OPENCLASSROOMS

Technical Doc. for To-Do App

Prepared for: Project 8 of Frontend Path, OpenClassrooms

Prepared by: Mohamed Abdel Nasser

OVERVIEW

Introduction

This is a web application of simple To-Do app, this app enable the user to be able to create, up-date, toggle and delete the tasks in simple way because it has a beautiful and clear design.
This app is implemented with Vanilla JavaScript by adapting “MVC” Model View Controller design pattern, so by using “MVC” it provides the app with flexibility, to be able to extend the app functionality in the future in very efficient way.

SPECIFICATION

This application is implement with three major technologies, so the app tech stack are HTML, CSS and JavaScript, so the rest of specification section will be about these three technologies in our app.

HTML

In the app we have only one HTML file (index.html) which is the entry point of the app and it contains structure of the app.

CSS

The app has two CSS files (index.css) and (base.css), they are representing the visual identity of the app and how it looks like.

JavaScript

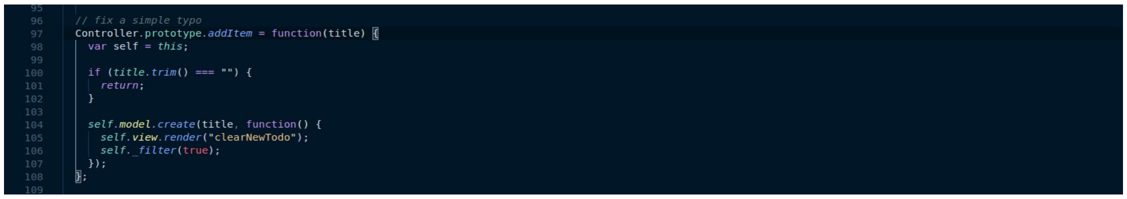
The app has seven JavaScript script files:

1. **app.js:** creates a new instances of the app by initializing model, view and controller
2. **controller.js:** controls the both model an view instances together
3. **helper.js:** contains helper functions to facilitate the interaction between JavaScript and the DOM “Document Object Model”
4. **model.js:** initialize new model instance based on MVC patten.
5. **store.js:** stores the tasks in the browser’s local storage
6. **template.js:** acts like templating engine like (Mustache) to provides helper functions.
7. **view.js:** provides the user with the app functionality, for creating a To-Do and rendering the UI

BUG FIXES

In our App, there are two bugs, one bug is a simple typo but it’s severe enough that to-do items can’t be even added. And the other bug involves an introduction of potential conflict between duplicate IDs.

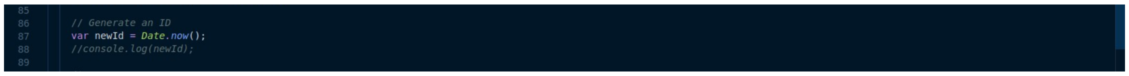
So I’m going to add snapshots of the fixed bug from the app code base:



```
// fix a simple typo
Controller.prototype.addItem = function(title) {
  var self = this;

  if (title.trim() === "") {
    return;
  }

  self.model.create(title, function() {
    self.view.render("clearNewTodo");
    self._filter(true);
  });
};
```

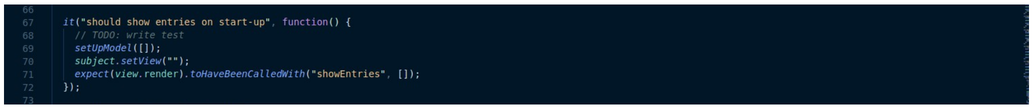


```
// Generate an ID
var newId = Date.now();
//console.log(newId);
```

TESTING

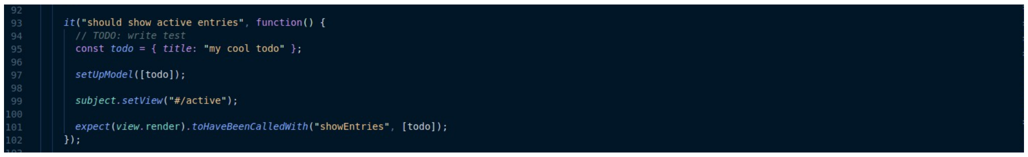
The application also contains unit test by using Jasmine Framework, and there was some features which didn’t have unit test, so we added new test specs to meet the expectation:

✓ should show entries on start-up



```
it("should show entries on start-up", function() {
  // TODO: write test
  setUpModel([]);
  subject.setView("");
  expect(view.render).toHaveBeenCalledWith("showEntries", []);
});
```

✓ should show active entries



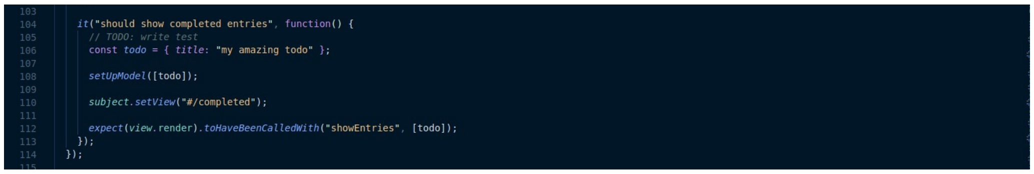
```
it("should show active entries", function() {
  // TODO: write test
  const todo = { title: "my cool todo" };

  setUpModel([todo]);

  subject.setView("#/active");

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

✓ should show completes entries



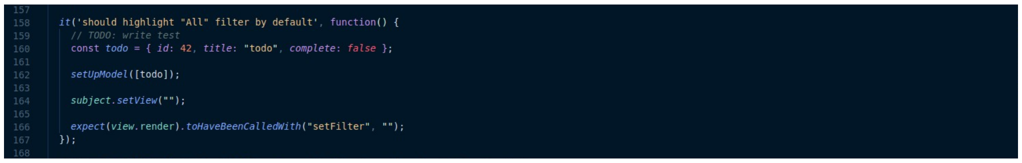
```
  it("should show completed entries", function() {
    // TODO: write test
    const todo = { title: "my amazing todo" };

    setUpModel([todo]);

    subject.setView("#/completed");

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

✓ should highlight “All” filter by default



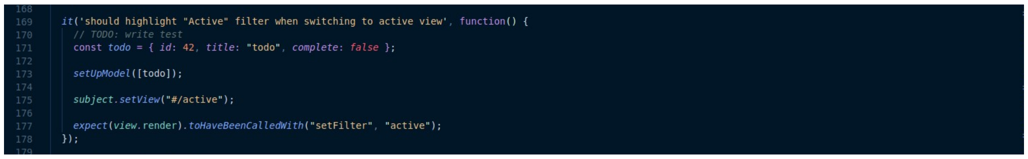
```
it('should highlight "All" filter by default', function() {
  // TODO: write test
  const todo = { id: 42, title: "todo", complete: false };

  setUpModel([todo]);

  subject.setView("");

  expect(view.render).toHaveBeenCalledWith("setFilter", "");
});
```

✓ should highlight “Active” filter when switching to active view



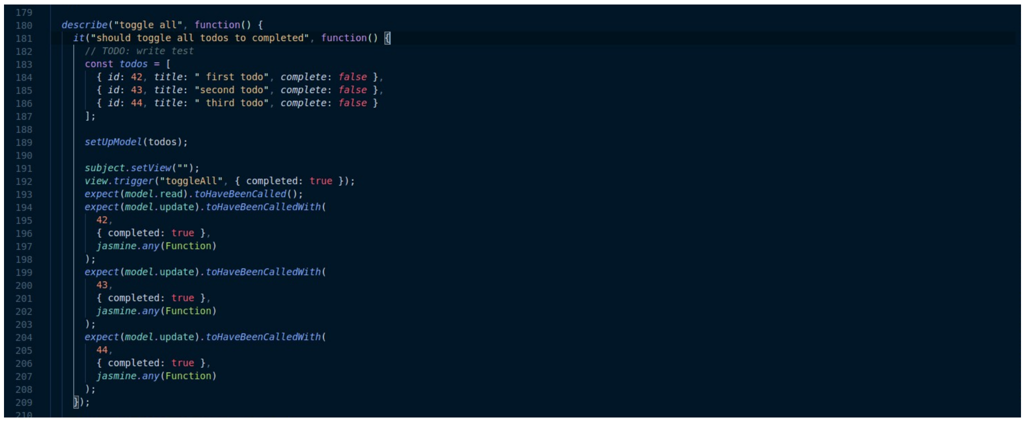
```
it('should highlight "Active" filter when switching to active view', function() {
  // TODO: write test
  const todo = { id: 42, title: "todo", complete: false };

  setUpModel([todo]);

  subject.setView("#/active");

  expect(view.render).toHaveBeenCalledWith("setFilter", "active");
});
```

