Contents

[1. Introduction 2](#_Toc520273784)

[1.1 Purpose of this Technical Documentation 2](#_Toc520273785)

[1.2 Purpose of the app 2](#_Toc520273786)

[1.3 App structure 3](#_Toc520273787)

[2. Testing 4](#_Toc520273788)

[2.1 Jasmine 4](#_Toc520273789)

[3. Architecture 4](#_Toc520273790)

[3.1 MVC 4](#_Toc520273791)

[3.2 Local Storage 5](#_Toc520273792)

[3.3 Display 5](#_Toc520273793)

[4. Auditing 6](#_Toc520273794)

[4.1 Performance Audit of http://todolistme.net/ 6](#_Toc520273795)

[4.2 Benchmark 7](#_Toc520273796)

[4.3 Optimizing Performance 8](#_Toc520273797)

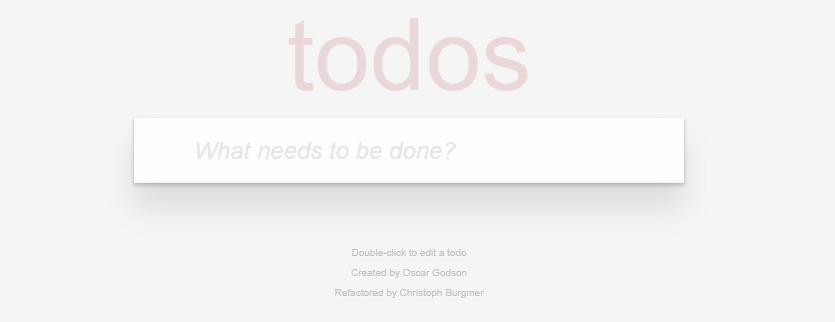
# Introduction

## 1.1 Purpose of this Technical Documentation

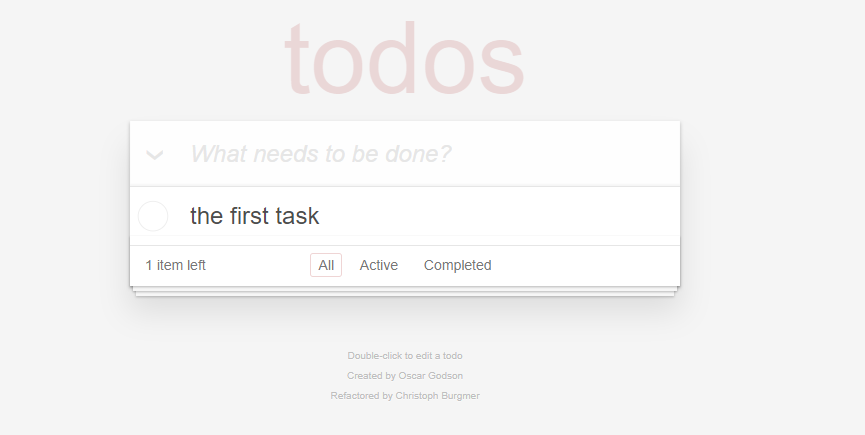
This documentation intends to provide technical information regarding the design pattern and implementation of the ToDo App.

## Purpose of the app

The user can interact with the app to add, edit, and delete tasks that were previously inserted. The app initially loads without any tasks.

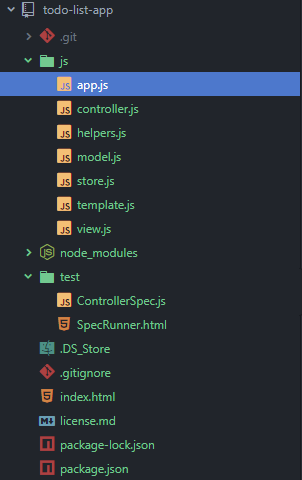


The appearance is changing by adding tasks which can later be manipulated. The image below represents the ToDo app with 1 uncompleted task.



## App structure

In the image below the general structure of the app can be found.



