Technical Documentation

To-do App

I- Summary

To-do app is a great tool for tasks and time management, the app offers a clean user interface with a smooth user experience, through an optimized and robust code.

II- Features

- Create a new todo.
- Edit an existing todo.
- Remove an existing todo.
- Toggle an existing todo as complete.
- Remove all completed todos.
- Filter todos(all, active, complete).

III- UI Description

Legend:

1: text input to enter the title of the todo.

2: text input of the title of the todo item.

3: radio button to toggle a todo as complete.

4: x button to remove a todo.

5: Counter of the todo items.

6: display all todo items.

7: filter active items.

8: filter completed items.

9: remove all complete items

IV- Code Modules Description

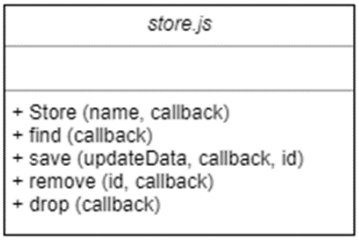
The application is built following the MVC (Model-View-Controller) architecture, which is a pattern that organizes the code into three interconnected parts, making the project more readable, maintainable and robust. Below the list of the application modules with their classes diagrams:

1- “app.js”:

Main Application module

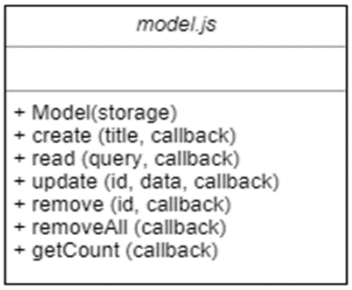
2- “store.js” :

The **database management system** of the app, using the browser local storage.

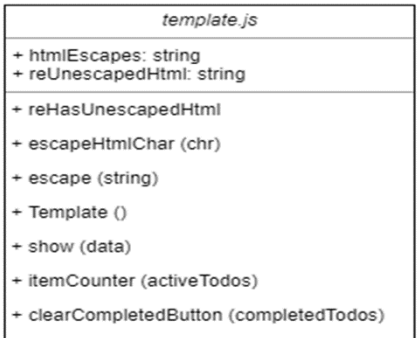


3- “model.js”:

The **Model** in the app MVC, it is a dynamic data structure and works as intermediate between the logic and the storage.



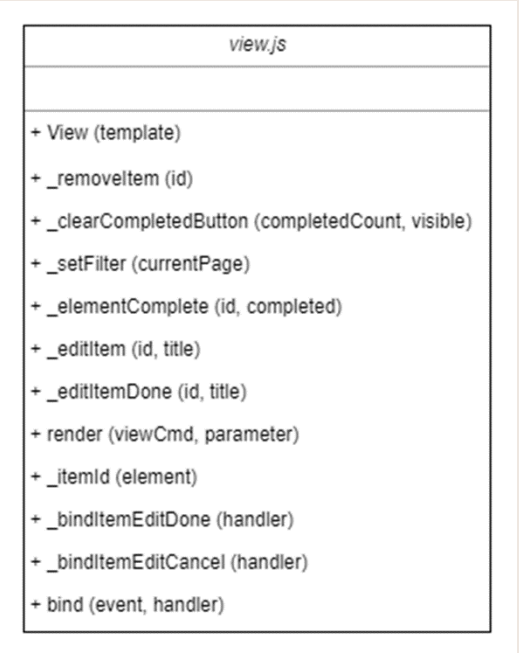
4- “template.js”:



This module is used by the View to build the interface in html.

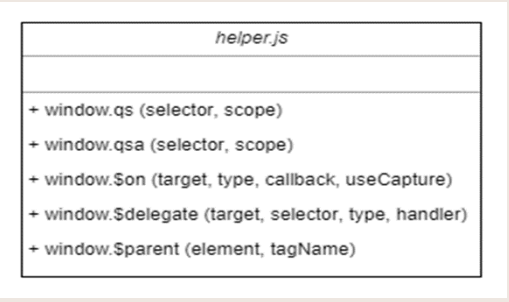
5- “view.js”:

The **View** in the app MVC, it is responsible of the visible part of the app, displaying data and getting user interactions.



