ToDo Application

**About**

Todo application in an online Task Manager.

How it works:

* New Todos may easily added, and deleted.
* It allows a user to modify tasks by double clicking it.
* User can mark the task by clicking on checkbox when it is completed and unmark by second click.
* Select all arrow at the top left allows the user to mark all as completed.
* The completed tasks are marked with a tick and crossed out text.
* It allows filtering for All, Active and Completed.
* Clear Completed button lets user to clear all completed tasks.

**Architecture:**

The app uses MVC pattern (Model–View–Controller) .



### Model

Model is where the application’s data objects are stored. The model doesn’t know anything about views and controllers. When a model changes, typically it will notify its observers that a change has occurred.

### View

View is what's presented to the users and how users interact with the app. The view is made with HTML, CSS, JavaScript and often templates. This part of your Chrome App has access to the DOM.

### Controller

The controller is the decision maker and the glue between the model and view. The controller updates the view when the model changes. It also adds event listeners to the view and updates the model when the user manipulates the view.

To summarize, the MVC pattern brings modularity to application developers and it enables:

* Reusable and extendable code.
* Separation of view logic from business logic.
* Allow simultaneous work between developers who are responsible for different components (such as UI layer and core logic).
* Easier to maintain.

## Application's Code Structure

**Main Javascript Files**

* **app.js** -> This is the file that creates a new instance of all the required MVC components.
* **controller.js** -> This is the file that is responsible for the Controller part of the MVC pattern. Contains all the actions that is required for the app, and links the model and view together.
* **helpers.js** -> This file contains helper functionality for the interactions with the DOM such as queries of DOM elements.
* **model.js** -> This file is the file that is responsible for the Model part of the MVC pattern. This file contains functions that manages the data of the App.
* **store.js** -> This file acts as the App's fake database. The storage utilises LocalStorage which creates a temporary file on the users PC and acts like a fake persistent DB, however upon emptying the cache this database will go away.
* **template.js** -> This file contains templates for List elements, HTML escape characters, and other DOM templates that help our App.
* **view.js** -> This is the file that is responsible for the View part of the MVC pattern. This file contains functionality that allows us displaying the data to the user, and visualise the app.

**Folders**

* **js** contains source files of the javascript
* **test** contains Jasmine unit test files.
* **node**\_**modules** contains integrated modules such as jasmine (unit testing) and todomvc (MVC framework).

**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)
* 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

**Testing step 1:**

### Manual bug fixing.

1. Problem: There was a typo in /js/controller.js file at line 95.

These were the original lines. There is an extra "d" letter in addItem.

Fixed:

Controller.prototype.addItem

#### 2. Problem: Toggle all needs to be working by clicking on label as checkbox is hidden by css. index.html (line 16)

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

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

Fixed:

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

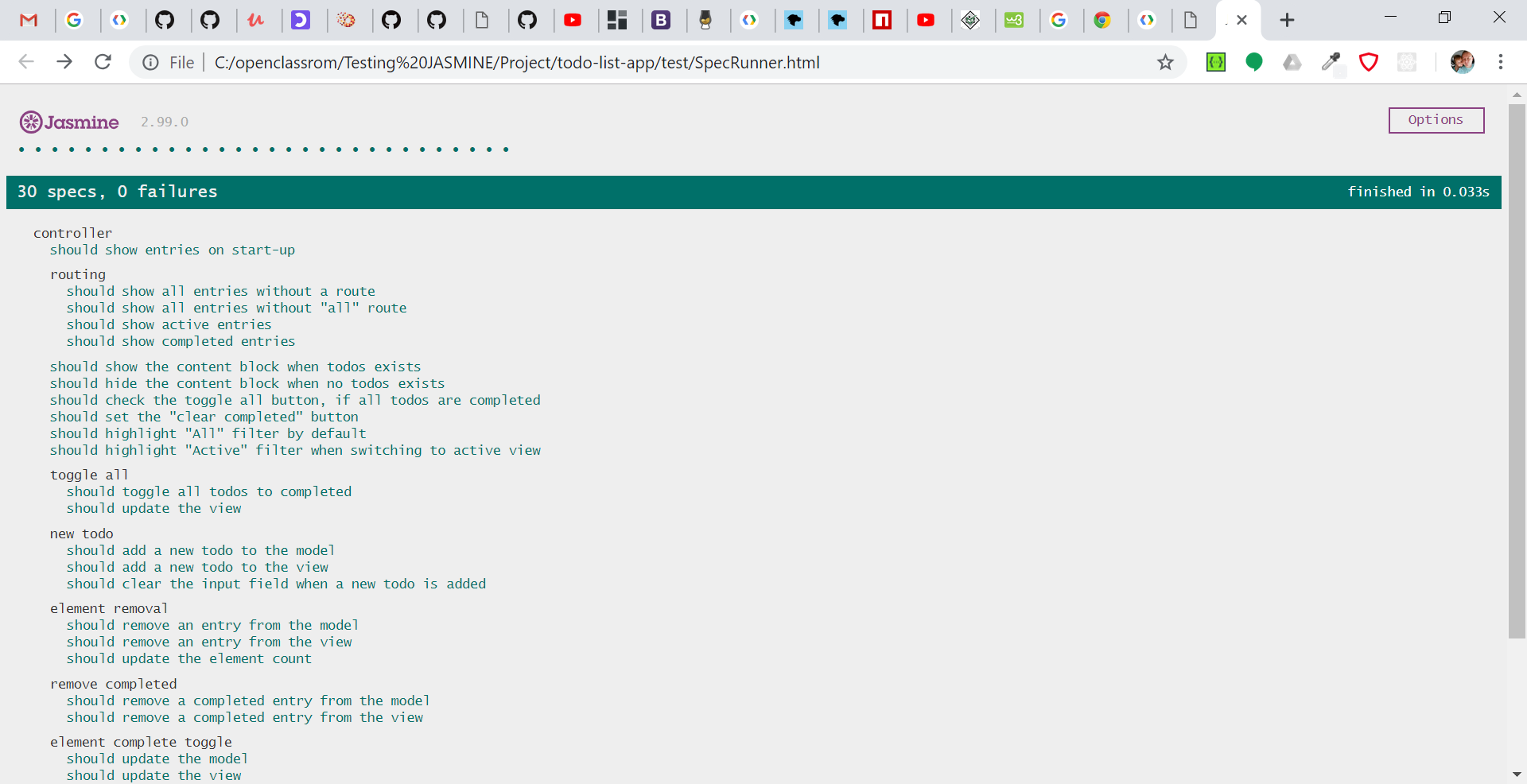
<label for="toggle-all">Mark all as complete</label>