✓ should toggle all to-dos to completed

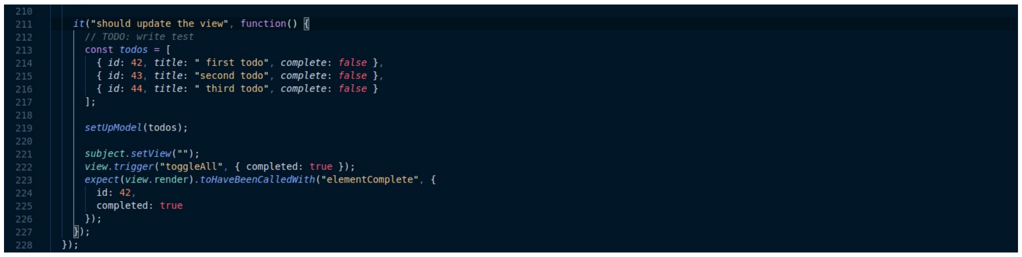


```
describe("toggle all", function() {
  it("should toggle all todos to completed", function() {
    // TODO: write test
    const todos = [
      { id: 42, title: " first todo", complete: false },
      { id: 43, title: "second todo", complete: false },
      { id: 44, title: " third todo", complete: false }
    ];

    setUpModel(todos);

    subject.setView("");
    view.trigger("toggleAll", { completed: true });
    expect(model.read).toHaveBeenCalled();
    expect(model.update).toHaveBeenCalledWith(
      42,
      { completed: true },
      jasmine.any(Function)
    );
    expect(model.update).toHaveBeenCalledWith(
      43,
      { completed: true },
      jasmine.any(Function)
    );
    expect(model.update).toHaveBeenCalledWith(
      44,
      { completed: true },
      jasmine.any(Function)
    );
  });
```

- ✓ should update the view

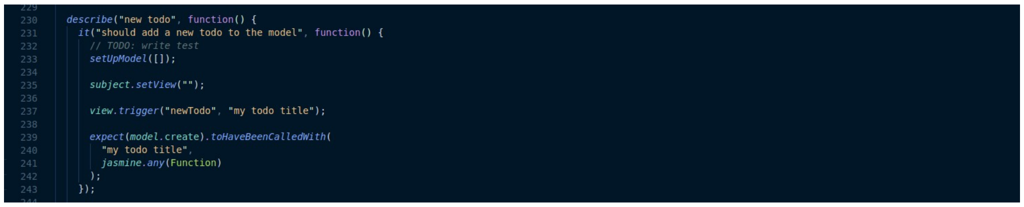


```
    it("should update the view", function() {
      // TODO: write test
      const todos = [
        { id: 42, title: " first todo", complete: false },
        { id: 43, title: "second todo", complete: false },
        { id: 44, title: " third todo", complete: false }
      ];

      setUpModel(todos);

      subject.setView("");
      view.trigger("toggleAll", { completed: true });
      expect(view.render).toHaveBeenCalledWith("elementComplete", {
        id: 42,
        completed: true
      });
    });
  });
```

