**TODOS**

**Introduction**.

To-do-app is an application which helps to manage tasks. The user can add new task to to-do list, update and easy delete it, separately or whole list when the tasks are completed. The app has minimalistic design and simply functionality.

**Technical Guide.**

The application has been created by using Model-View-Controller (MVC) – an architectural design pattern that separates three main functionalities: Model – View – Controller. MVC has been used here to add dynamical functionality: user can add new task without reloading the webpage (like Single Page Application – SPA).

**Model** as a central component manages data logic and methods. Here are created prototypes to manage a local storage object and main app functionality like adding, updating and deleting task of the list.

**View** as an output representor displays and manages the user interactions with the application, manipulates DOM structure and represents its functionality.

**Controlle**r as a third part of the MVC pattern connects model and view by converting inputs from the View for the Model component.

**Detailed description of all functions.**

Controller Object– controls interactions between Model and View.

Parameters: model object and view object.

Prototypes:

* setView (loads and initialize the view),
* showAll (displays all items in the todo-list),
* showActive (renders uncompleted tasks),
* showComplited(renders completed tasks),
* addItem (creates new todo task, saving it in the local storage by adding ID),
* editItem (starts editing mode of todo task by matching with the correct ID),
* editItemSave (successfully edits item and save the changing by using matched ID),
* editItemCancel (cancels the item editing mode),
* removeItem (removes item from to-do-list and storage by using its ID as a parameter),
* removeCompletedItems (removes all completed tasks),
* toggleComplete (gives ID and updates the state of completeness of task in the storage),
* toggleAll (change the state of completeness of the tasks: on/off),

Model Object– creates new Model instance and connects it with the storage.

Parameters: storage object.

Prototypes:

* create (creates a new todo model and saves it in the storage),
* read (finds and returns a model in storage, if the query isn’t given, returns everything),
* update (updates a model, every action based on unique ID),
* remove (removes a model from storage),
* removeAll (removes all data from storage),
* getCount (counting active, completed and total tasks by finding the in the storage).

View Object – manipulates DOM structures attached to user interaction. It has two simple entry points:

* bind (takes a todo application event and registers the handler),
* render (renders the given command with the options).

Storage Object – manages data storage by using the local session storage.

Helpers - a bunch of helper methods for querying the selectors and encapsulating the DOM.

Template – delivers template function to display list items, change button states, escape characters.

**Non Techniacal Guide.**

Todo application is used to manage daily activites. For these todo application is categorised into active and completed section.

1. Adding task.

In the application there is an input box where user can insert a task then a user has to press enter (return) button so as to add a particular task in the task list.

1. Deleting task.

On each particular task in the todo list at the right end of each task there is **X** button like of which when clicked it deletes a particular task in the todo list. And that button appers when a task is hoverd.

1. Task completion.

Once a task in the tasklist is completed a user has to mark it as readed of which in the left side of the of each task in the task list there is radio like button. When checked a green tick appear and style of the particular task chnages. And like wise to uncheck the task as not completed.

1. Navigation.

At the bottom of the todos application there are three navigation menus which are **All, Active, Completed** of which **All** displays all task within the task list and **Active** displays the tasks that are not marked as completed and **Completed** displays tasks that are marked as completed.

1. Counter.

At the bottom left of the application there is a counter of which it counts number of the task that are not yet marked as completed.

1. Clear completed.

Once there is any task that is marked as completed at the bottom right there is button called **Clear Completed,** once clicked all tasks that are marked as completed will be deleted from the task list.

**Manual bug fixing.**

During the manual debugging were two bugs found:

1. Type mistake:

* Location: js/controller.js (line 95),
* Controller.prototype.adddItem,

Fixed:

Controller.prototype.addItem.

1. Associated label to the input with a class:

* Location: index.html, (line 16);
* <input class="toggle-all" type="checkbox">

<label for="toggle-all">Mark all as complete</label>

Fixed:

<input class="toggle-all" id=” toggle-all” type="checkbox">

**Testing with Jasmine**

In this Jasmine dubbing process was required to add some tests to already written ones. New tests have to check following cases:

1. ‘should show entries on start-up’

* the ‘todo’ array should be empty, when the application starts;

1. ‘should show all entries without "all" route’

* shows total count of the tasks, array can be empty or filled it with the tasks;

1. ‘should show active entries’

* the completed tasks which are set to false (completed = false);

1. 'should show completed entries'

* the completed tasks which are set to true (completed = true);

1. 'should show the content block when todos exists'

* create a list of the tasks, when they exist;

1. 'should highlight "All" filter by default'