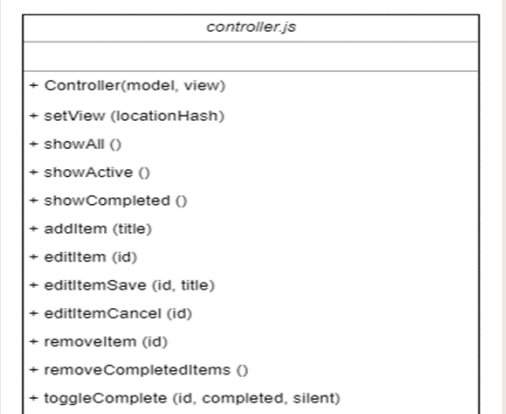
6- “helper.js”

This module is a library of methods used globally by the other app modules.



7- controller.js”:

The **Controller** in the app MVC, it



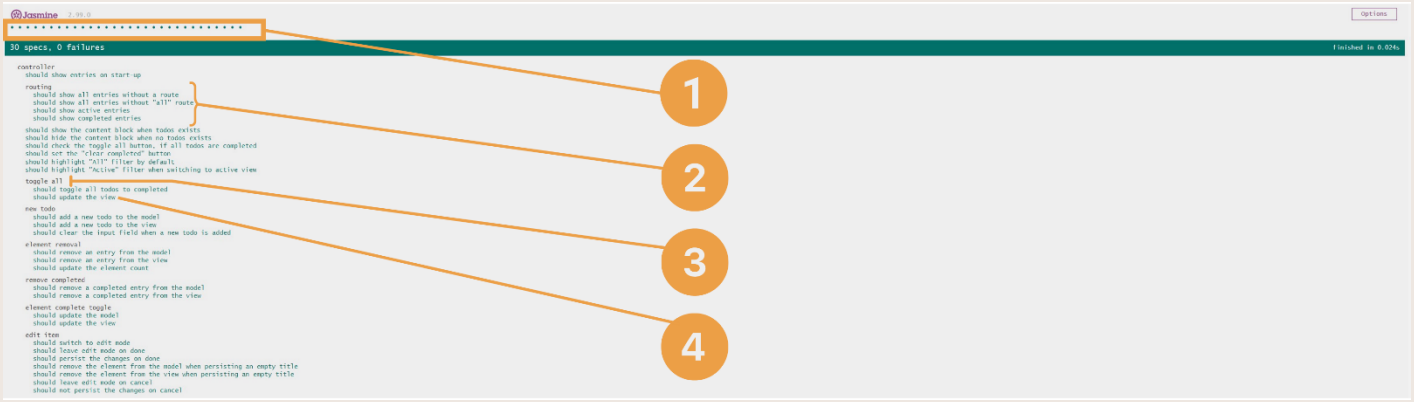
holds the logic of the app, handling user input transferring commands to the model and the view.

V- Testing

The code of the app is provided with a set of tests, built with “Jasmine” the testing framework built for “behavior driven development”.

In the folder “test” there are two file:

1- “SpecRunner.html”: contains the interface to check the tests in a visual way, and get error messages in case some tests fail, otherwise all the dots in the summary section will look green, as in the description below:



Legend:

1: dots for each test, green for success and red for failure.

2: suite of tests.

3: title of a suite of tests.

4: title of a test function.