- ✓ should add new to-do to the model



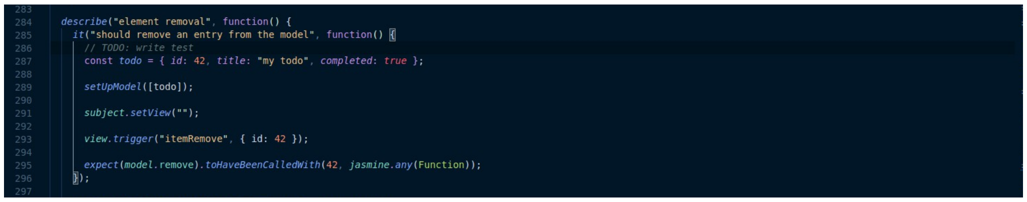
```
  describe("new todo", function() {
    it("should add a new todo to the model", function() {
      // TODO: write test
      setUpModel([]);

      subject.setView("");

      view.trigger("newTodo", "my todo title");

      expect(model.create).toHaveBeenCalledWith(
        "my todo title",
        jasmine.any(Function)
      );
    });
```

- ✓ should remove an entry from the model



```
  describe("element removal", function() {
    it("should remove an entry from the model", function() {
      // TODO: write test
      const todo = { id: 42, title: "my todo", completed: true };

      setUpModel([todo]);

      subject.setView("");

      view.trigger("itemRemove", { id: 42 });

      expect(model.remove).toHaveBeenCalledWith(42, jasmine.any(Function));
    });
```

PERFORMANCE OF COMPETITOR SITE

Made all performance analysis on Google Chrome browser installed on Linux machine

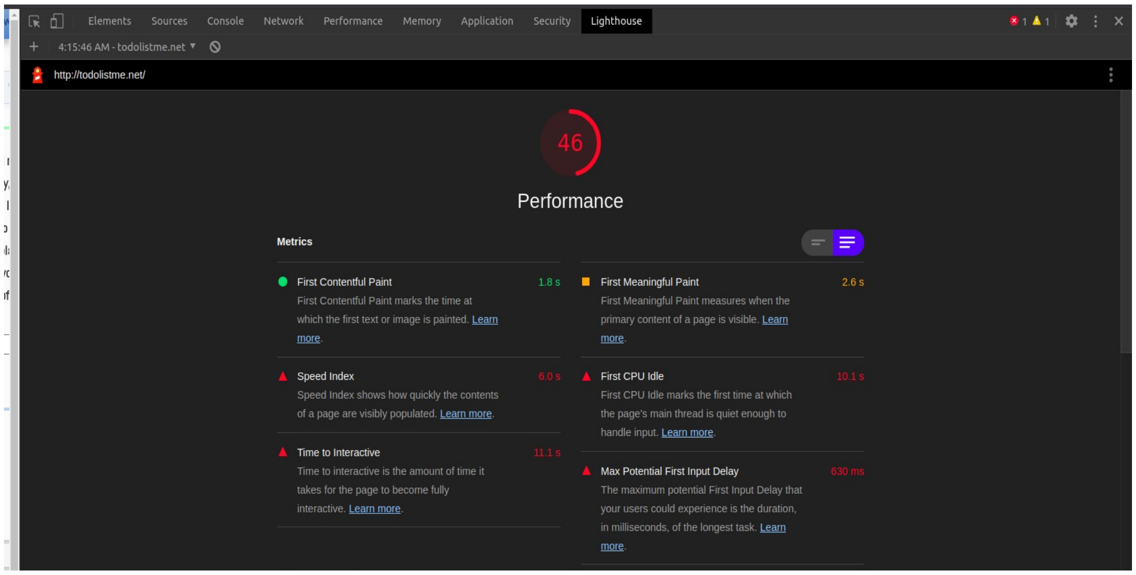
If we take a look of how the competitor website is looks like and we try to analyze its performance, we are going to notice some of the performance issues, so here I'm going to explain in details the real issues that make the website doesn't perform well.

Used the developer tools "DevTools" of the "Chrome Browser", especially the performance and the audit "Lighthouse" tabs, when I analyzed the performance of the application, I figured out the performance percent of it was **46%**, so the application has a lot of performance issues, and I going to outline the reasons behind this bad performance.

One of the major performance issues on the competitor’s app is about Images, the images which are used in the app are in “.PNG” format, so by changing this format to “.JPEG” or “Web P”, we going to be able to reduce the images size and it will save us about **0.38s** according to Audit report of Chrome Browser.

