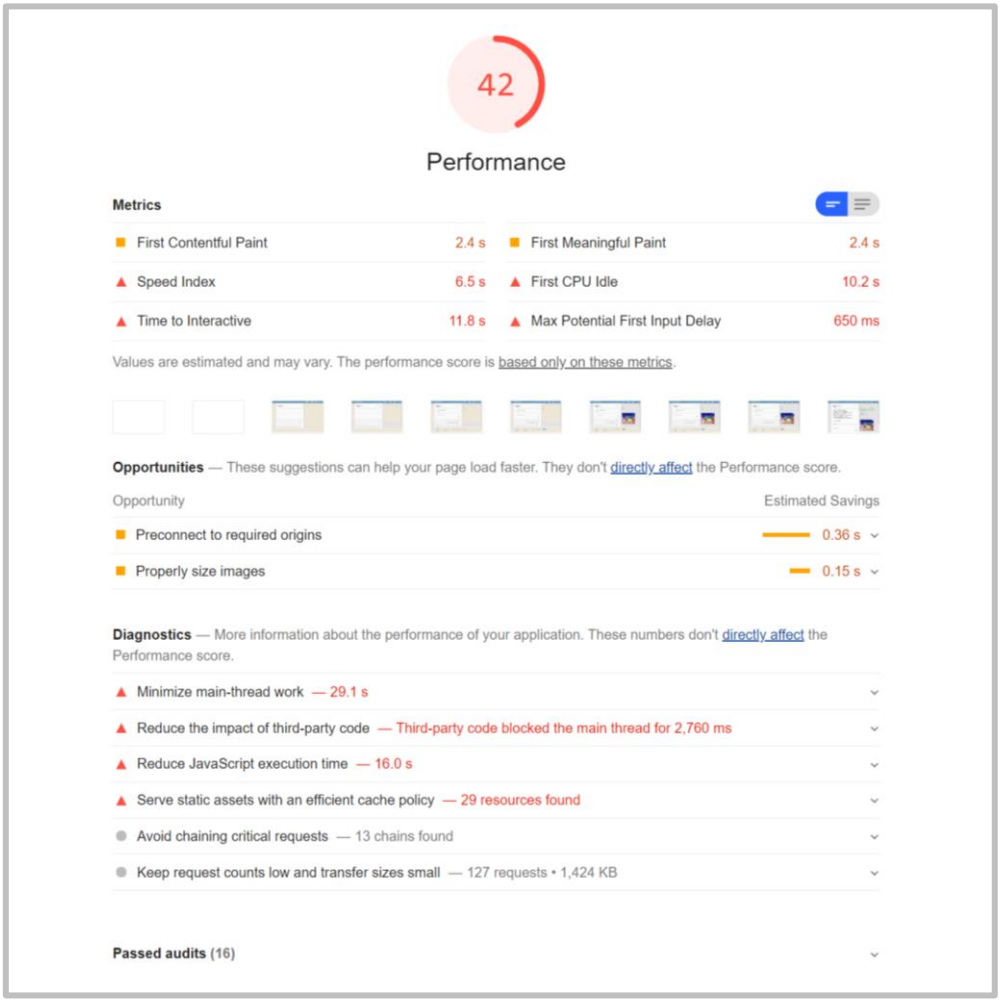
2. Audit du site

Pour réaliser des audits et présenter les informations, plusieurs outils en lignes sont disponibles. Ici, l'audit a été réalisé à l'aide de la console et de l'inspecteur du navigateur Chrome (Version 81.0, 64 bits) et de Lighthouse sur un ordinateur Windows 10.