**Testing Step 2:**

#### Automatic Jasmine unit testing.

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
2. should show active entries
3. should show completed entries
4. should highlight "All" filter by default
5. should highlight "Active" filter when switching to active view
6. should toggle all todos to completed
7. should update the view
8. should add a new todo to the model
9. should remove an entry from the model

****

## Audit on ToDo APP

Using Chrome DevTools

The application loads quite quickly due to low content size, using pure javascript, and because it doesn't require any load-time demanding assets such as images, videos, font libraries, etc. The app doesn't load CSS from external CDN servers either.

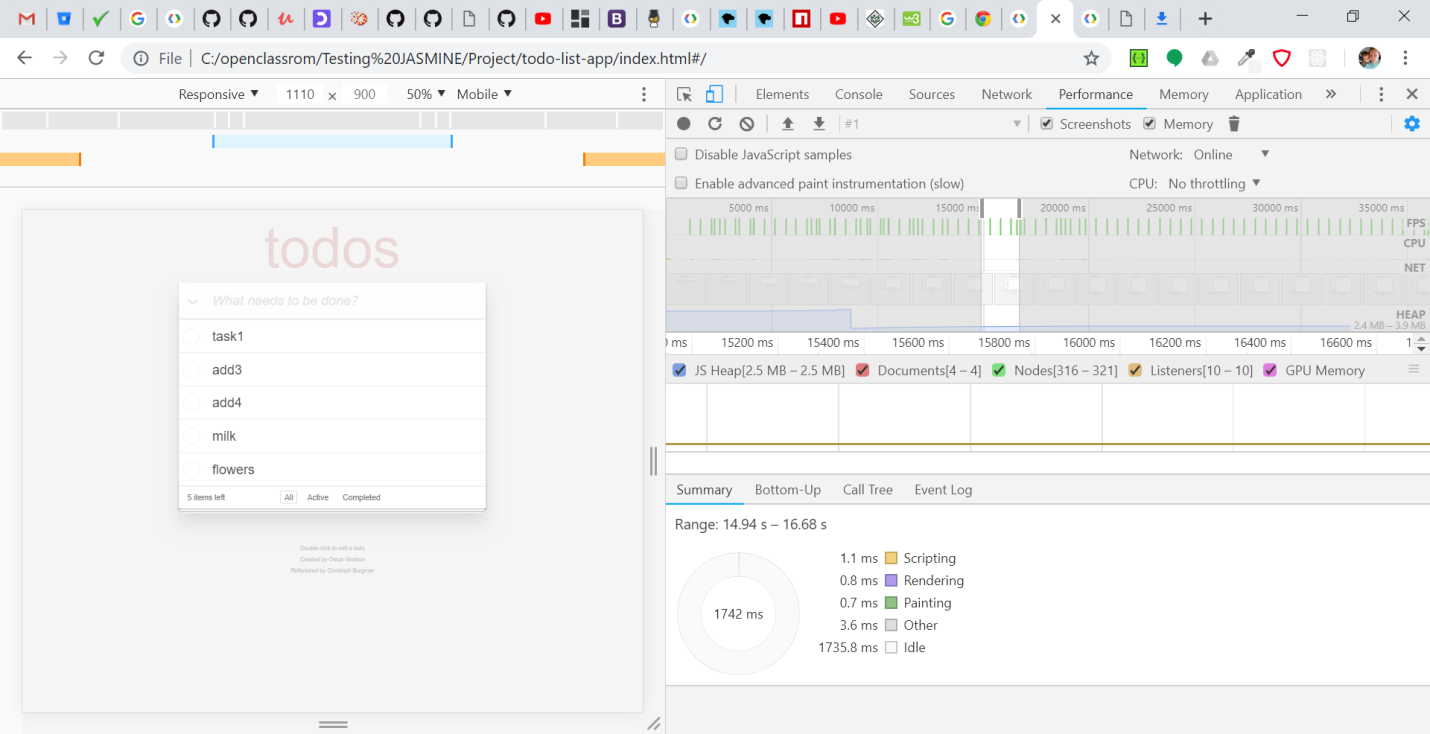
The load tests were made by both simulating low 3G network speeds, and online speeds.  
Result examples will be attached as images below later.

The average results on Win PC using Google Chrome:

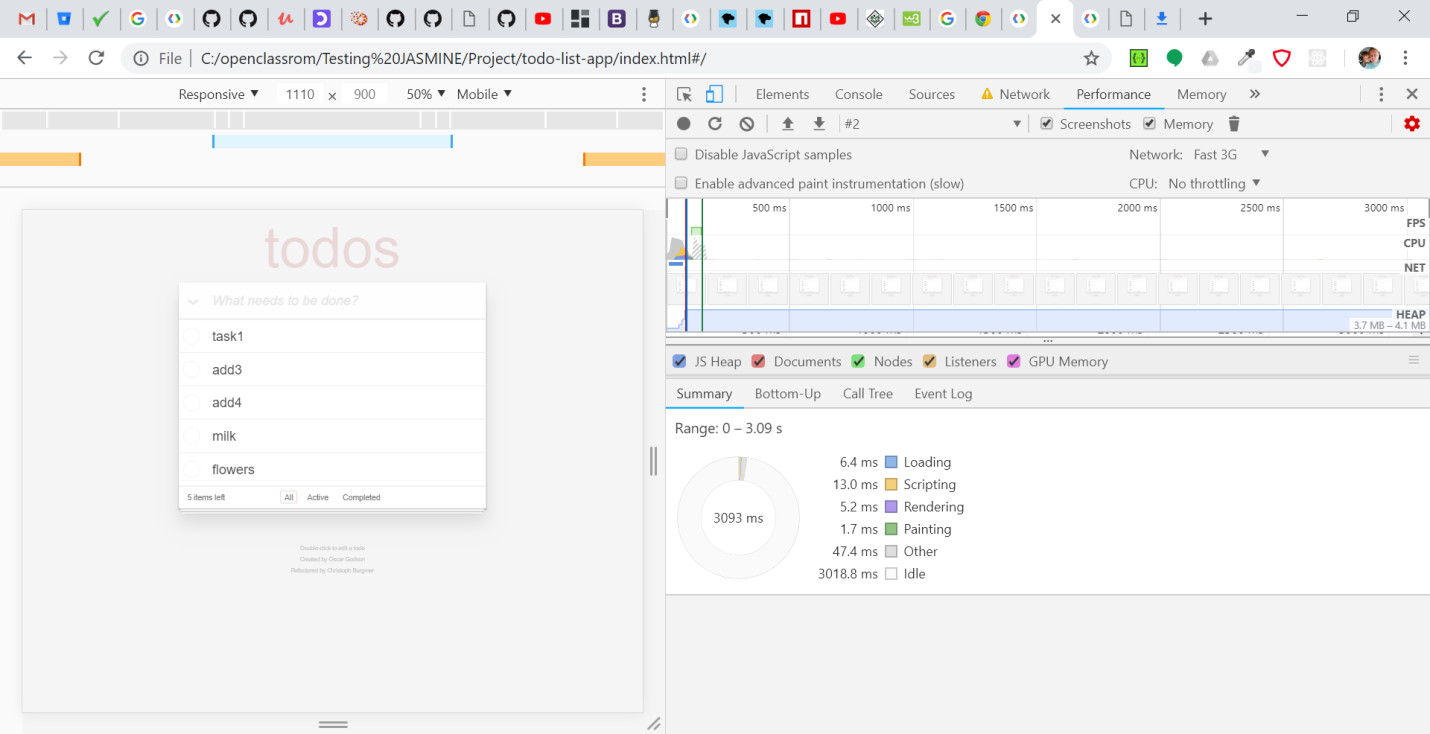
* App size - 50KB
* Load time - 35ms - 50ms
* DOMContentLoaded - 47 ms

Win PC results:

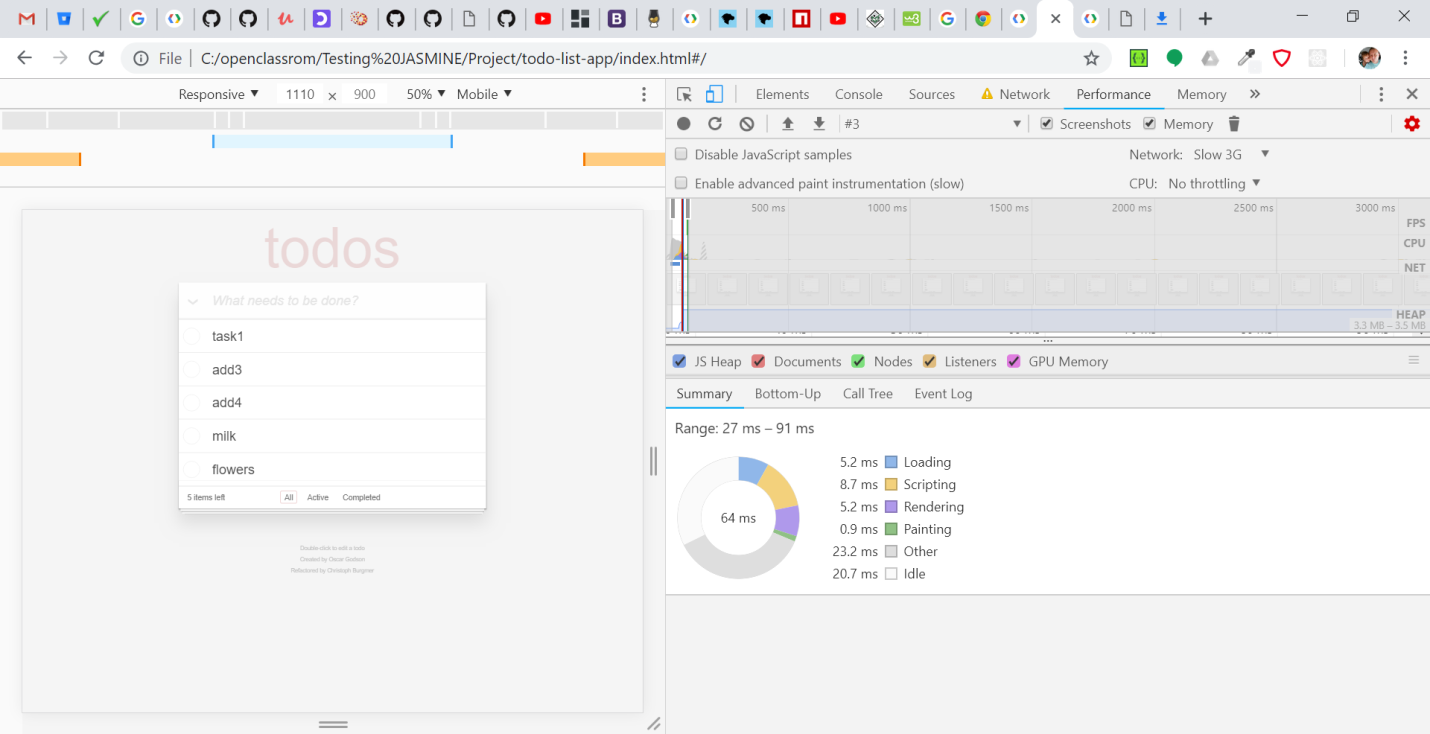
* Page Load - Online network



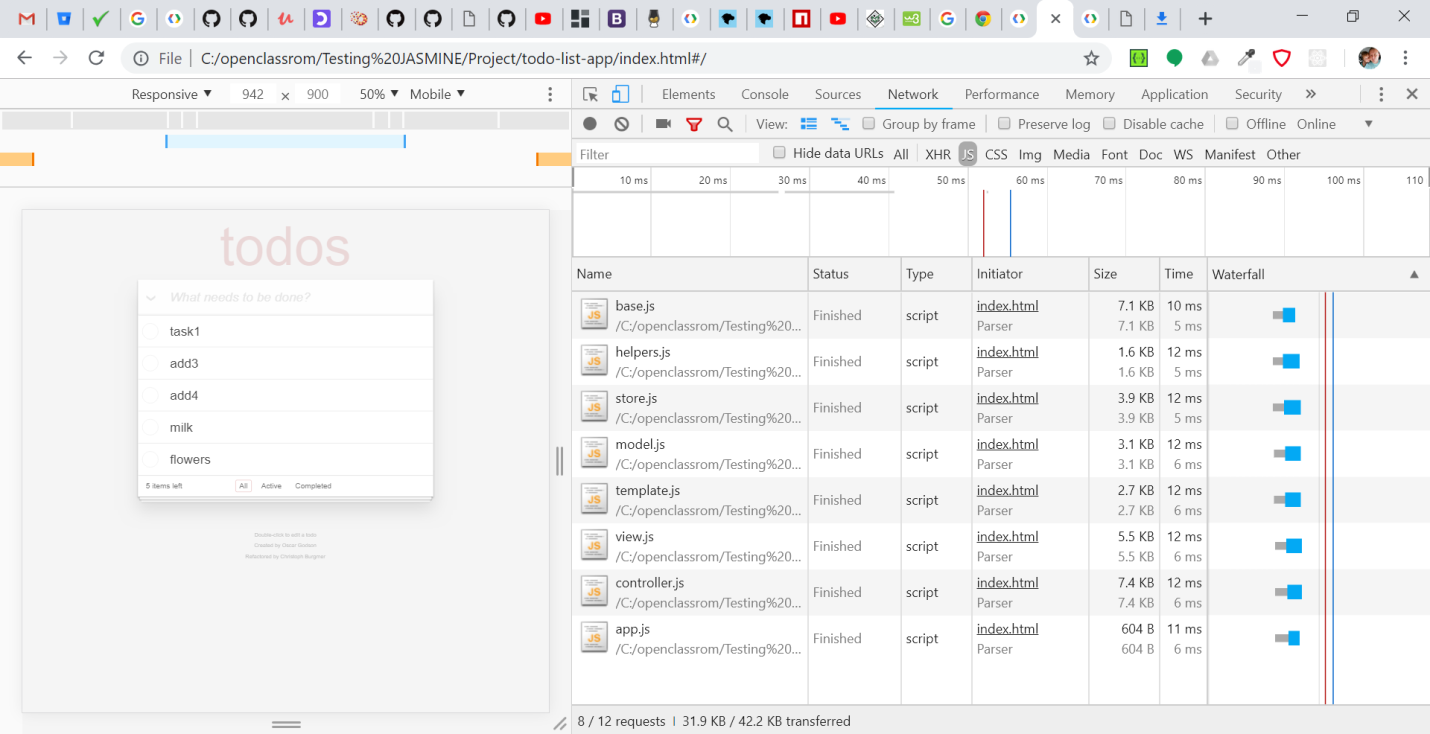