* sets 'all' as default, takes total count, even if it's empty;

1. 'should toggle all todos to completed'

* updates all tasks as completed (model component);

1. 'should update the view'

* updates the status as completed (view component);

1. 'should add a new todo to the model'

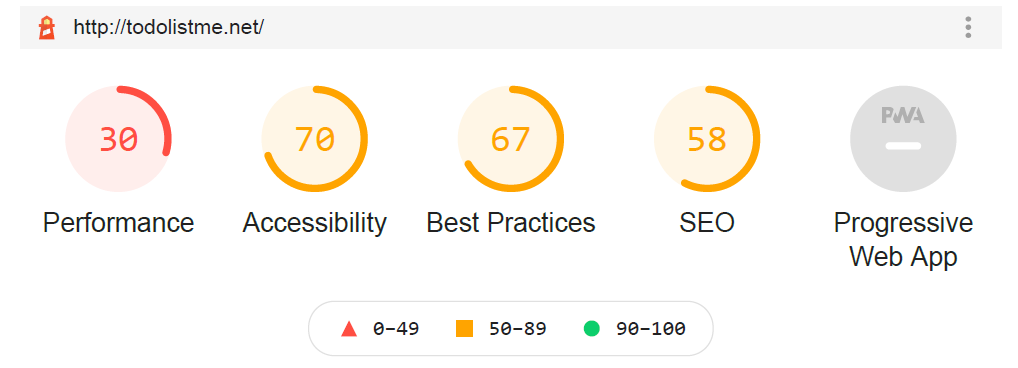
* adds new task to the list;

1. 'should remove an entry from the model'

* removes todo task (model component), empty array.

**Audit Competitor’s Site**

On the side of the competitor’s site, auditing is made possible using Google Lighthouse. And the following are the general outcomes of the result.



The Red colour describes that the average result falls on the **WORSE** category.

The Orange colour describe that the average result falls on the **NORMAL** category.

The Green colour describe that the average result falls on the **GOOD** category.

1. **PERFORMANCE**

Performance of the competitor’s site it is worse due to the following features as addressed in the Google light house:

* 1. Reduce the impact of third-party code.

Use of the third-party code tends to make a load on the performance of the competitor’s site much lower. Some of those third-party code includes

* jQuery CDN
* Google/DoubleClick Ads
* Google fonts and others

Solution:

Reduce the use of the third-party code, through integrating possible codes as among the project file like downloading jQuery source files, google font file and others. This will reduce the time on loading the page.

* 1. Removing unused JavaScript.

In the competitor’s site there are unused script file as a result the consume a lot of the network activity.

Solution:

To prevent the use of unwanted script that can result to high consumption of the network activity.

* 1. Serve images in next-gen formats.

Images that have been used in the competitor’s site are of PNG format and tends to consume high data consumption.

Solution:

The used of JPEG 2000, JPEG XR provides a better compression hence to makes faster download and less data consumption.

1. **ACCESSIBILITY.**

The overall accessibility of the competitor’s site is 70 percent. And the major source that affect the accessibility are:

* 1. Contrast.

Background and foreground colour do not have a sufficient contrast ratio hence it is difficult or impossible for many users to read content.

* 1. Name and labels.
* The use of iframe on the site lack tittle hence it will be difficult for the screen readers user to use the site
* Images that are found in the site lack alt, hence there will be no an alternative text when images fail to load. And others….
  1. Internationalization and localization.

The HTML tag of the competitor’s site does not have a language attribute hence it become so difficult for the screen reader user to determine the language of the site or is it on default language.

1. **BEST PRACTICES.**

The average rate for the best practises of the competitor’s site is 67%, and the major causes that lead to lower the practices are as follows:

1. Trust and safety.

The competitor’s does not use HTTPS hence the site is not protected even once handling sensitive data. Also, the third-party code may contain security vulnerabilities hence its cloud be easy for intruder to attack the site.

1. User experience.

The images used have low resolution and these is due to the low dimensions used on the images.

1. **SEARCH ENGINE OPTIMIZATION(SEO).**

These ensures that your page is optimised for the search engine results ranking. And the average rate for the SEO id 58%. And that rate is caused by the following:

1. Mobile friendly.

The competitor’s site is not mobile friendly due to the missing of the meta tag with name viewport hence it is not accessible to mobile screen

1. Content best practice.

Image elements do not have an alt attribute.

1. Crawling and indexing.

Links are not crawlable, in optimization **“href”** attributes on links are crawl websites, and t ensure that the anchor tag should have a href for link crawling. But anchor tag misses the href.