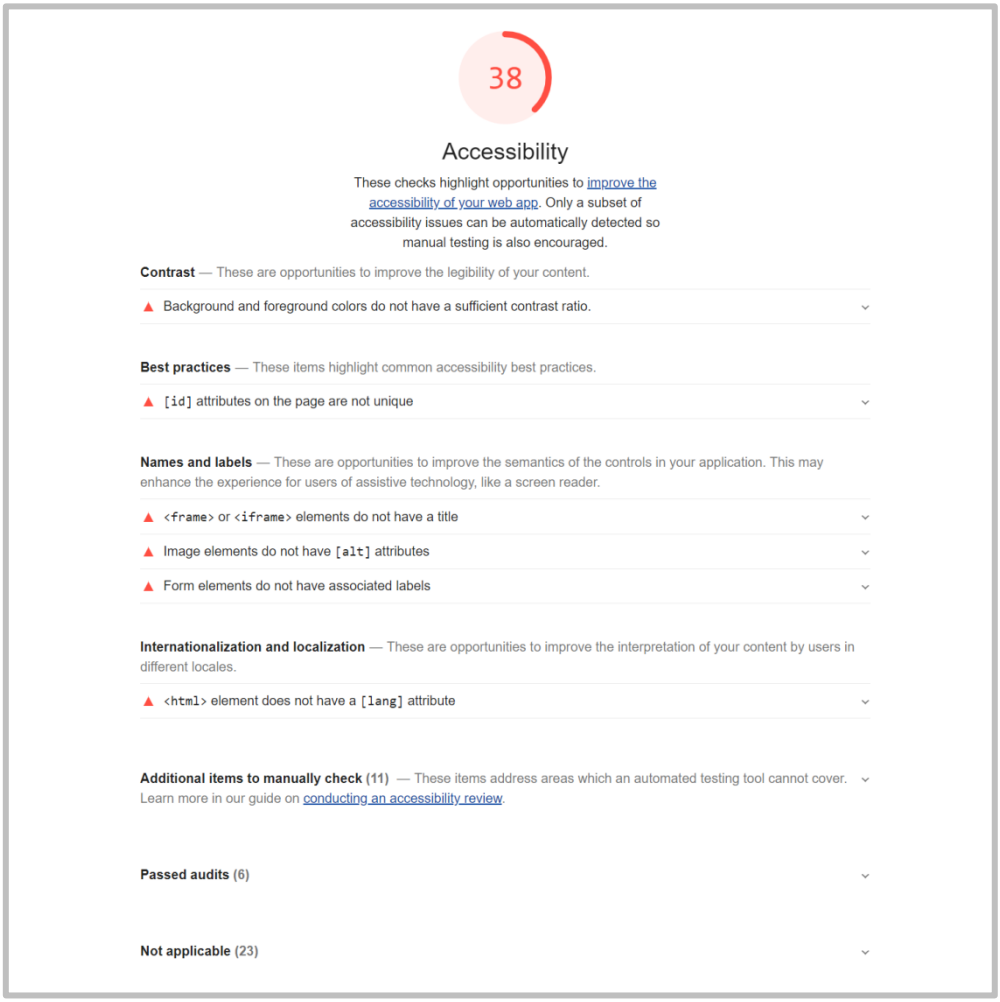
Le score de performance globale est de 42% et celui de l'accessibilité de 38%. Les meilleures pratiques et le SEO ont respectivement un score de 57% et 78%.

Performances



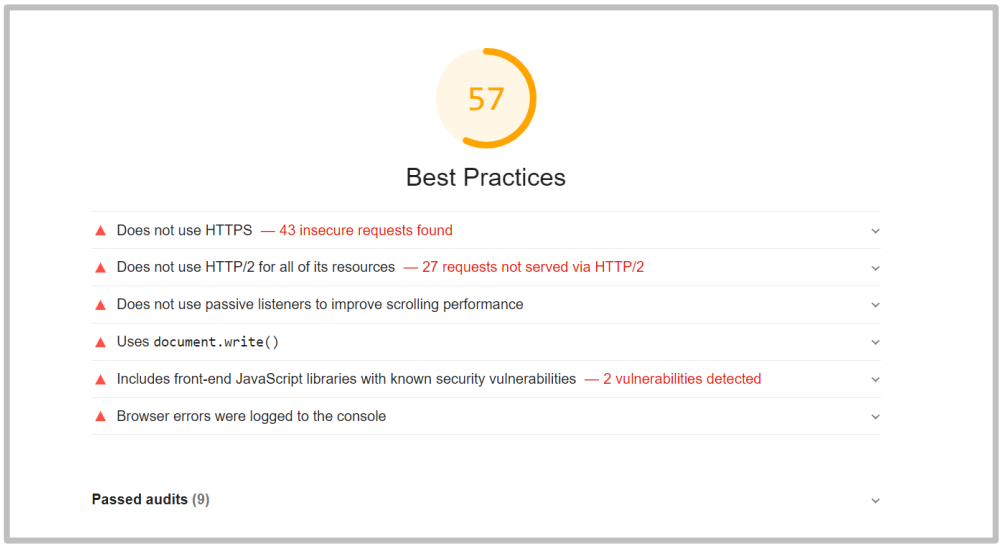
On observe que le **Time to Interactive** est de 11.8s, un **Speed Index** de 6.5s et un **First Contentful Paint** de 2.4s. On constate beaucoup d'appels externes (réseaux sociaux, Google, …) et des fichiers trop lourds (images, Jquery, …), on obtient donc un temps de chargement très long.

