**Introduction**

The to-do-list app is a simple and attractive application with a clean, minimalist UI which allows a user to list and manage their pending tasks.

The app provides basic functionality – the ability to list tasks, mark tasks as complete and filter tasks by pending, complete or all. Until such time as a task is deleted from the list, it persists in the user’s local storage.

The app can be viewed and tested here:

<https://clydspyd.github.io/todo-list/>

**Application specifications**

*Overview*

The application is built using the Model-View-Controller (MVC) design pattern and as such separates the programme’s concerns into three interconnected objects -

Model: The central object which contains the underlying logical data

structure of the application

View: A representation of the information contained within the app.

Manipulates the DOM and displays the information to the user.

Controller: Receives user input and connects model and view objects

*File break-down*

Below are listed the files which make up the to-do-list app.

**HTML**

* **Index.html** – Creates the apps basic structure and initial DOM

**CSS**

* **Index.css** – Defines app-specific CSS styles
* **Base.css** – Defines common styles

**JavaScript**

* **Controller.js** – Takes a model and view and acts as the controller between them**.** Contains methods to listen for and handle all user actions.
* **Model.js** – Creates a new Model instance and hooks up the storage. Contains methods for managing app data.
* **View.js** – View that abstracts away the browser's DOM completely. Contains methods for displaying app data to user.
* **App.js** – initiates Model, View and Controller objects, creating an instance of the program
* **Store.js** - Creates a new client-side storage object and will create an empty collection if no collection already exists.
* **Template.js** – Provides template function for displaying items and controlling button states
* **Helpers.js** – Provides a number of auxiliary functions

**Testing report**

The to-do-list app was both manually tested and automatically tested using the Jasmine testing framework. The results of both stages are as follows:

*Manual testing*

Two bugs were identified and corrected during this stage of testing.

* Mis-spelled function declaration contained in **controller.js**

**Erroneous code:**

*Controller*.prototype.adddItem

**Corrected code:**

*Controller*.prototype.addItem

* Random ID generation which runs the risk of repeating IDs, contained in **store.js**

**Erroneous code:**

for (*var* i = 0; i < 6; i++) {

newId += charset.charAt(Math.floor(Math.random() \* charset.length));}

**Corrected code:**

*var* uniqueID = false;

      while (!uniqueID){

for (*var* i = 0; i < 6; i++) {

        newId+=charset.charAt(Math.floor(Math.random()\* charset.length));}

        uniqueID = true;

        for (*var* i = 0; i < todos.length; i++) {

          if (todos[i].id == newId) {

            uniqueID = false;}

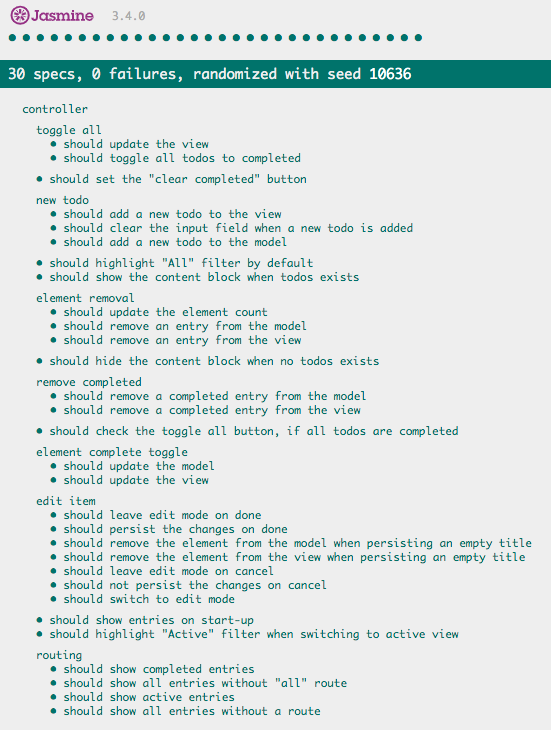
}}

*Jasmine module testing*

In addition to an extensive set of pre-existing tests provided with the application, a number of new tests were written and performed. Details of those tests were as follows:

* should show entries on start-up
* should show active entries
* should show completed entries
* should highlight "All" filter by default
* should highlight ‘Active’ filter when switching to active view
* 'should update the view'
* 'should add a new todo to the model'
* 'should remove an entry from the model'

A full list of Jasmine tests performed, and their results, is shown below:

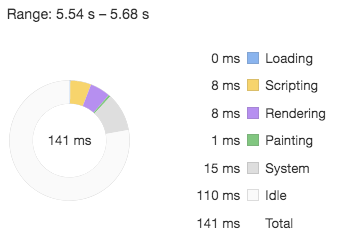
****

**Performance Audit**

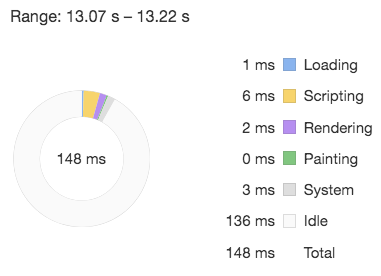
The to-do-list app performed exceedingly well when audited. The performance score obtained was an impressive 98 out of a possible 100. All of the major audit points (such as eliminating render-blocking elements, minimizing main-thread work and JS execution time, not providing unused code etc) were all passed without problems. Our page load is lightning fast, with a time-to-interactive of just 2.2s and a first CPU idle at just 2.1s.

One area which could be approved is our SEO, with a score obtained of 75. In order to address this issue, we should look at our metadata and add more information to allow us to be more visible to crawlers and so more likely to show higher on search results.

**Operation Analysis:**

****

Tasks occupying main thread during the operation of adding a new item to the to-do list

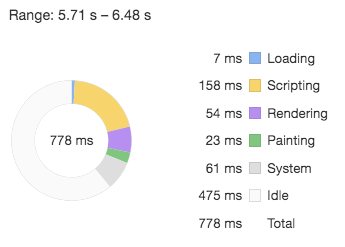


Tasks occupying main thread during the operation of removing an item from the to-do list

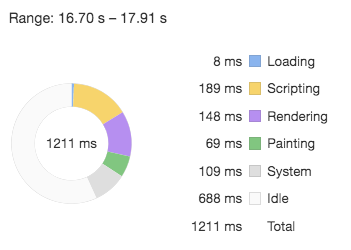
Performance Audit: Competitor Site

Having performed a full performance audit on our competitor site, [todolistme.net](http://todolistme.net/), a number of ways in which we could optimize load time and performance have been identified. The score obtained by the site in said audit was a 47 – less than half of the possible score. By following the steps below, we could expect to raise that number substantially.

**Operation Analysis:**

****

Tasks occupying main thread during the operation of adding a new item to the to-do list



Tasks occupying main thread during the operation of removing an item from the to-do list

**Main opportunities for performance optimization on competitor site:**

Minimize main thread work *(*12.3s total during page loading alone)

The large amount of time spent evaluating (7,842ms alone), parsing and compiling script can be reduced significantly, and by doing so liberate the main thread for other tasks. Numerous JS files are utilizing only a small percentage of the code included in the file (eg jquery-ui.js has 76.2% unused code) and often this unused code is comments. Action necessary: strip out all comments and reduce file size by minifying all JS, HTML and CSS files.

Reduce time to interactive (a massive 11.2s)

JavaScript execution time (8.1s) may be reduced by implementing code splitting (dividing our code into several bundles which can be loaded on demand or in parallel), giving the load time a serious boost by only sending the code needed.

By making use of the ‘font-display’ CSS feature, we can ensure that text remains user-visible while webfonts are loaded from fonts.gstatic.com.

Serve imagesin next-gen formats

4,59ms can be saved by delivering /images/texture.png in an image format such as JPEG 2000, JPEG-XR or WebP due to the better compression offered by these formats.

Improve caching policy

Repeat visit load times can be optimized by the use of an efficient cache policy. Static objects can have their ‘cache-control: max-age’ directive set so that they are cached for one year or more so that on subsequent requests, the browser can use the local copy rather than fetching from the server.

Deliver critical CSSinline

By delivering our critical CSS (elements currently contained within style\_g.css and jquery-ui.css) inline, we eliminate render-blocking resources and shave 960ms off of our first paint.

Reduce criticalrequest chain depth

With a total of 13 chains, the critical path is too long. We must analyze the path and decide which elements are in fact critical for our page load and which elements can be deferred or marked as asynchronous. We must also minimize the download size of all crucial resources.

Comparisson

To-do-list

pros

* Clean, minimalist UI
* Extremely low memory consumption and data transfer
* Well-commented code base utilizing MVC system will assist greatly in any future contribution or development

cons

* Very limited functionality
* Data only stored locally

Competitor site

pros

* Advanced, highly sophisticated functionality
* Remote data storage

Competitor Cons

* High memory consumption and slow load time
* Un-optimized code base (inefficient)
* Google Ads creates a sever drain on memory

Contribute

To report any bugs, suggest improvements or contribute to the future development of the to-do-list app, please feel free to contact the developer through oscarD@todo.me or @OscarGodson. Alternatively, fork our GitHub repo and make a pull request

<https://github.com/ClydSpyd/todo-list>