Also the app has a large CSS file that take much time to be rendered on the parse tree, and according to coverage tab In Chrome DevTools, the app has unused CSS rules, so we might use these rules so it could save us around **150ms.**

As we notice that the app has render-blocking resources, these resources are blocking the first paint of the app page, we should eliminate these resources and we are going to save almost **0.26s**



Metrics

● First Contentful Paint	1.8 s	■ First Meaningful Paint	2.6 s
First Contentful Paint marks the time at which the first text or image is painted. Learn more.		First Meaningful Paint measures when the primary content of a page is visible. Learn more.	
▲ Speed Index	6.0 s	▲ First CPU Idle	10.1 s
Speed Index shows how quickly the contents of a page are visibly populated. Learn more.		First CPU Idle marks the first time at which the page's main thread is quiet enough to handle input. Learn more.	
▲ Time to Interactive	11.1 s	▲ Max Potential First Input Delay	630 ms
Time to interactive is the amount of time it takes for the page to become fully interactive. Learn more.		The maximum potential First Input Delay that your users could experience is the duration, in milliseconds, of the longest task. Learn more.	

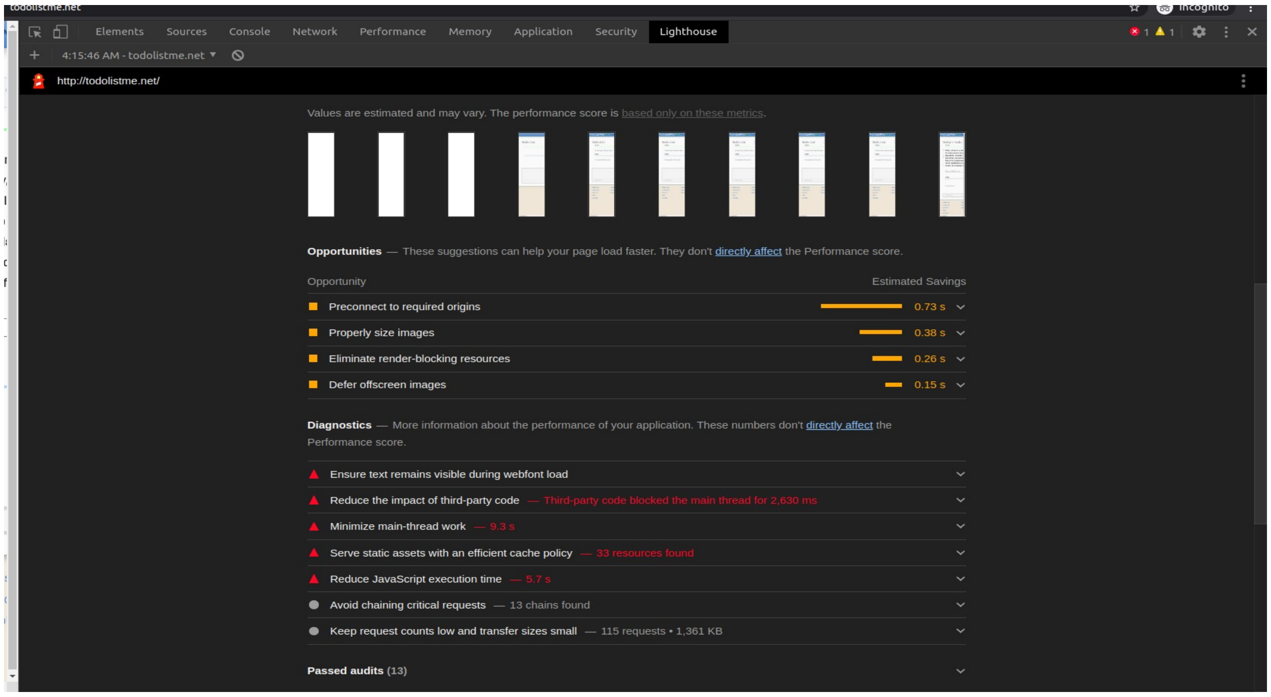
Based on above screenshot of the overall performance percentage of the app, these are some suggestion to solve these performance issues:

- ➔ Leverage the font-display CSS feature to ensure text is user-visible while web fonts are loading. Savings around **440ms**
- ➔ Third-party code can significantly impact load performance. Limit the number of redundant third-party providers and try to load third-party code after your page has primarily finished loading. Third-party code blocked the main thread for **2,630 ms**

➔ Consider reducing the time spent parsing, compiling and executing JavaScript. You may find delivering smaller JS payloads helps with this. Saving around **9.3s**.