Accessibilité



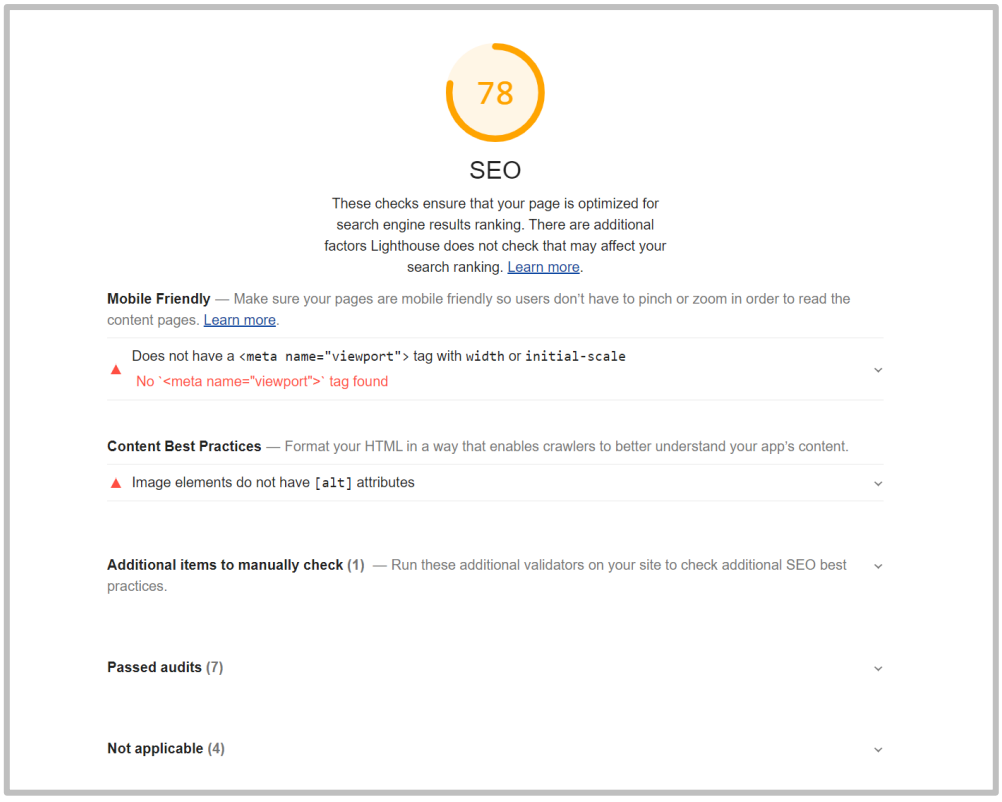
L'accessibilité est la catégorie la moins bien notée. Ainsi le **ratio de contraste** entre les couleurs en premier et second plan n'est pas assez élevé ce qui peut compliquer la navigation de certain utilisateurs. On note également que certains « **id** » ne sont pas uniques ce qui est problématique et peut générer des dysfonctionnements au niveau du code. Il n'y a pas d'**attribut « lang »** au tag html ce qui permet de référencer la langue principale du site. Et certaines images n'ont pas d'**attribut « alt »** et les inputs de formulaires n'ont pas de **labels associés**.

Bonnes pratiques



Là encore plusieurs problèmes sont remontés. Le site n'est pas sécurisé car il **n'utilise pas HTTPS**, il s'agit d'une faille de sécurité importante. On note également que la version de **Jquery** utilisée est une ancienne version contient des failles de sécurités connues et elle n'est pas minifiée. Il y a également des **erreurs dans la console**.