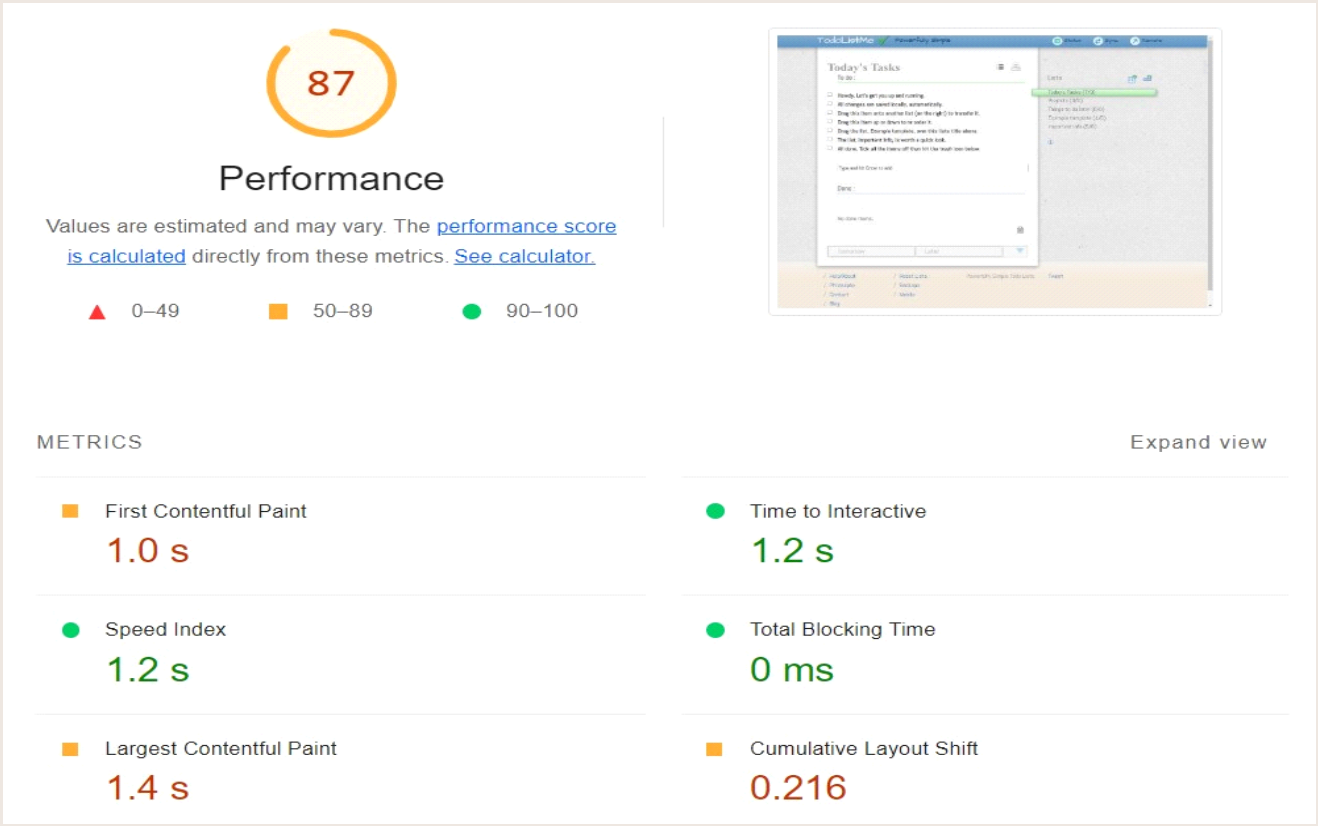
2- “ControllerSpec.js”: contains the specification for the test, written in blocks of “**describe()**”, that describe a suite of tests and “**it()**” as an individual test specification. This is a typical sample of the code:

```
describe("routing", function () {

    it("should show all entries without a route", function () {
        var todo = {title: 'my todo'};
        setUpModel([todo]);

        subject.setView("");

        expect(view.render).toHaveBeenCalledWith("showEntries", [todo]);
    });

});
```

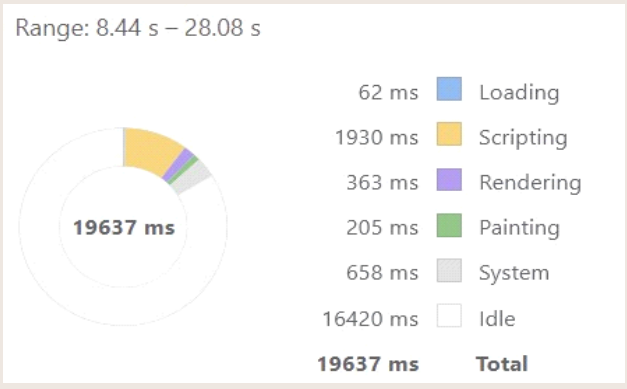
VI- Performance Recommendations

I- Diagnostics of “http://todolistme.net/” performance:

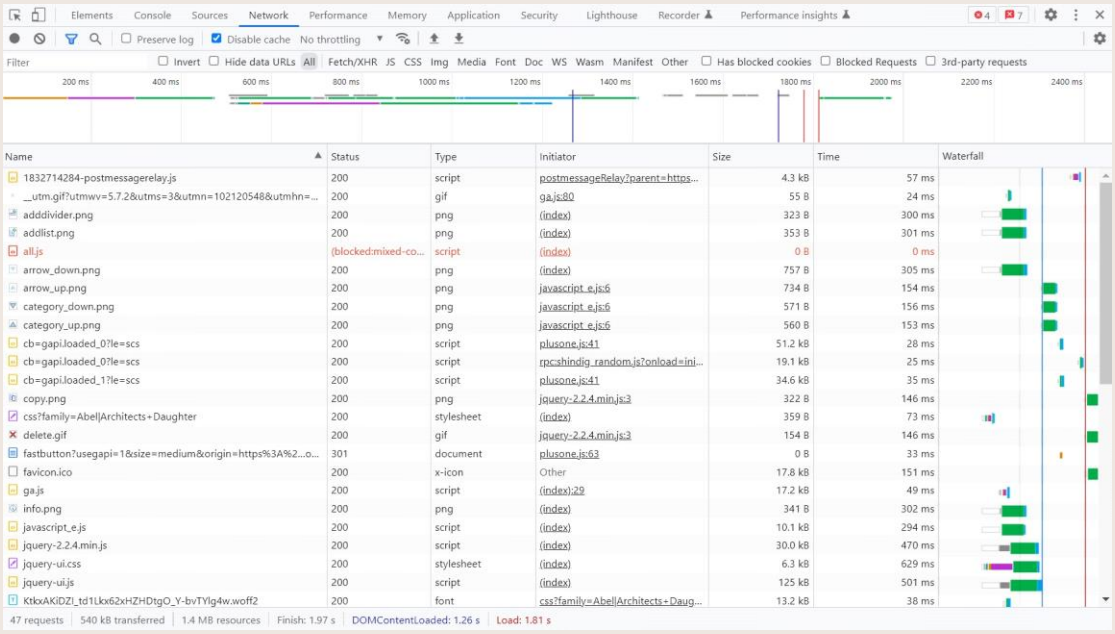
1- Auditing Score: 87%



2- Summary of Performance analysis:



3- Network use analysis:



Performance issues:

- The image file “texture.png” is the top largest resource as shown by “network use analysis”.
- The 2nd file in the largest size resources is the script file “jquery-ui.js”, as shown by the “network use analysis”; there are also other JavaScript files with large size.
- Scripting is taking the most part of the running process.

II- Opportunities to fix performance issues:

- Images compression and appropriately sizing, which means faster downloads and less data consumption, and replace icons by font-icons to reduce time of

interactions. Also appropriately size images, no need for huge images for small display areas of few pixels.

- Minifying JavaScript files can reduce payload sizes and script parse time, JavaScript minification refers to the process of removing unnecessary or redundant data, like white space characters, new line character, comments, which are used to add readability to the code but are not required for it to execute.
- Deferring all non-critical JS and CSS, by default when browser encounters an external script it has to stop and execute it before it can continue parsing the HTML. there are 3 ways to do it:
    - Make JavaScript Asynchronous by using the “async” attribute eg:

      <script **async** src=””></script>
    - Defer the unnecessary scripts for rendering the initial page, until the end.
- Serve static assets with an efficient cache policy [2].

References:

- https://web.dev/uses-webp-images/?utm_source=lighthouse&utm_medium=devtools
- https://web.dev/unused-javascript/?utm_source=lighthouse&utm_medium=devtools
    - **Best practices for cache control:**
- https://medium.com/pixelpoint/best-practices-for-cache-control- settings-for-your-website-ff262b38c5a2

August 2022