Page Load - Fast 3G network:

****

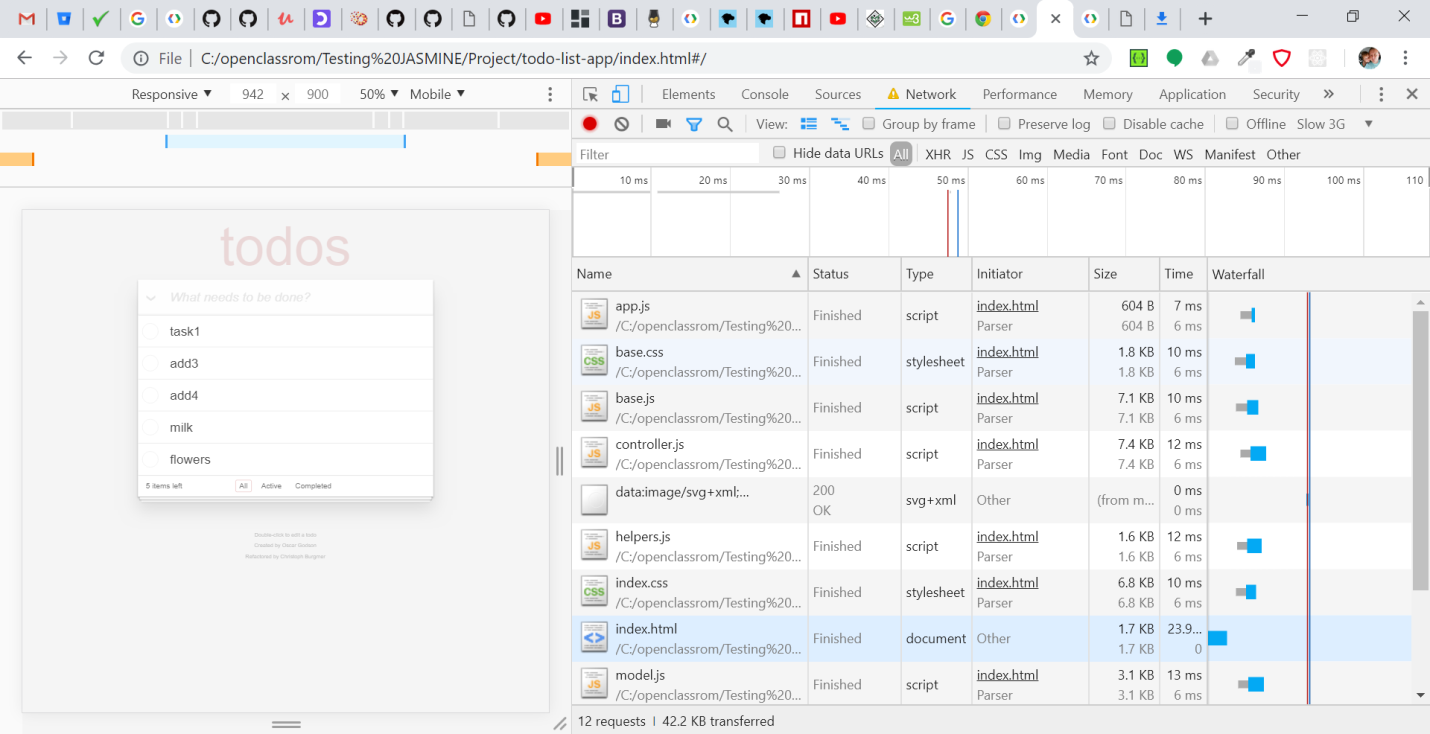
Page Load – Slow 3G network:



**Network: Online**

****

**Network: 3G Slow**

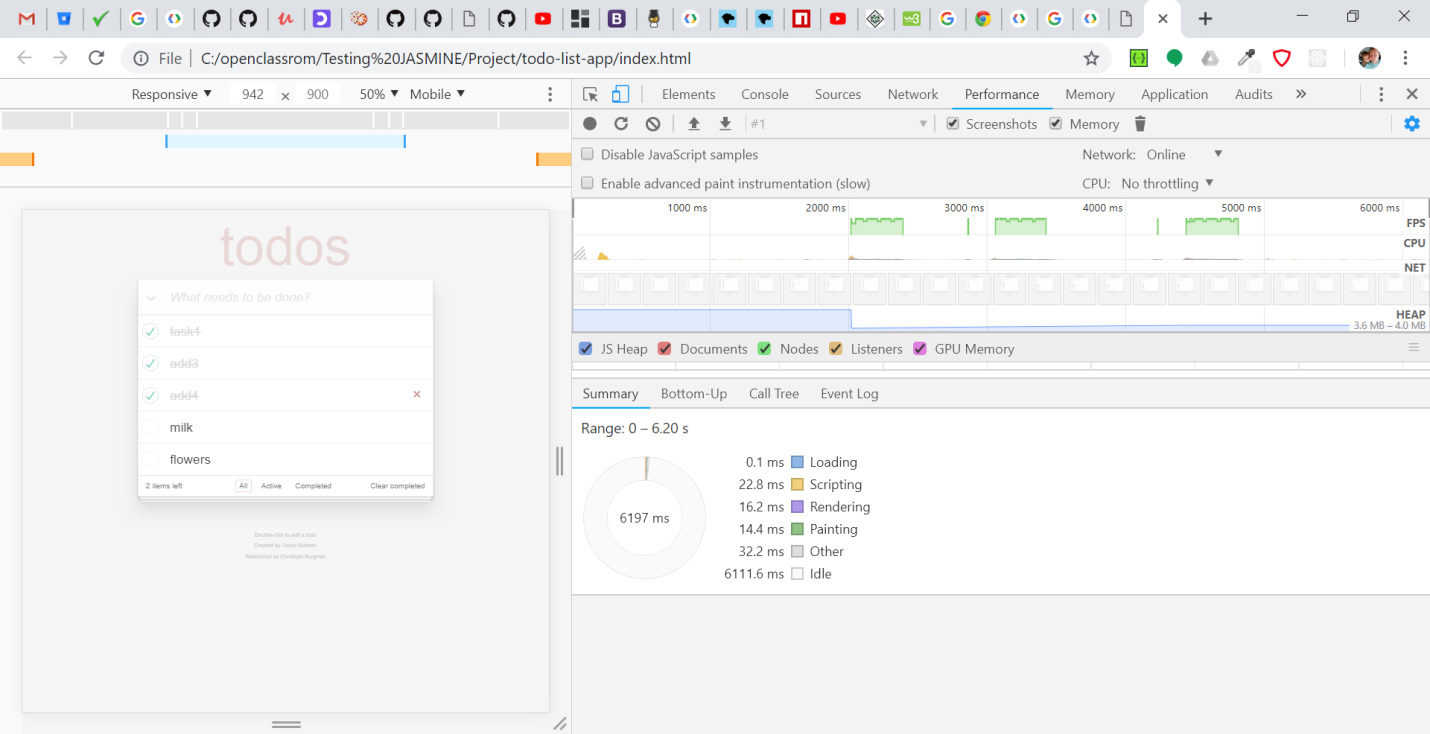
****

**Application use:**

**Marking todo:(recording made in 6.2 sec)**

**22.8 ms scripting**

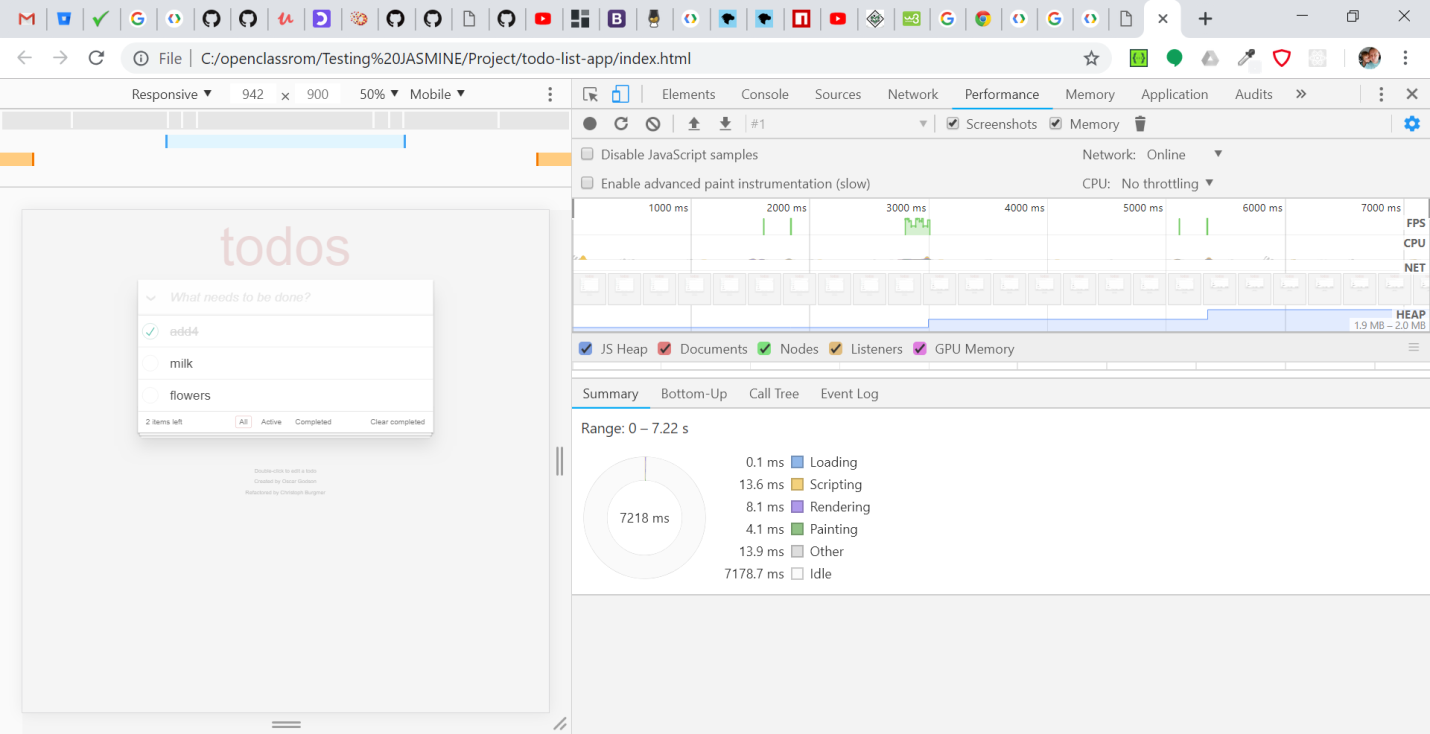
**16.2 ms rendering**

****

**Delete Todo:( recording made in 7 sec)**

**13.6 ms scripting**

**8.1 ms rendering**

****

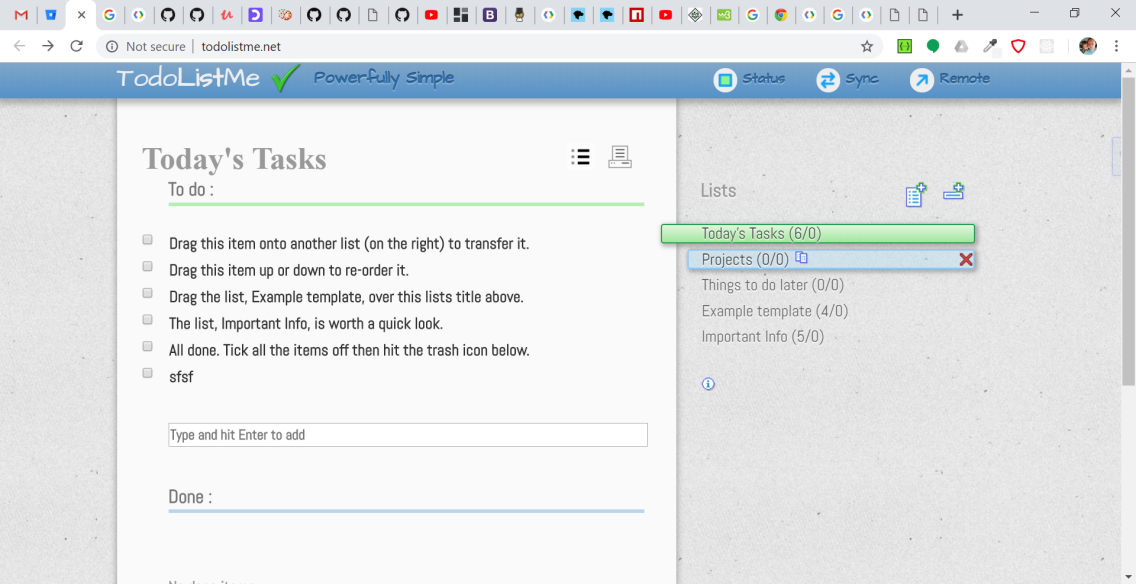
## Competitor App Audit

The competitor site was <http://todolistme.net/> .

The competitor site features a rich app that is a full featured application that focuses around creating and managing several todo list in parallel, scheduling them to calendar dates, maintaining several lists that can have tasks migrated from and to, the list can be ordered and the results can be printed with a single button press.

In addition to the extra features in functionality the visualisation and the UX of the app is much more advanced as well featuring drag and drop functionality, modals/dialogs, pop up windows.