SEO (Search Engine Optimization)



Ici, on constate un problème concernant le côté mobile de l'application. En effet, le **« viewport » metatag** n'est pas défini, ainsi les utilisateurs sur mobile auront une page adaptée pour desktop. Et certaines images n'ont pas d'**attributs « alt »**.

3. Résumé

Voici un rappel des éléments observés lors de cet audit avec tout d'abord les avantages et inconvénients de notre concurrent :

Avantages	Inconvénients
Nombreuses fonctionnalités	Temps de chargement
Possibilité de stocker les tâches sur un serveur	Beaucoup de scripts tiers chargés (publicité, réseaux sociaux, …)
	Site ni responsive, ni accessible

L'audit réalisé a permis d'identifier des points de comparaison sur les choses à faire et à ne pas faire concernant notre application.

A faire :

- Retravailler les fonctionnalités et l'agencement
- Rendre le site responsive et améliorer l'utilisation mobile
- Améliorer l'accessibilité (contraste, polices d'écriture, …)
- Mettre à jour régulièrement et minifier Jquery
- Optimiser le code JavaScript

A ne pas faire :

- Laisser des failles de sécurité (HTTPS, erreurs console, …)
- Maintenir les calls externes non essentiels (polices d'écriture, publicités, réseaux sociaux, librairies)
- Ne pas retravailler et optimiser les images (taille, format, …)

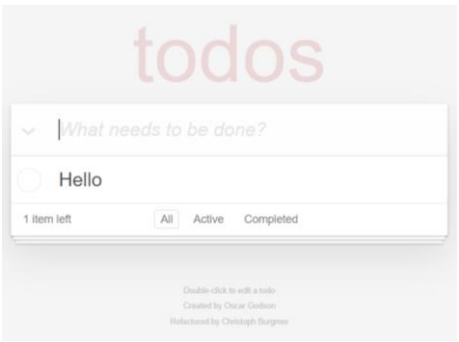
Guide utilisateur

L'application **todos** permet de créer et de gérer une liste de tâches à l'aide de fonctionnalités simples. Les tâches sont stockées dans le *localStorage*, il suffit d'utiliser toujours le même navigateur pour les retrouver. L'utilisateur peut :

- Ajouter une tâche
- Modifier une tâche
- Supprimer une tâche
- Terminer une ou plusieurs tâche(s)
- Supprimer toutes les tâches terminées
- Filtrer les tâches

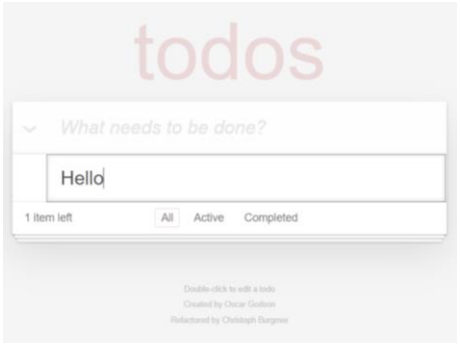
1. Ajouter une tâche

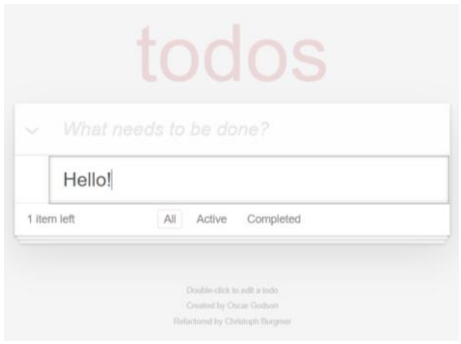
Pour ajouter une tâche, il suffit de cliquer sur le champ **What needs to be done?**, d'écrire votre tâche et d'appuyer sur la touche *Entrée* du clavier ou de cliquer en dehors du champ de saisie. La tâche créée s'ajoute à la liste.

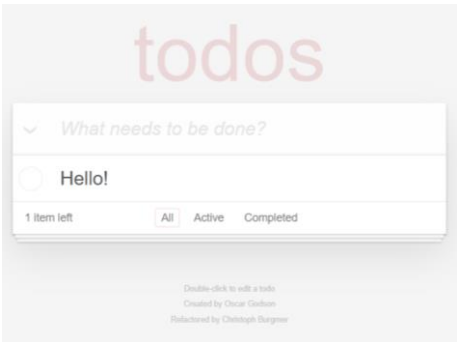


2. Modifier une tâche

Pour modifier une tâche (ou l'éditer), *double-cliquez* sur la tâche pour afficher le champ d'édition. Une fois la modification effectuée, il suffit d'appuyer sur la touche *Entrée* du clavier ou de cliquer en dehors du champ de saisie pour qu'elle soit prise en compte. Si le champ n'est plus renseigné, la tâche sera supprimée.