# Testing

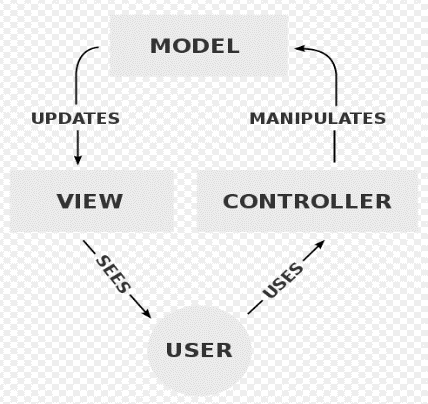
## 2.1 Jasmine

For the successful completion of the project, Jasmine Behaviour Driven Testing was implement to ensure that all critical functionalities and all user events were correctly triggered and provided the desired outcome. All unit testing has been carried out in the file ControllerSpec.js.

# Architecture

## 3.1 MVC

The app is using the architectural pattern of Model-View-Controller. The user is directly interacting with the Controller portion of the app. The controller is handling the user events/inputs which in turn manipulates the model which directly manages the data and logic. When the data has been properly updated, the model in turn will update the view for the user. The benefit of using this architectural pattern is that it allows for simultaneous development of the code. This might not be imminently evident in such a smaller app, but it sets the foundation should the app scale in the future. Below one can see the chain of commands for an MVC pattern:



## 3.2 Local Storage

The app is currently utilizing the local storage for storing the events results i.e. addition of tasks on the list. Which means the list items persist when the page is refreshed. Clearing the storage from the browser will serve the app as new with no tasks present. The local storage is manipulated via the store.js file

## 3.3 Display

The rending of newly added elements are handled by the template.js. This is done by creating an <li> HTML element within the <ul class="todo-list"></ul> in the index.html file when the create event is triggered

# Auditing

## 4.1 Performance Audit of <http://todolistme.net/>

The app performance is at **34% performance** (with + 2% variation per audit) with a series of improvements that can be made (see Figure 1). The app is performing poorly in all major aspects of performance:

* First meaningful paint: **3,850 ms** (measures when the primary content is visible)
* First Interactive: **12,120 ms** (marks the point the page is minimally interactive)
* Consistently Interactive: **Failed**
* Perceptual Speed Index: **12,663 ms** (indicates the content efficiency and the critical rendering path)

Opportunities that haven’t been utilized to increase speed are:

1. Preload key requests: **1,520 ms**
2. Reduce render-blocking stylesheets: **890 ms**
3. Serve images in next-gen formats: **600 ms**
4. Enable text compression:  **500 ms**
5. Minify Javascript: **400 ms**
6. Unused CSS rules: **150 ms**

The extensive First Interactive / Consistently Interactive timeframe is hindering the main purpose of the app which is adding, editing and removing elements to the DOM.

With perfect score in Estimated Input Latency, Avoids Enormous Network Payloads and Avoids an excessive DOM size counters somewhat the performance.

There is a payload related to Google Ads which varies depending on the ad with an average loading time of **3500 ms** (see Fig. 3).

## 4.2 Benchmark

Comparing my site and <http://todolistme.net/> on can see that there is a significant difference in performance. However, it’s important to mention that my site has been audited in a local environment rather a production environment. Auditing the performance in the same environment would yield a more reliable outcome. Below a comparison between the 2 sites in the areas were both **don’t** score 100:

|  |  |  |
| --- | --- | --- |
| Metrics | My App | Competitor |
| Performance | 89 | 34 |
| First meaningful paint | 2630 ms | 3850 ms |
| First Interactive | 3020 ms | 12120 ms |
| Consistently Interactive | 3020 ms | Failed |
| Perceptual Speed Index | 80 | 12 |
| Approximate time reduction after utilizing opportunities | 2250 ms | 4080 ms |
| Inefficient cache policy on static assets | 11 | 68 |
| Passed audits | 15 | 10 |

## 4.3 Optimizing Performance

It is evident that a live app consumes additional resources via links and ads. Before scaling the app one should **Reduce render-blocking stylesheets** by adding the front pages CSS within the head on the main html document. Additionally the javascript files (.js) should be **compressed** (e.g. GZIP) and **minified**. Furthermore, unused CSS rules should be removed to save space and reduce loading time. Should a future scaled app use images, those should use high compression rate by implementing JPEG XR, JPEG 2000 or WebP. Maintaining the principles from passed audits a production ready app can maintain a high performance metric as it currently does.