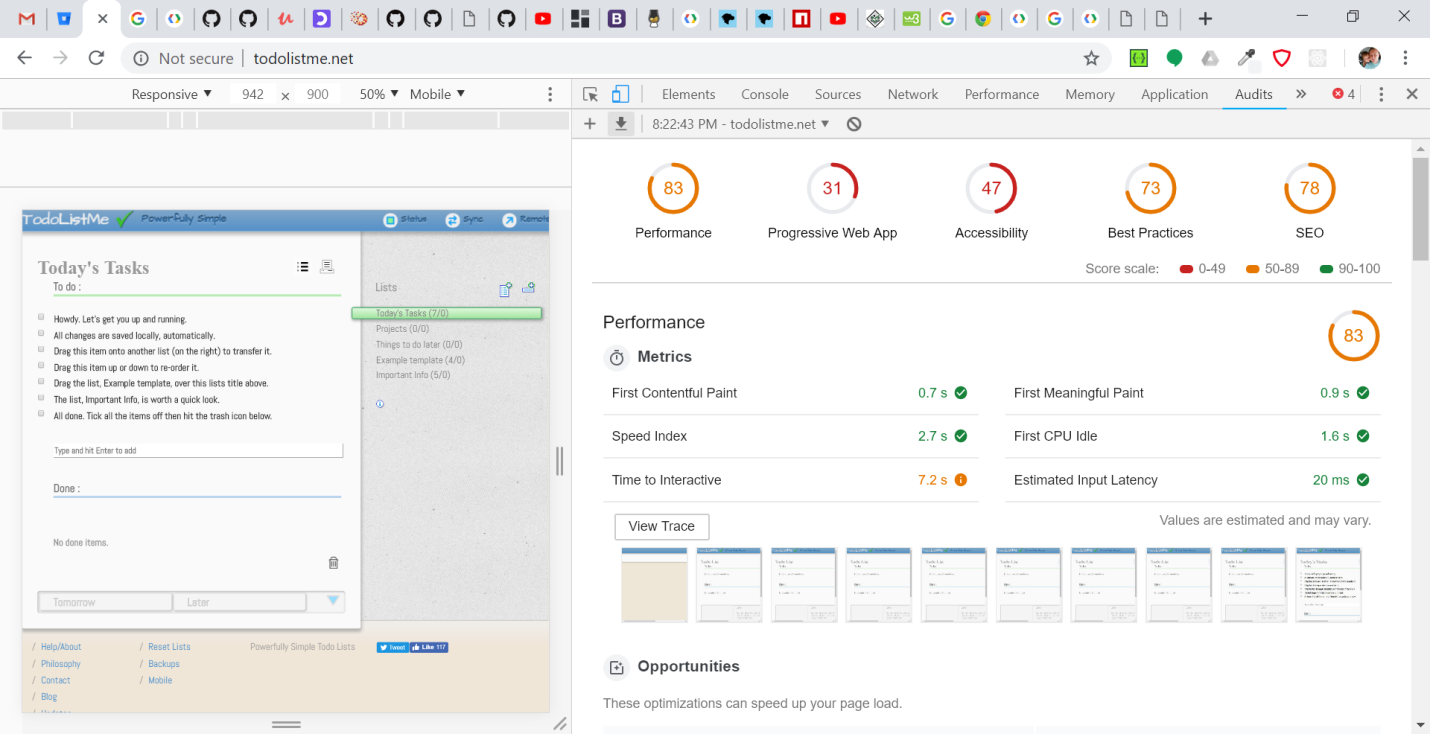
There are several static pages accessible on the competitors site that leads to description infos of various features of the todo manager application. It also has a feature to reset the database to the preset values. It also uses local storage for showing a fake database.

****

**Audit Test Results**

Chrome DevTools is used for testing.

General results of audit test:

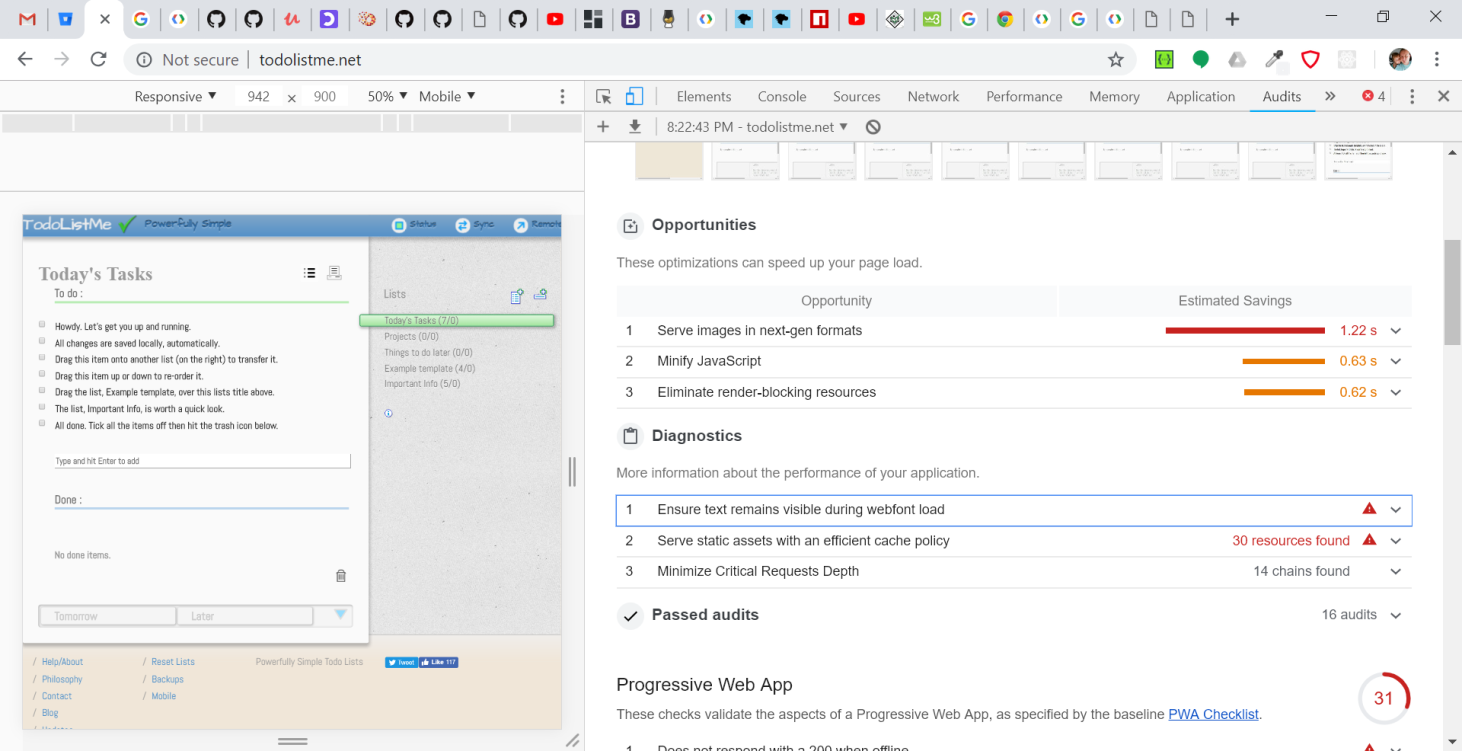


According to de audit report;

First Contentful Paint (0.7s) and First Meaningful Paint (0.9 s) appear fastly but Speed Index (2.7 s),First CPU Idle (1.6 s) and Time to Interactive (7.2 s) values need to be improved.