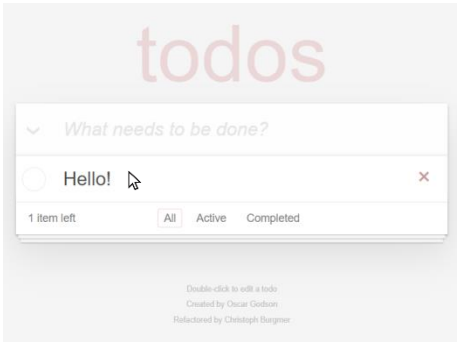


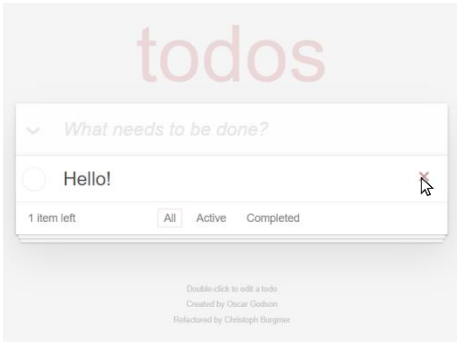


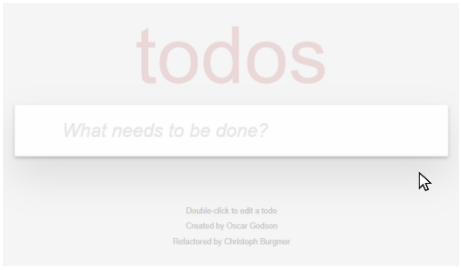


3. Supprimer une tâche

Pour supprimer une tâche, il suffit de passer sur la tâche à supprimer et de cliquer sur la croix qui apparaît sur la droite, en face de la tâche. Attention, il n'est pas possible de revenir en arrière, en effet, toute suppression est irréversible.







4. Terminer une ou plusieurs tâche(s)

Pour terminer une tâche (ou la compléter), il suffit de cliquer dans le cercle de la tâche terminée qui se situe à gauche de chaque tâche.

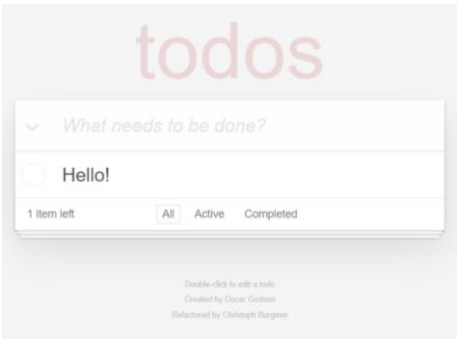
Il est également possible de marquer les tâches comme terminées toutes à la fois. Pour cela, il suffit de cliquer sur la flèche vers le bas qui se situe en haut à gauche du champ.

Une tâche terminée apparaît différemment : le texte à une couleur grise plus claire, il est barré et le cercle à sa gauche est coché.

Il est également possible de rétablir les tâches comme actives en cliquant de nouveau soit sur le cercle soit sur la flèche vers le bas.

5. Supprimer toutes les tâches terminées

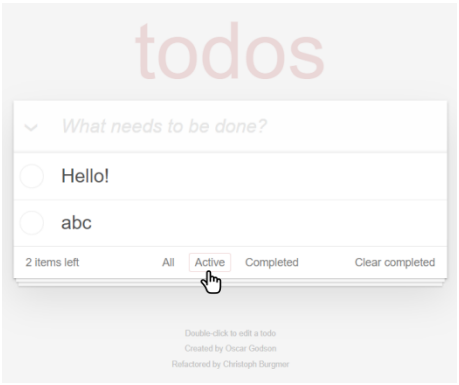
Toutes les tâches ayant le statut terminé (« completed ») peuvent être supprimées d'un seul coup. Il suffit de cliquer sur le bouton **Clear completed** qui apparaît en bas à droite dès qu'une tâche au moins est terminée.