➔ Serve static assets with an efficient cache policy - **33** resources found - A long cache lifetime can speed up repeat visits to your page.

➔ Consider reducing the time spent parsing, compiling, and executing JavaScript. You may find delivering smaller JavaScript payloads helps with this. Saving about **5.7s**.



Values are estimated and may vary. The performance score is based only on these metrics.

Opportunities — These suggestions can help your page load faster. They don't directly affect the Performance score.

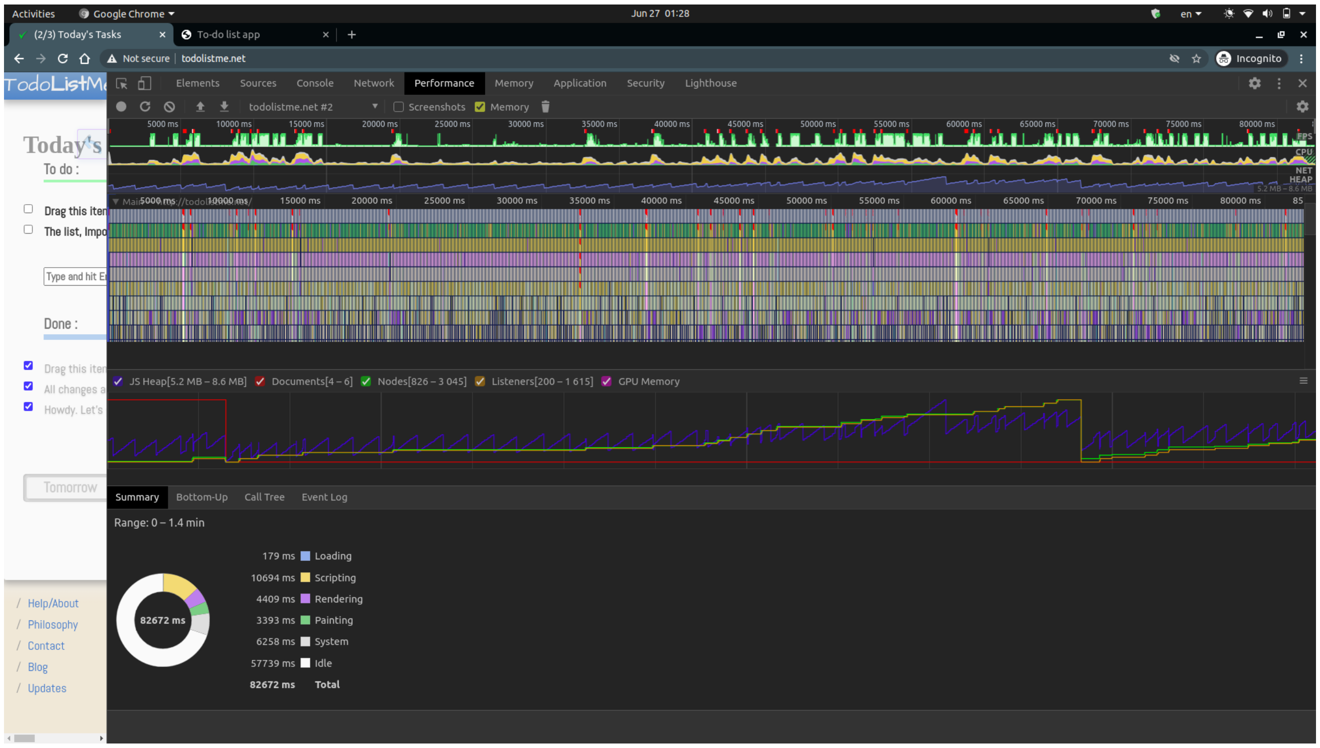
Opportunity	Estimated Savings
Preconnect to required origins	0.73 s
Properly size images	0.38 s
Eliminate render-blocking resources	0.26 s
Defer offscreen images	0.15 s

Diagnostics — More information about the performance of your application. These numbers don't directly affect the Performance score.