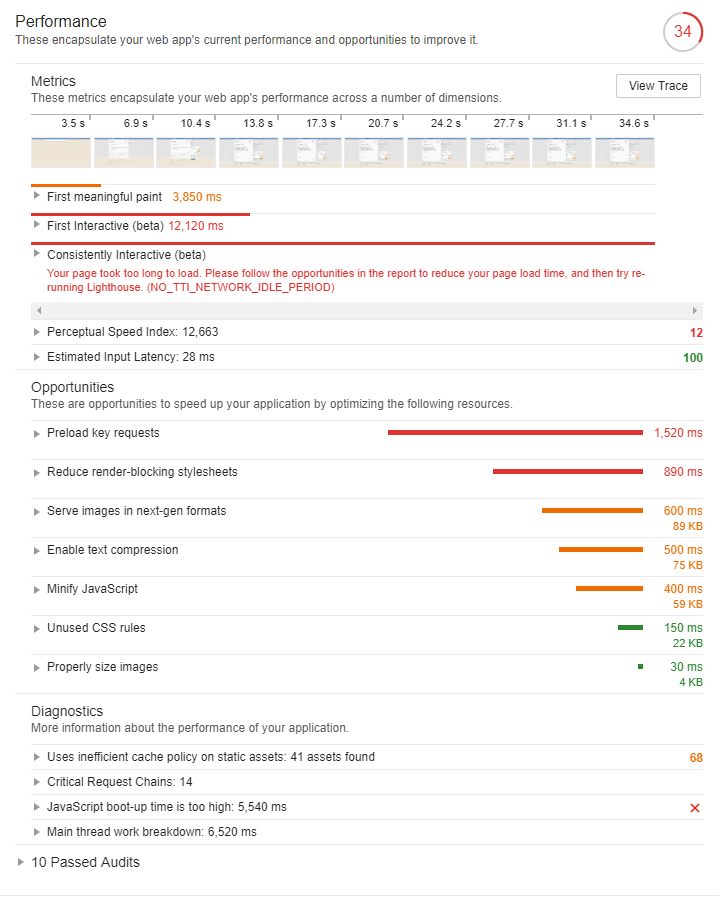


Fig. 1

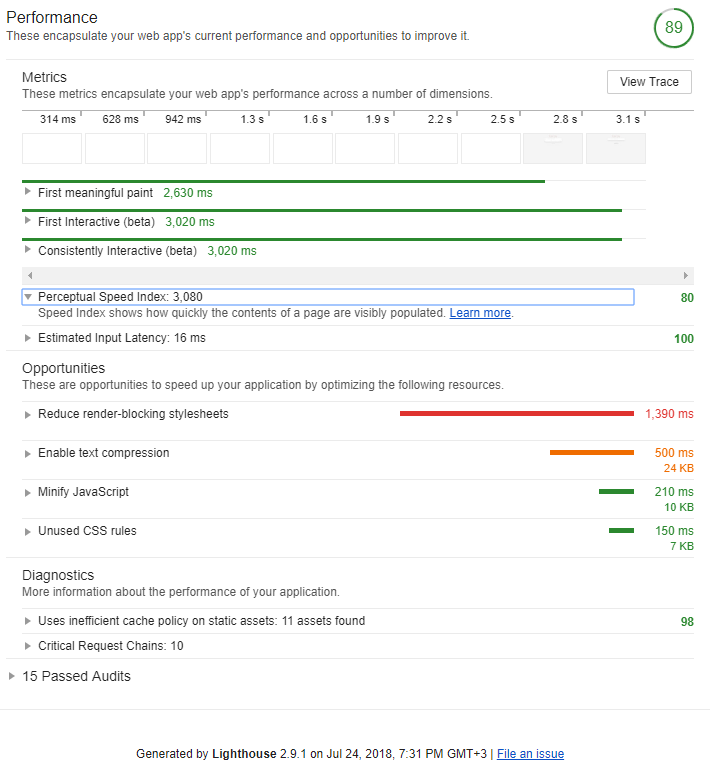