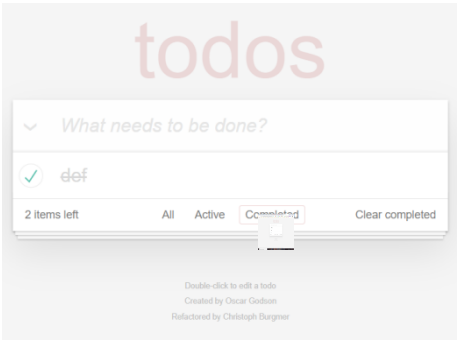


6. Filtrer les tâches

L'application propose trois boutons sous la liste de tâches. Il s'agit de trois filtres qui permettent d'afficher les tâches selon leur état :

- Le filtre **All** permet d'afficher toutes les tâches quelque soit leur état de réalisation.
- Le filtre **Active** permet de n'afficher que les tâches actives, c'est-à-dire non terminées.
- Le filtre **Completed** permet de n'afficher que les tâches terminées.





Documentation technique

En ce qui concerne le côté utilisation et interface de l'application, veuillez-vous reporter au guide utilisateur.

1. Technologie

Le site est développé en **HTML5** pour le fichier html, en **CSS3** pour les fichiers de style et en **JavaScript** (ECMAScript 5) pour les fichiers fonctionnels.

Le Framework **Jasmine** est utilisé pour la réalisation des tests unitaires de l'application.

2. Arborescence

L'arborescence ci-dessous présente les principaux dossiers et fichiers du projet :

```
p8
|+--index.html
|+--package.json
|
|____+--js
|        +--app.js
|        +--controller.js
|        +--helpers.js
|        +--model.js
|        +--store.js
|        +--template.js
|        +--view.js
|
|____+--node_modules
|        |____+--jasmine-core
|        |____+--todomvc-app-css
|        |____+--todomvc-common
|
|____+--test
|        +--ControllerSpec.js
|        +--SpecRunner.html
```

- Le fichier **`index.html`** lance l'application dans le navigateur.
- Le fichier **`package.json`** liste les dépendances nécessaires à installer avec npm.
- Le dossier **`js`** contient tous les fichiers JavaScript nécessaire pour le fonctionnement de l'application.
- Le dossier **`node_modules`** contient toutes les packages node / dépendances nécessaires au projet : les feuilles de style CSS de l'application et le Framework Jasmine pour la réalisation des tests unitaires.
- Le dossier **`test`** contient les tests unitaires qui utilisent Jasmine : un fichier JavaScript et une page html pour visualiser les résultats de tests.

3. MVC

L'architecture de l'application est basée sur une architecture Modèle-vue-contrôleur (ou MVC).

L'objectif du MVC est de séparer la logique du code en trois parties distinctes :

- **Le modèle** : élément qui contient les données et la logique en rapport avec les données (lecture, enregistrement, validation).
- **La vue** : partie visible de l'application, elle contient des éléments visuels ainsi que la logique nécessaire pour afficher les données issues du modèle.

- **Le contrôleur** : élément qui fait le lien entre le modèle et la vue, il s'occupe des changements suite aux actions de l'utilisateur et modifie les données du modèle et de la vue.

4. Les différents fichiers

helpers.js

Ce fichier permet de simplifier la syntaxe de JavaScript sans avoir à utiliser une bibliothèque lourde comme jQuery. Il crée des fonctions réutilisables et fournit ainsi des raccourcis utilisés dans les principaux fichiers. Cela permet également de garder un code clair.

- **`qs`** : simplifie la méthode `querySelector`, récupère le premier élément HTML d'après un argument sélecteur (`selector`) à l'intérieur d'un autre élément (`scope`).
- **`qsa`** : simplifie la méthode `querySelectorAll`, récupère tous les éléments HTML d'après un argument sélecteur (`selector`) à l'intérieur d'un autre élément (`scope`).
- **`$on`** : créé un écouteur d'évènement à l'aide de la méthode `addEventListener` sur l'élément choisi.
- **`$delegate`** : créé un gestionnaire d'évènements sur un élément existant ou futur.
- **`$parent`** : récupère le parent d'un élément HTML en précisant la balise recherchée (`tagName`).

store.js

Ce fichier permet la création de la base de données de l'application. Un nouvel objet base de données (store) sera stocké dans le `LocalStorage` du navigateur. Ainsi les données de la *todolist* seront stockées de cette manière. Ces méthodes ne sont appelées que dans le fichier model.js.

