Step 4: Technical Documentation

1 – Product information

Todolist is a tool that keep keep track of your tasks. A very easy and fast and intuite app.

1.2 – Features

- Add todo
- Edit todo
- Delete todo
- Mark a todo as completed
- Toggle all todos as completed
- Filter todos
- Clear all completed todos

1.3 - Adding a todo

Click on the input field ‘What needs to be done' and enter the title of your new todo.

1.4 - Editing a todo

Double click on the title of the item you would like to edit.

1.5 - Deleting a todo

Hover over the title of the item you would like to edit, a close icon appears. By clicking on the icon the item is deleted.

1.6 - Marking a todo as completed

Click on the check symbol in the item field, the check symbol turns green and the todo is marked as completed.

1.7 - Toggling all todos to complete

Click on the arrow in the input field, all items will be toggled as completed.

1.8 - Filtering todos

Click on one of the buttons below the list, the list will update and only show items of the chosen category.

1.9 - Clear all completed todos

Click on one of the 'Clear completed' button below the list, all completed items will be deleted.

2 - Technical Information

2.1 - Code Base

This application was built using Node.js (todoMVC) as framework with HTML, CSS, JavaScript files.

MVC architecture

MVC stands for MODEL - VIEW - CONTROLLER.

MODEL, VIEW and CONTROLLER are three different entities separate from each other.

MODEL and VIEW never interacts together. They can do it only through CONTROLLER.

MODEL and VIEW can interact only with CONTROLLER.

CONTROLLER is only one entity which can interact with MODEL and VIEW and it acts like connection between them.

2.2 - Files and Methods

.HTML FILES

index.html - Our application starting file (in other words - It is the application entry point.)

.CSS FILES

index.css - Defines our application CSS styles

.JS FILES

app.js - Sets up a brand new Todo list.

model.js - Creates a new Model instance and hooks up the storage.

Methods:

- create - Creates a new todo model
- read - Finds and returns a model in storage
- update - Updates a model
- remove - Removes a model from storage
- removeAll - Removs all data from storage
- getCount - Returns a count of all todos

controller.js - Takes a model and view and acts as the controller between them.Methods:

- setView - Loads and initialises the view.
- showAll - Will get all items and display them in the todo-list.
- showActive - Renders all active tasks.
- showCompleted - Renders all completed tasks.
- addItem - Adding new item to our todos list.
- editItem - Triggers the item editing mode.
- editItemSave - Finishes the item editing mode.
- editItemCancel - Cancels the item editing mode.
- removeItem - Remove item from the DOM and also remove it from storage.
- removeCompletedItems - Will remove all completed items from the DOM and storage.
- toggleComplete - Toggles item between completed and not completed (active).
- toggleAll - Will take all todos and make them complete or incomplete.

- _updateCount - Update number of todos remaining as incomplete (active).
- _filter - Re-filters the todo items, based on the active route.
- _updateFilterState - Simply updates the filter nav's selected states.

helpers.js - It functions are:

- Getting element by CSS selector and attaching event listener to it.
- Attaching a handler to event for all elements that match the selector.
- Finding the element's parent with the given tag name.
- Allowing for looping on nodes by chaining forEach method.

store.js - Creates a new client side storage object.

Methods:

- find - Finds items based on a query given as a JS object.
- findAll - Will retrieve all data from the collection.
- save - Will save the given data to the DB.
- remove - Will remove an item from the Store based on its ID.
- drop - Will drop all storage and start fresh.

template.js - Sets up defaults for all Template methods such as a default template.

Methods:

- show - Creates an <li> HTML string and returns it for placement in our app.
- itemCounter - Displays a counter of how many to dos are left to complete.
- clearCompletedButton - Updates the text within the "Clear completed" button.

view.js - View that abstracts away the browser's DOM completely. It has two simple entry points:

- bind(eventName, handler) - Takes a todo application event and registers the handler.

- render(command, parameterObject) - Renders the given command with the options.

3 - Bugs fixing

3.1 - Bug which not allows adding new todos to the list (simple typo bug).
controller.js line 95

```
95  -       Controller.prototype.adddItem = function (title) {
```

has been changed into

```
95          Controller.prototype.addItem = function (title) {
```

3.2 - Bug which may leads to potential conflict between duplicate IDs (ID for new todos has been generated randomly which could leads to create duplicated ID's).