- Ensure text remains visible during webfont load
- Reduce the impact of third-party code — Third-party code blocked the main thread for 2,630 ms
- Minimize main-thread work — 9.3 s
- Serve static assets with an efficient cache policy — 33 resources found
- Reduce JavaScript execution time — 5.7 s
- Avoid chaining critical requests — 13 chains found
- Keep request counts low and transfer sizes small — 115 requests • 1,361 KB

Passed audits (13)

Also on Performance tab on Chrome browser, I profiling the app after using the app for 1 minute, and this is a snapshot of the output results:

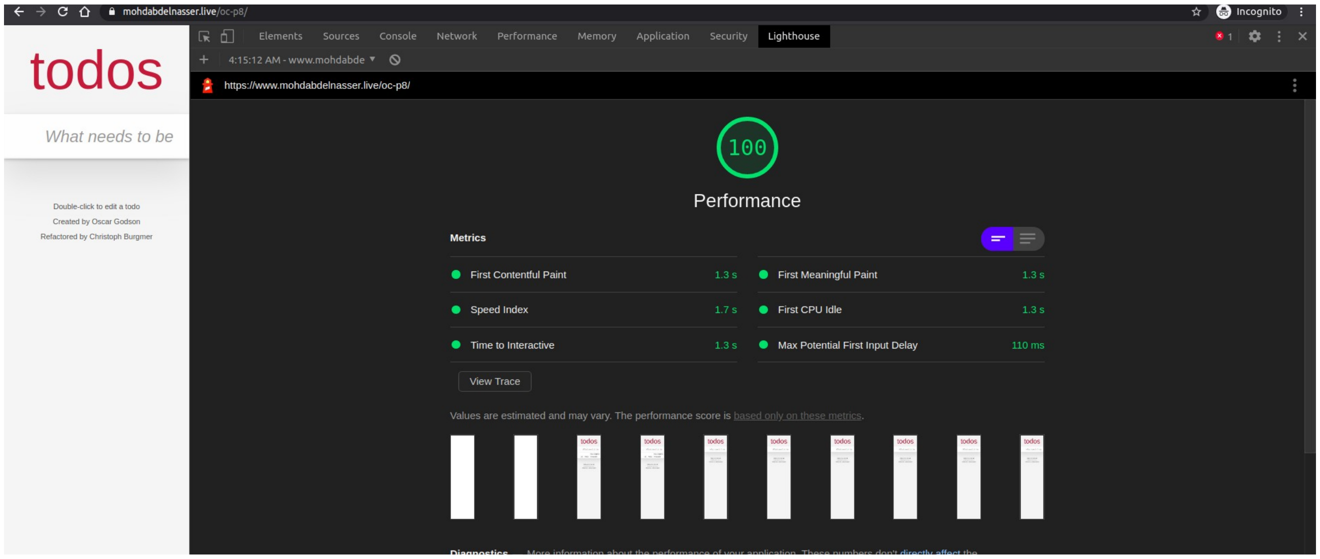


```
Range: 0 – 1.4 min

  179 ms  Loading
10694 ms  Scripting
 4409 ms  Rendering
 3393 ms  Painting
 6258 ms  System
57739 ms  Idle
82672 ms  Total
```

Our To-Do App Performance

If we have a look to measure the performance of our To-Do app, our app is actually pretty good under the performance matrix according to Chrome DevTools, our app is a light and fast app which loads without any sorts of medias, images or videos and it has a better performance in terms of loading speed and time to interactive.