- **`Store`** : créé une nouvelle base de données dans le `LocalStorage` du navigateur si elle n'existe pas déjà. La base de données est un tableau d'objets qui représentent les différentes entrées (tâches) de la *todolist*.
- **`Store.prototype.find`** : récupère une donnée dans la base de données à partir d'une requête (`query`).
- **`Store.prototype.findAll`** : récupère toutes les données de la base de données.
- **`Store.prototype.save`** : enregistre (si elle n'existe pas) ou met à jour (si elle existe) une entrée dans la base de données (avec un ID unique).
- **`Store.prototype.remove`** : supprime une entrée de la base de données (d'après sont ID).
- **`Store.prototype.drop`** : efface entièrement la base de données.

app.js

Ce fichier sert de point d'entrée de l'application. Il permet de lancer l'application, d'initialiser la base de données (store), le modèle (model), le template (template), la vue (view) et le contrôleur (controller).

- **`Todo`** : initialise l'ensemble de l'application.
- **`setView`** : gère le chargement de la vue via le contrôleur en fonction de l'url. La fonction est appelée au chargement de la page et lors de changement d'url (#/, #/active, #/completed).

controller.js

Ce fichier permet de faire le lien entre la base de données (Model) et l'interface du site (view) selon les actions de l'utilisateur.

- **`Controller`** : créé un lien entre le modèle et la vue.
- **`Controller.prototype.setView`** : initialise la vue en fonction de l'url demandée.
- **`Controller.prototype.showAll`** : affiche toutes les entrées de la base.
- **`Controller.prototype.showActive`** : affiche toutes les entrées actives de la base.
- **`Controller.prototype.showCompleted`** : affiche toutes les entrées terminées de la base.
- **`Controller.prototype.addItem`** : ajoute une nouvelle entrée dans la base.
- **`Controller.prototype.editItem`** : déclenche le mode édition d'une entrée.

- **`Controller.prototype.editItemSave`** : enregistre la modification d'une entrée.
- **`Controller.prototype.editItemCancel`** : annule la modification d'une entrée.
- **`Controller.prototype.removeItem`** : supprime une entrée de la base.
- **`Controller.prototype.removeCompletedItems`** : supprime toutes les entrées terminées.
- **`Controller.prototype.toggleComplete`** : termine ou active une entrée dans la base (coche/décoche).
- **`Controller.prototype.toggleAll`** : termine ou active toutes les entrées dans la base (coche/décoche).
- **`Controller.prototype._updateCount`** : met à jour le nombre d'entrées actives restantes.
- **`Controller.prototype._filter`** : filtre les entrées en fonction de leur statut (All, Active ou Completed).
- **`Controller.prototype._updateFilterState`** : met à jour le statut sélectionné de la navigation du filtre

model.js

Ce fichier sert à gérer les données. Il s'occupe de récupérer et de gérer les informations dans la base de données. Il créé un nouveau modèle de base de données en se connectant au `LocalStorage` du navigateur (via store). Il utilisera les méthodes du fichier `Store.js` pour effectuer les opérations sur la base de données, qui seront appelées via le contrôleur.

- **`Model`** : créé un nouveau modèle et lui associe une base de données (`LocalStorage`).
- **`Model.prototype.create`** : créé un nouveau modèle pour la todo.
- **`Model.prototype.read`** : récupère dans la base un modèle todo selon une requête (ID ou nom). Si l'argument requête n'est pas renseigné, cela retourne toutes les entrées de la base.
- **`Model.prototype.update`** : met à jour une entrée en la sauvegardant dans la base de données (`LocalStorage`) selon son ID.
- **`Model.prototype.remove`** : supprime une entrée de la base de données (`LocalStorage`) selon son ID.
- **`Model.prototype.removeAll`** : supprime toutes les données de la base de données (`LocalStorage`).
- **`Model.prototype.getCount`** : retourne le compte de toutes les entrées.

view.js

Ce fichier gère l'affichage et la manipulation du DOM. Les changements d'états du template HTML s'effectuent ici. Ce fichier permet de mettre à jour le visuel de l'application. La fonction view permet deux types d'actions : attacher un évènement d'une tâche donnée avec un gestionnaire d'évènement (bind) et faire le rendu d'une commande avec ses options (render).