Location: **store.js** starting from line 84

```
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  -           }
```

has been changed into

```
84  +           var newId = Date.now();
```

3.3 - Unnecessary code and console log (Console log displayed message when user delete todo)

Location: **controller.js** starting from line 165

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

Peace of code above has been commented out

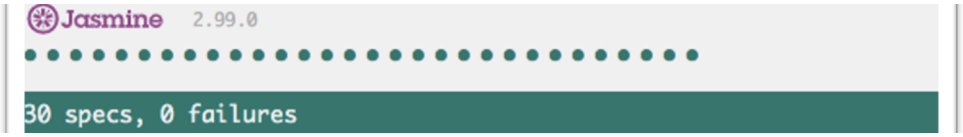
3.4 - Change if else condition to switch case in order to be easy to read and understand -programmer can easily understand the code written with the particular cases as it is divided into separate cases.

Location: **view.js** starting on line 175

```
176 +     case 'newTodo':
177 +       $on(self.$newTodo, 'change', function () {
178 +         handler(self.$newTodo.value);
179 +       });
180 +       break;
181 +
182 +     case 'removeCompleted':
183 +       $on(self.$clearCompleted, 'click', function () {
184 +         handler();
185 +       });
186 +       break;
187 +
188 +     case 'toggleAll':
189 +       $on(self.$toggleAll, 'click', function () {
190 +         handler({completed: this.checked});
191 +       });
192 +       break;
```

4 - Adding tests

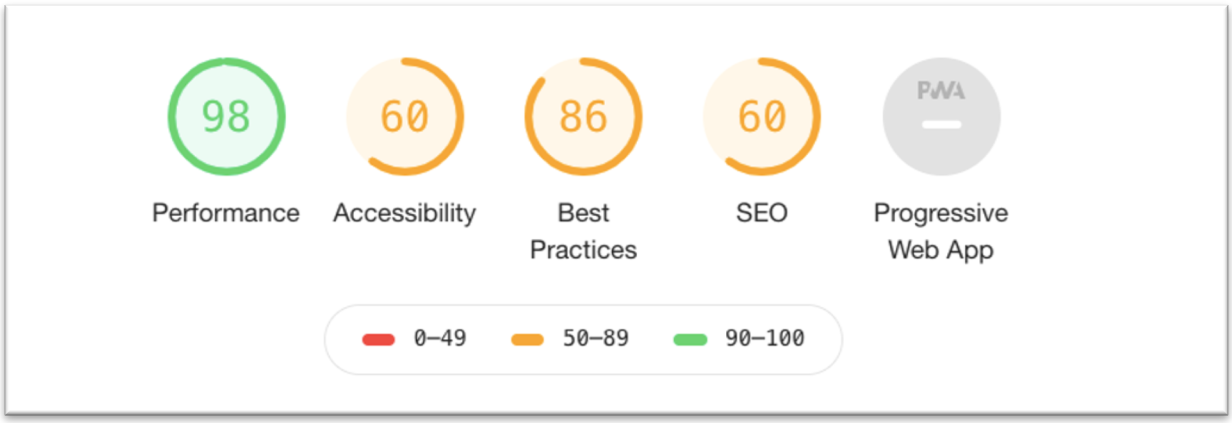
- should show entries on start-up
- should show active entries
- should show completed entries
- should highlight "All" filter by default
- should highlight "Active" filter when switching to active view
- should toggle all todos to completed
- should update the view
- should add a new todo to the model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