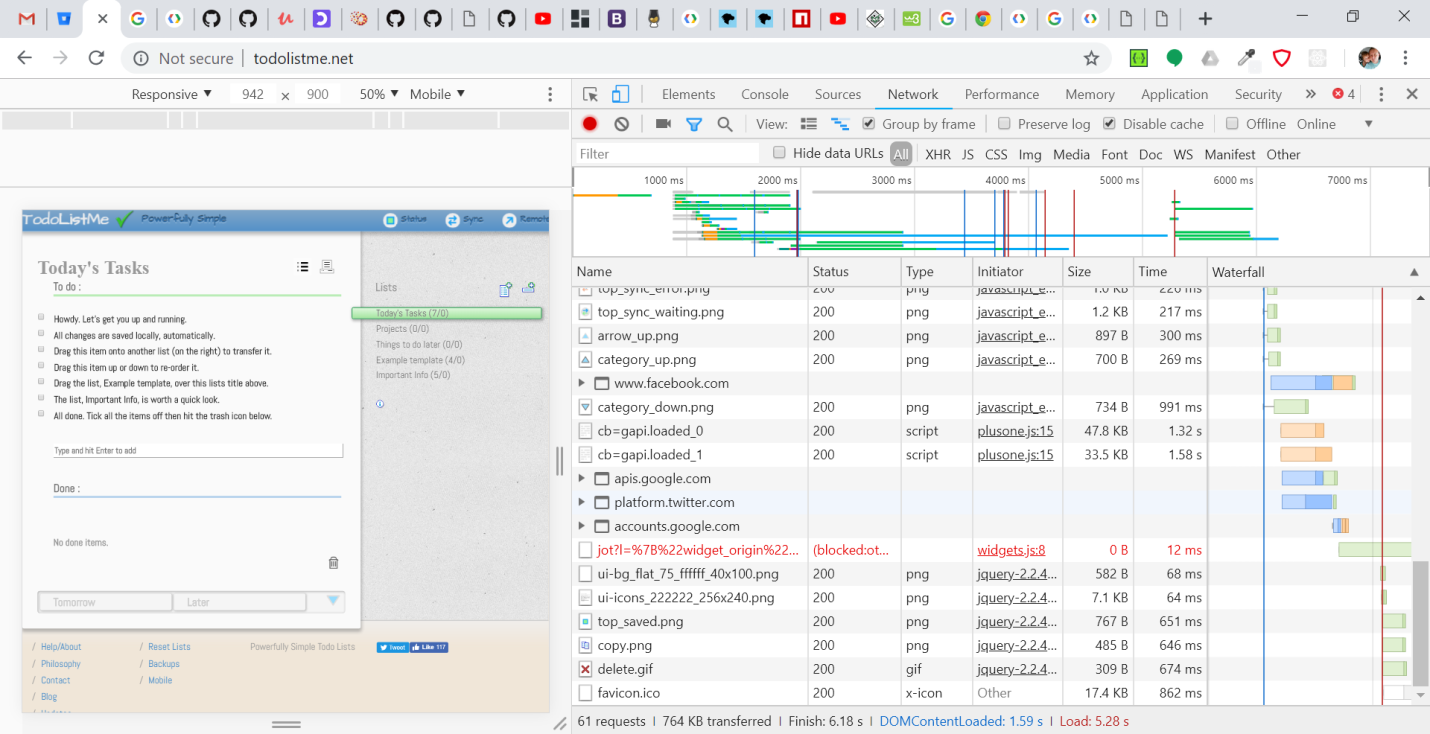
Image formats like JPEG 2000, JPEG XR, and WebP often provide better compression than PNG or JPEG, which means faster downloads and less data consumption. Loading javascript files is the most time consuming activity. Files needed to be **minified and compressed**. Another thing which affects the performance is **advertisement** section from Google Add Services which is loading before loading of tasks from storage.

Detailed performance results are shown in screenshots.

****

**Network performance:**

* App size - 63KB
* Load time – 5.28 s
* DOMContentLoaded – 1.59 s

****

**Overall comparison:**

**ToDo App:**

* App size - 50KB
* Load time - 35ms - 50ms
* DOMContentLoaded - 47 ms

**Competitor App:**

* App size - 63KB
* Load time – 5.28 s
* DOMContentLoaded – 1.59 s

As per above results it can be observed that the competitor app loads significantly slower, allocates more memory, and remains slow and inefficient when adding new todos and operating with its lists and their data.

Our app:

## Pros (+)

* Simple design
* Basic functionality, easy to understand for a user
* Based on MVC model, which is easy to read and develop
* Low data transfer
* Low memory consumption
* Clean and readable code
* Page loads fast, because it based only on html, css and vanilla.js technologies
* Page doesn’t need a large amount of memory, because it doesn’t require any media files
* Application is simple, without any heavy fonts, animations or complicated styles to be load

## Cons (-)

* Limited functionality
* Only local storage in use, not possible to save the data for longer period of time

Competitor App:

## Pros (+)

* Many functionalities
* The data can be saved locally and remotely, after regestration

## Cons (-)

* Very slow loading time
* Memory consumption
* Google Ads delay loading and increases data transfer
* Code isn't well organized
* To many variables decleared globaly