- **`View`** : définit le template à utiliser et prépare les éléments à cibler via les class CSS.
- **`View.prototype._removeItem`** : supprime l'élément contenant une entrée selon son ID.
- **`View.prototype._clearCompletedButton`** : affiche ou cache le bouton « Clear completed ».
- **`View.prototype._setFilter`** : active le bouton de filtre sélectionné (All, Active Completed).
- **`View.prototype._elementComplete`** : barre une entrée si elle a la classe « .completed », et enlève la barre sinon.
- **`View.prototype._editItem`** : gère l'affichage d'une entrée existante lors de la modification de son titre.
- **`View.prototype._editItemDone`** : retourne à l'affichage initial d'une entrée qui vient d'être modifiée.
- **`View.prototype.render`** : lance une des fonctions d'affichage suivantes selon la commande en paramètre.
    - **`showEntries`** : affiche les entrées.
    - **`removeItem`** : supprime une entrée.
    - **`updateElementCount`** : actualise le nombre d'entrées.

    - **clearCompletedButton** : actualise l'affichage du bouton « Clear completed ».
    - **contentBlockVisibility** : affiche ou cache le footer de la todolist selon la présence d'entrées ou non dans la liste.
    - **toggleAll** : change le statut de toutes les entrées en terminé (« completed »).
    - **setFilter** : gère l'affichage selon les filtres.
    - **clearNewTodo** : vide le champ de texte pour ajouter une nouvelle entrée.
    - **elementComplete** : gère l'affichage d'une entrée terminée (« completed »).
    - **editItem** : gère l'affichage d'une entrée en cours d'édition.
    - **editItemDone** : gère l'affichage d'une entrée après modification.
- **View.prototype._itemId** : rècupère l'ID d'une entrée.
- **View.prototype._bindItemEditDone** : gère l'affichage lors de la perte de focus d'une entrée en cours d'édition.
- **View.prototype._bindItemEditCancel** : gère l'affichage d'une entrée dont la modification est annulée.
- **View.prototype.bind** : gestionnaire d'évènements qui permet d'effectuer un rendu particulier en fonction des actions réalisées par l'utilisateur (évènement rentré en paramètre).

template.js

Ce fichier permet de fournir une trame HTML pour chaque nouvelle entrée ajoutée. Le template est gérée et actualisée par le fichier view.js.

- **Template** : génère un template HTML par défaut.
- **Template.prototype.show** : récupère le template par défaut et y injecte les informations de la nouvelle entrée.
- **Template.prototype.itemCounter** : affiche le nombre d'entrées actives restantes.
- **Template.prototype.clearCompletedButton** : affiche le bouton « Clear completed » dès qu'une entrée est terminée.

5. Fonctionnement général

Comme vu précédemment, le fichier helpers.js fournis des raccourcis qui sont utilisés dans les principaux fichiers. Nous avons ensuite six classes : store, model, template, view, controller et app.

Le fichier **app.js** sert de point d'entrée de l'application. En effet, il va initialiser les différents objets. Lors d'une action ou de changements causés par l'utilisateur, le fichier **controller.js** notifie **model.js** pour mettre à jour la base de données, et **view.js** pour mettre à jour le visuel. Le fichier **store.js** enregistre la data elle-même dans le `LocalStorage` et le fichier **model.js** communique avec le fichier **store.js**. Le fichier **view.js** met en place l'HTML en communiquant avec le fichier **template.js**.

6. Tests unitaires

Les tests sont détaillés dans le fichier `ControllerSpec.js` situé dans le dossier test. Pour les lancer, il suffit d'ouvrir le fichier `SpecRunner.html`, également situé dans le dossier test, dans votre navigateur.

Les tests suivants ont été complétés :
`should show entries on start-up`
`should show active entries`
`should show completed entries`
`should highlight "All" filter by default`
`should highlight "Active" filter when switching to active view`
`should toggle all todos to completed`
`should update the view`
`should add a new todo to the model`
`should remove an entry from the model`

Pour plus d'informations sur l'utilisation de Jasmine, veuillez-vous référer à la documentation disponible à cette adresse : https://jasmine.github.io/.