Technical Documentation of Todo-List-App

Introduction:

Todo-list-app is a very simple application that has the basic features of a task management app. Using this app, users can :

- add new todos
- delete todos
- mark todo as active or completed separately
- mark all todos as active or completed
- display any one of the lists of all, active or completed todos
- clear all completed todos

Live demo - https://todo-openclassrooms-mithhu.netlify.com/

App repository: https://github.com/mithhu/todo-openclassrooms

Technical Specifications:

Our app was implemented by MVC pattern.

1. MODEL
   Model is responsible for managing the data of the application. It receives user input from the controller. It is responsible for CRUD (create, read, update and delete) operations.
2. VIEW
   View is a user interface. View display data using model to the user and also enables them to modify the data
3. CONTROLLER
   Controller listens to user actions from View and calls Model to perform the requested operations. Then calls view to display results from model.

Files and methods:

HTML files:
index.html - application's entry point

CSS files:

Index.css - defines all application CSS styles

Base.css - defines common styles

JS files:

App.js - creates an instance of a program by initiating storage, model, template, view, and controller.

model.js -.Creates a new model instance and hooks up the storage
Methods:

- *create* - Creates a new todo model
- *read* - Finds and returns a model in storage
- *update* - Updates a model
- *remove* - Removes a model from storage
- *removeAll* - Removes all data from storage
- *getCount* - Returns a count of active, completed and total todos

controller.js - Takes a model and view and acts as the controller between them.
Methods:

- *setView* - Loads and initialises the view.
- *showAll* - Will get all items and display them in the todo-list.
- *showActive* - Renders all active tasks.
- *showCompleted* - Renders all completed tasks.
- *addItem* - Adds a new item to the todos list.
- *editItem* - Triggers the item editing mode.
- *editItemSave* - Finishes the item editing mode.
- *editItemCancel* - Cancels the item editing mode.
- *removeItem* - Remove item from the DOM and also remove it from storage.
- *removeCompletedItems* - Will remove all completed items from the DOM and storage.
- *toggleComplete* - Toggles item between completed and active.
- *toggleAll* - Will take all todos and make them complete or active.
- *_updateCount* - Update the number of remaining todos
- *_filter* - Re-filters the todo items, based on the active route.
- *_updateFilterState* - Simply updates the filter nav's selected states.

helpers.js - It works as a helper function. Such as-

- Getting element by CSS selector and attaching event listener to it.
- Attaching a handler to event for all elements that match the selector.
- Finding the element's parent with the given tag name.
- Allowing for looping on nodes by chaining.

store.js - Creates a new client side storage object.
Methods:

- *find* - Finds items based on a query given as a JS object.
- *findAll* - Will retrieve all data from the collection.
- *save* - Will save the given data to the DB.

- *remove* - Will remove an item from the Store based on its ID.
- *drop* - Will drop all storage and start fresh.

template.js - Sets up defaults for all Template methods such as a default template.
Methods:

- *show* - Creates an <li> HTML string and returns it for placement in our app.
- *itemCounter* - Displays a counter of how many to dos are left to complete.
- *clearCompletedButton* - Updates the text within the "Clear completed" button.

view.js - View that abstracts away the browser's DOM completely. It has two simple entry points:

- bind(eventName, handler) - Takes a todo application event and registers the handler.
- render(command, parameterObject) - Renders the given command with the options.

Bugs Fixing:

1. *A bug which does not allow adding new todos to the list because of misspelling "adddItem"*

In **controller.js**

```
Controller.prototype.adddItem = function
```

has been changed into

```
Controller.prototype.addItem = function
```

2. *A bug which may lead to cause conflict between duplicate IDs*

In **store.js**

```
var newId = "";

var charset = "0123456789";

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

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

}
```

has been changed into

```
var newId = Date.now();
```

3. *Console log displayed message when user delete todo (unnessesary code and console log)*

In **controller.js**

Removed the code

```
items.forEach(function(item) {
  if (item.id === id) {
    console.log("Element with ID: " + id + " has been removed.");
  }
);
```

4. *Missing id in input tag for toggle-all label which prevents toggle all todos function to work properly.*

In **index.html**

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

has been changed into

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

Add Tests:

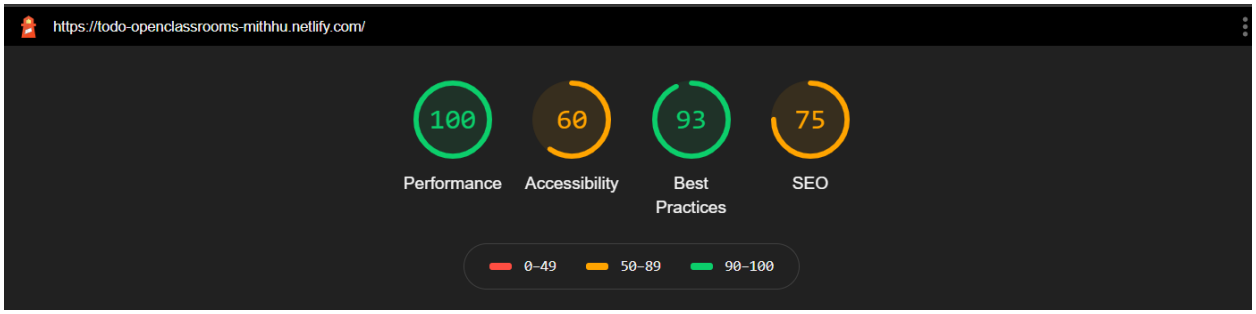
We used Jasmine to create tests. 9 tests have been added to the existing tests:

- 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 the active view
- should toggle all todos to completed
- should update the view
- should add a new todo to the model
- should remove an entry from the model

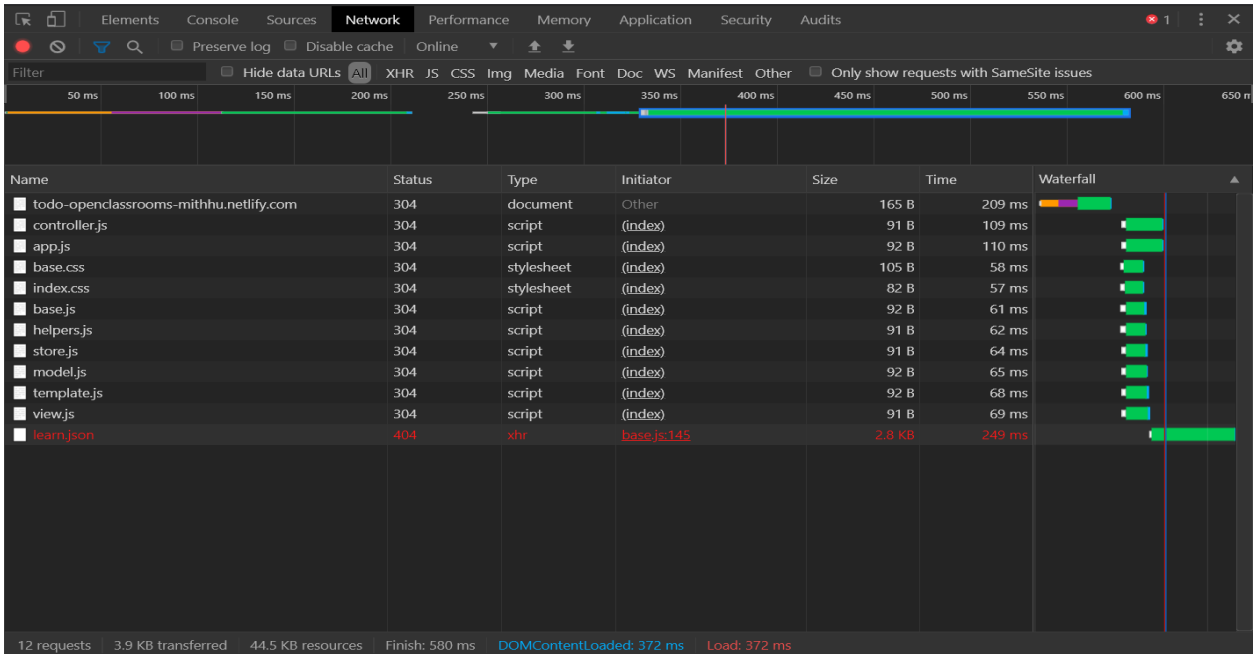
Our todo-list-app's Performance Audit

Browser: Version 80.0.3987.122, OS- Windows 10 (64 bit)