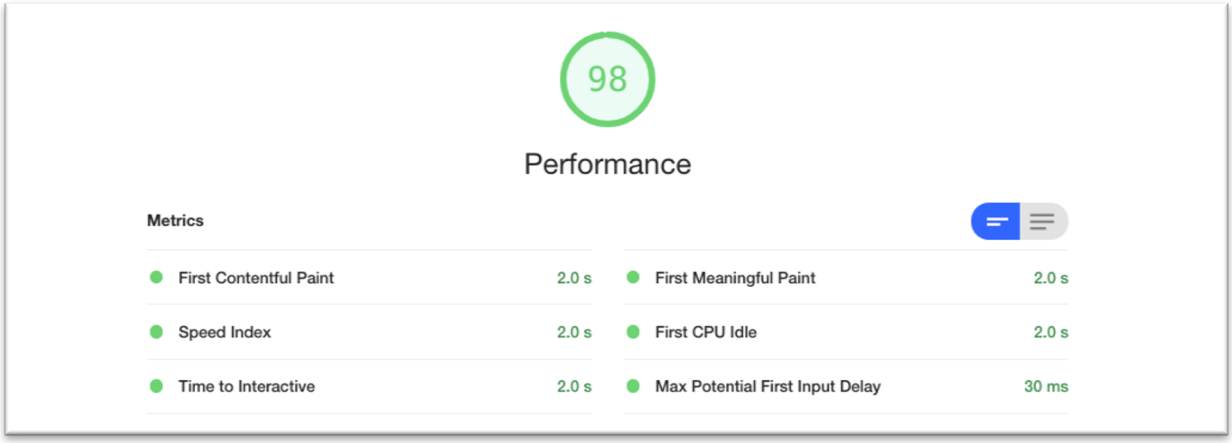
5 - Analyse performance

5.1 - Overall

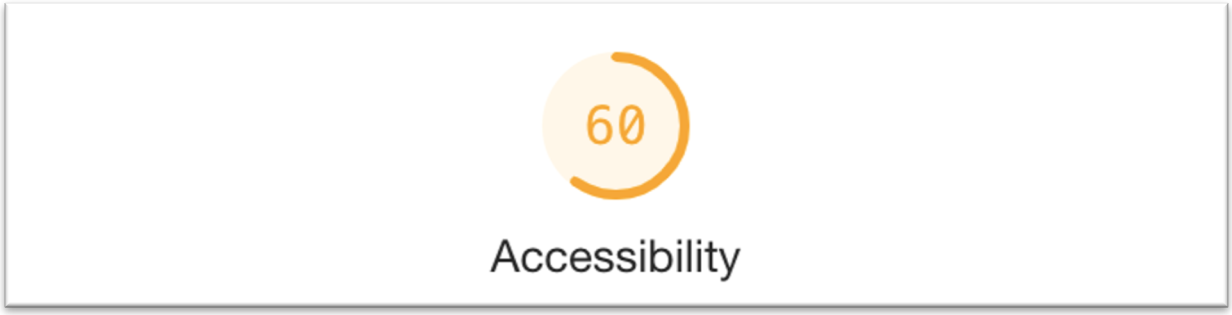
Using the google chrome devtools and auditing the website:



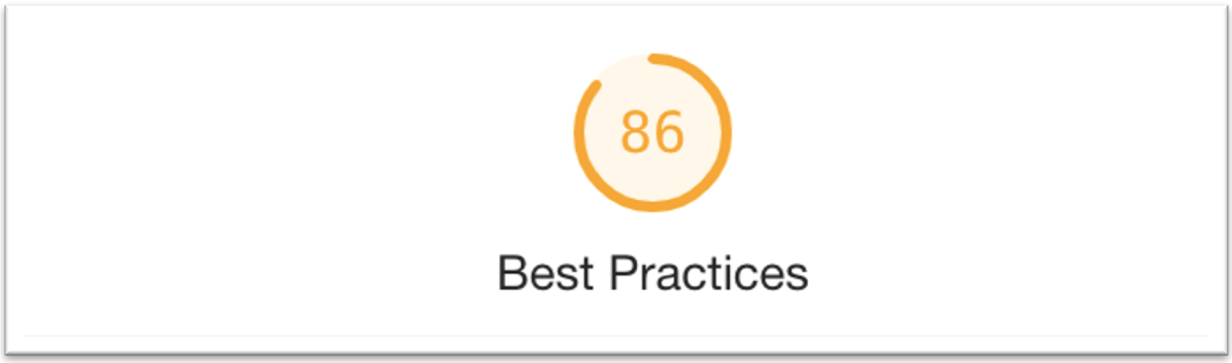
5.2 - Performance



5.3 - Accessibility



5.4 - Best Practices



4.5 - CEO

5.5 - Improvements that should be made :

- Background and foreground colors do not have a sufficient contrast ratio. Low-contrast text is difficult or impossible for many users to read.
- Form elements do not have associated labels. Labels ensure that form controls are announced properly by assistive technologies, like screen readers. Failing Elements - input.new-todo.
- Does not have a <meta name="viewport"> tag with width or initial-scale. Add a viewport meta tag to optimize your app for mobile screens.
- Document doesn't use legible font sizes. Text is illegible because there's no viewport meta tag optimized for mobile screens. Font sizes less than 12px are too small to be legible and require mobile visitors to “pinch to zoom” in order to read.
- Tap targets are not sized appropriately. Tap targets are too small because there's no viewport meta tag optimized for mobile screens. Interactive elements like buttons and links should be large enough (48x48px) and have enough

space around them to be easy enough to tap without overlapping onto other elements.