mohdabdelnasser.live/oc-p8/

todos

What needs to be

Double-click to edit a todo
Created by Oscar Godson
Refactored by Christoph Burgmer

https://www.mohdabdelnasser.live/oc-p8/

100

Performance

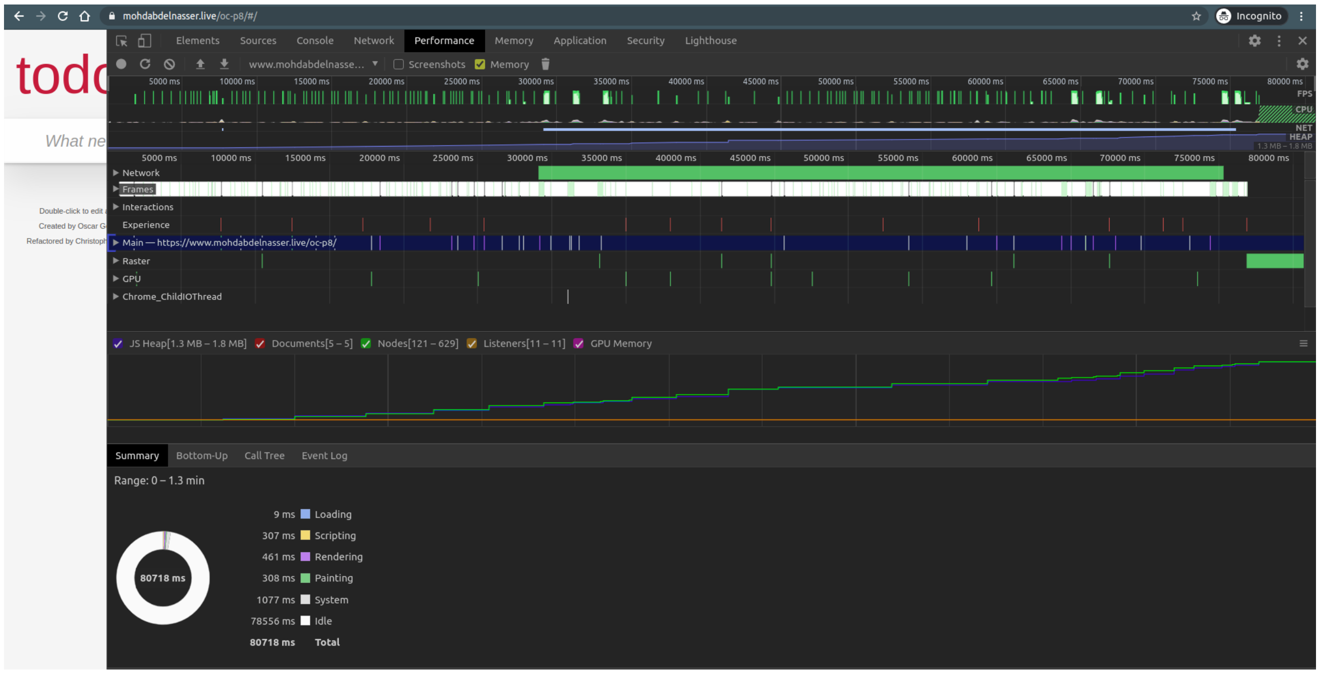
Metrics

First Contentful Paint	1.3 s	First Meaningful Paint	1.3 s
Speed Index	1.7 s	First CPU Idle	1.3 s
Time to Interactive	1.3 s	Max Potential First Input Delay	110 ms

View Trace

Values are estimated and may vary. The performance score is based only on these metrics.

Also on Performance tab on Chrome browser, I profiling the app after using the app for 1 minute, and this is a snapshot of the output results


mohdabdelnasser.live/oc-p8/#/

JS Heap[1.3 MB – 1.8 MB] Documents[5 – 5] Nodes[121 – 629] Listeners[11 – 11] GPU Memory

Range: 0 – 1.3 min

```
    9 ms  Loading
  307 ms  Scripting
  461 ms  Rendering
  308 ms  Painting
 1077 ms  System
78556 ms  Idle
80718 ms  Total
```

SUMMARY

Our app is performing in a very good way and more efficient than the competitor’s app, but on the other hand competitor’s app has more useful functionalities and provides rich user interface and user experience so the user can do more with the specific task. But in the terms of loading time and general performance we do better than competitor app.