Fig. 2

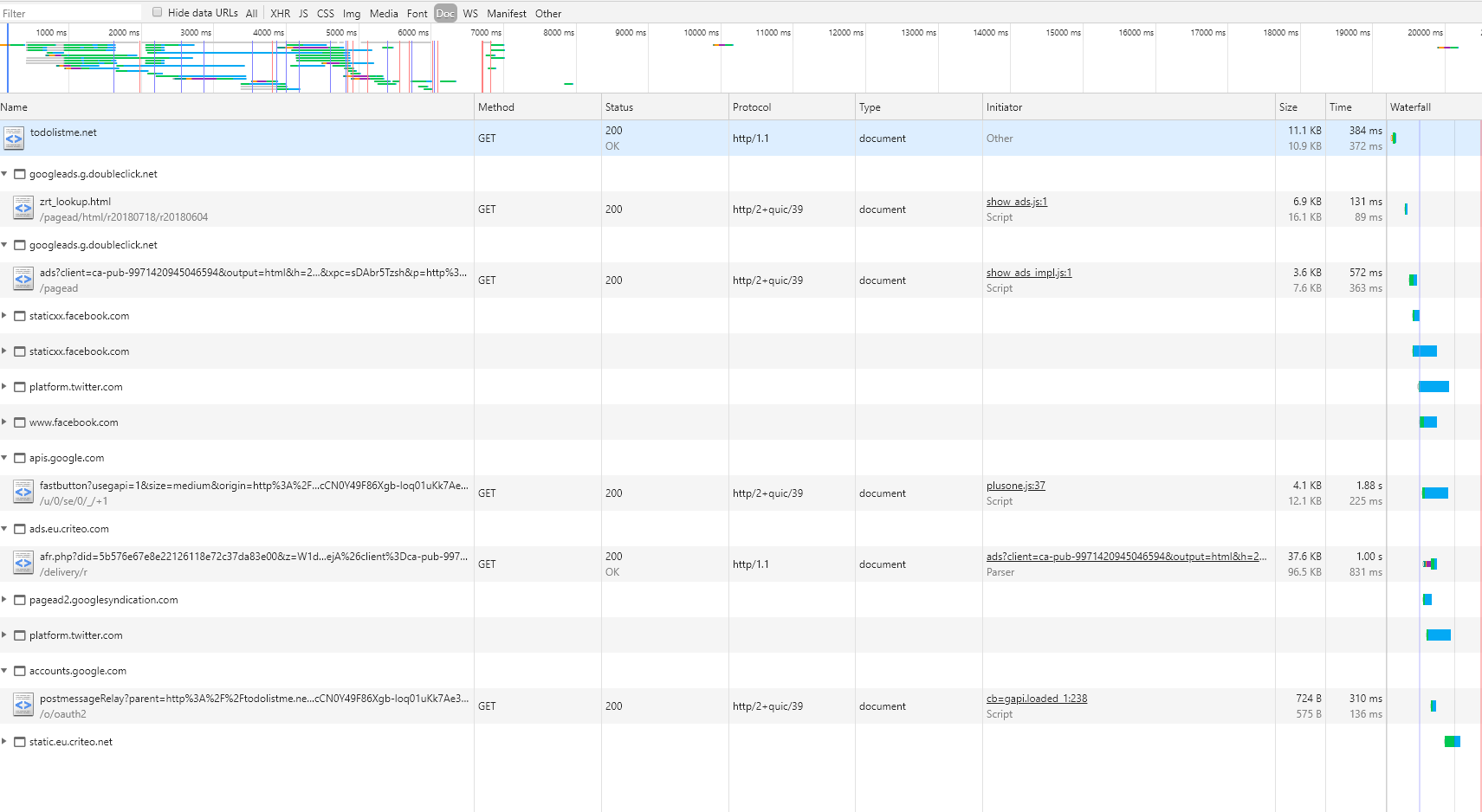


Fig. 3