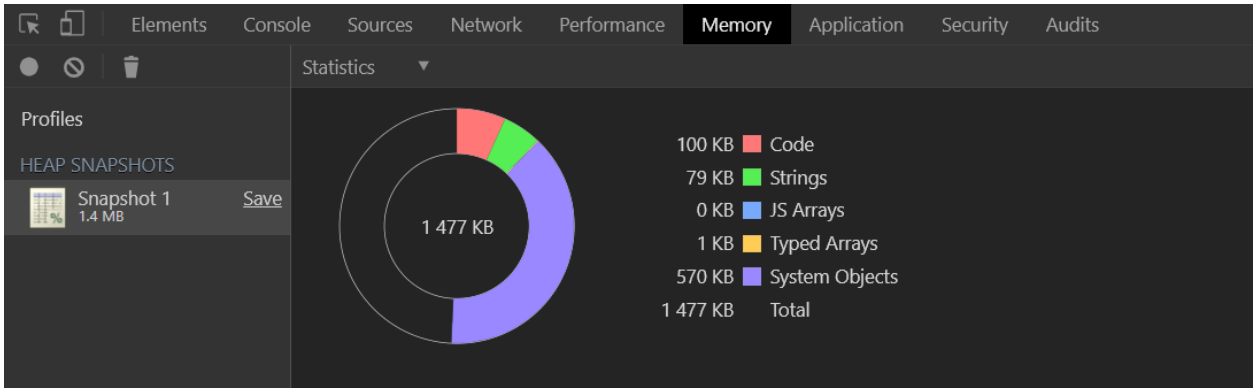
Audit Results:



Loading:

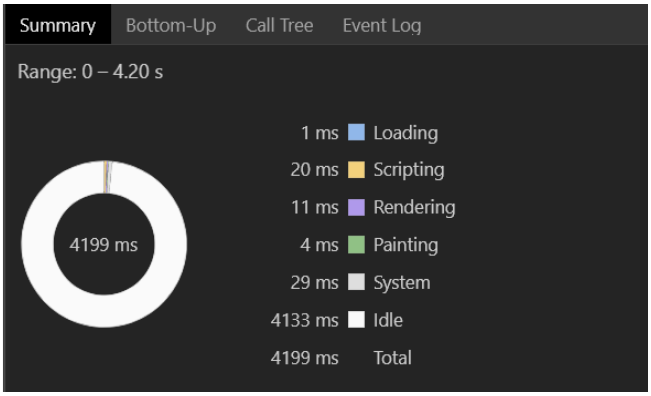


Memory:

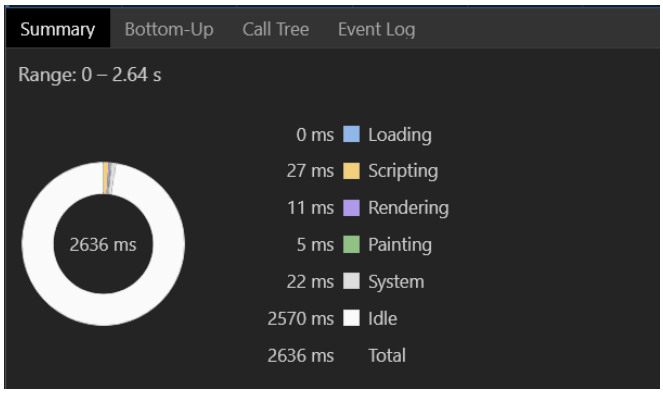


User Actions:

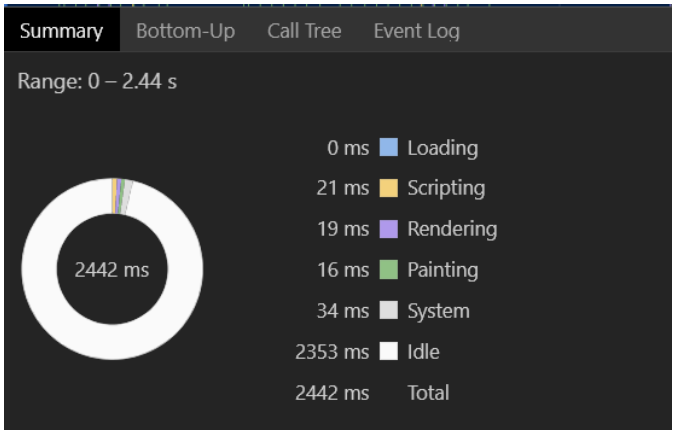
Add a new task:



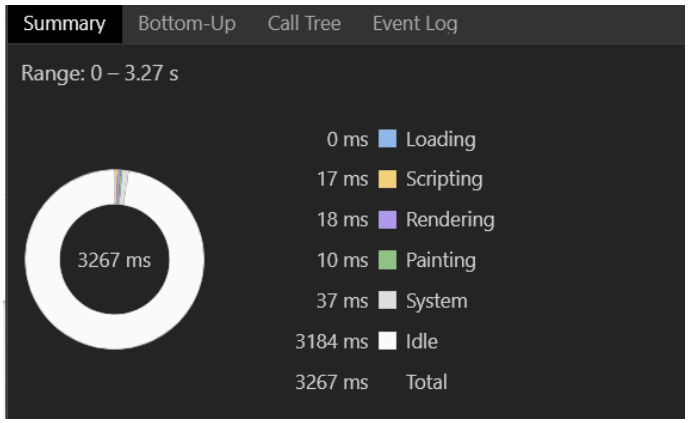
Delete a task:



Make one task completed:



Make all task completed:

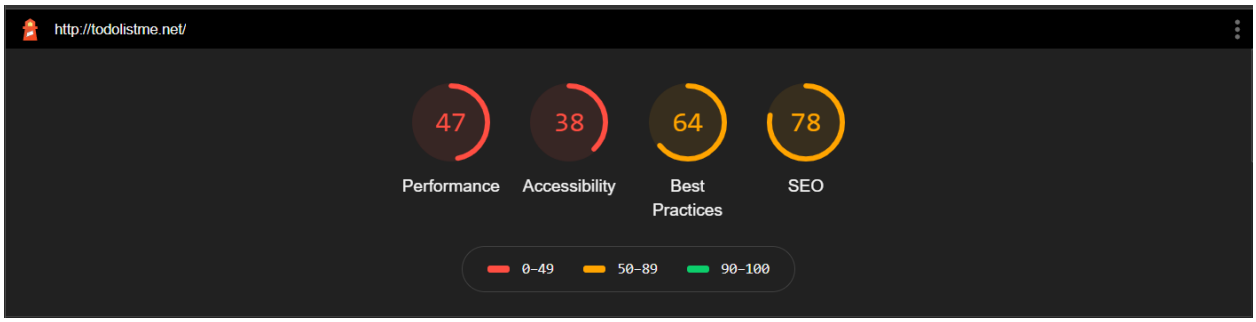


Improvements:

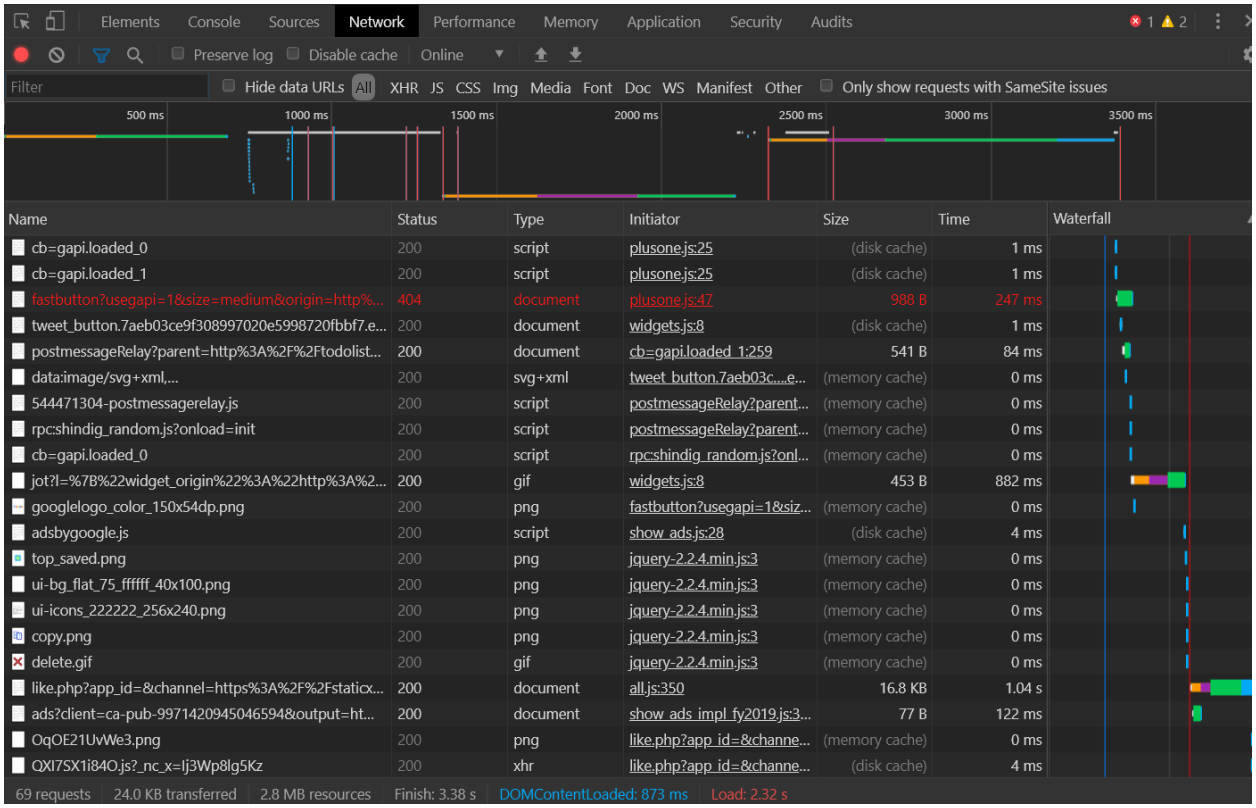
1. To get 100% on Best Practices, we need to remove the base.js file. Because it was failing to load learn.json file and showing error on console.
   [ref- https://github.com/tastejs/todomvc-common/issues/14]
2. Add Label on form elements. Labels ensure that form controls are announced properly by assistive technologies, like screen readers.
3. Add a `<meta name="viewport">` tag to optimize app for mobile screens. Without a viewport meta tag, mobile devices render pages at typical desktop screen widths and then scale the pages down, making them difficult to read.
4. Add a <meta name=description> element to the <head> of each of your pages. A high-quality, unique meta description makes your page appear more relevant and can increase your search traffic.

Competitor's App Performance Audit:

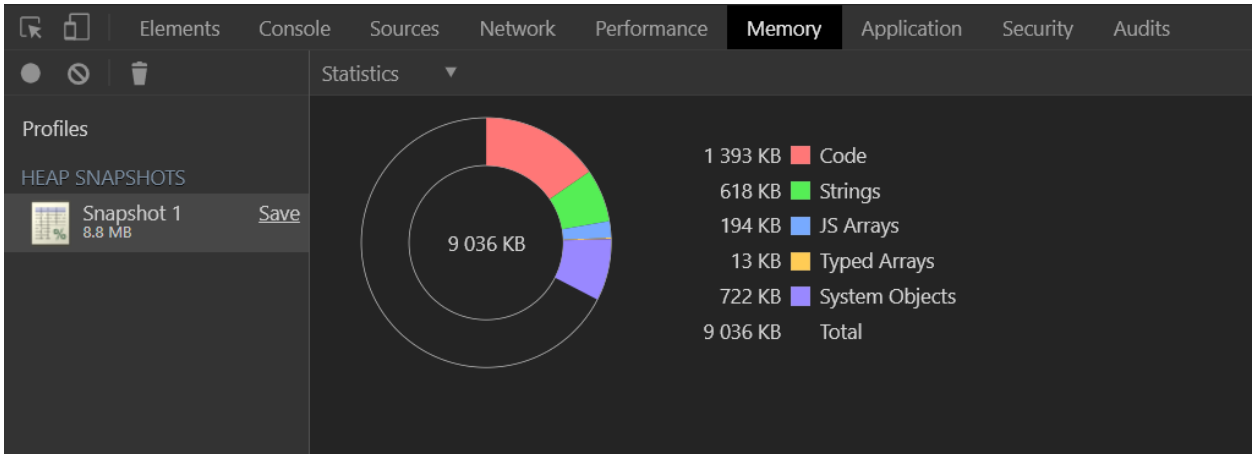
Audit Results:



Loading:

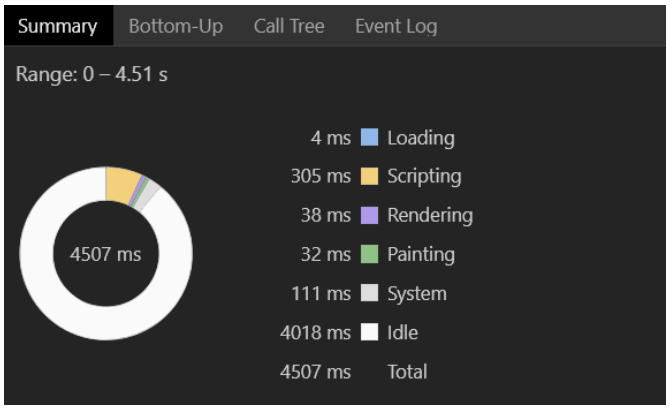


Memory:

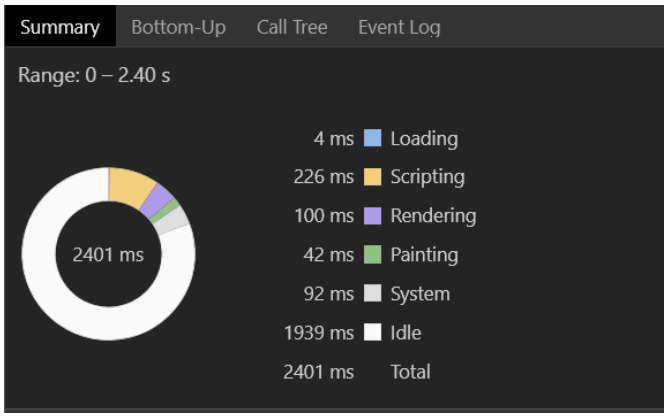


User actions:

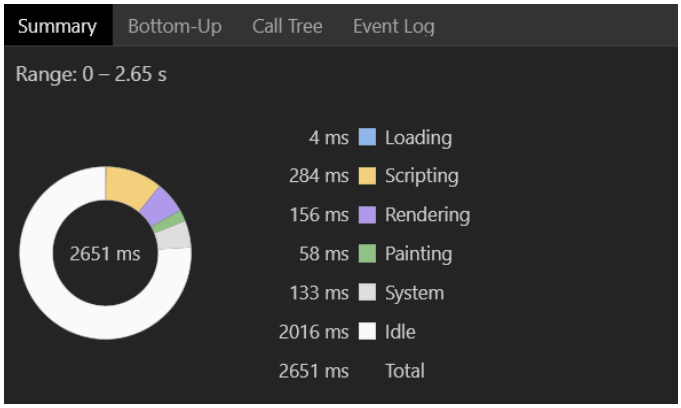
Add a new task:



Delete one task:



Make one task completed:



Make all task completed:

No option to make all tasks completed.

Improvements:

1. Consider adding preconnect or dns-prefetch resource hints to establish early connections to important third-party origins.
   <link rel="preconnect"> informs the browser that your page intends to establish a connection to another origin, and that you'd like the process to start as soon as possible.
2. Serve images that are appropriately-sized to save cellular data and improve load time.
3. Fonts are often large files that take a while to load. Some browsers hide text until the font loads. The easiest way to avoid showing invisible text while custom fonts load is to temporarily show a system font.
4. 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
5. HTTP caching can speed up your page load time on repeat visits.
6. Consider reducing the time spent parsing, compiling and executing JS. You may find delivering smaller JS payloads helps with this
7. Background and foreground colors do not have a sufficient contrast ratio. Low-contrast text is difficult or impossible for many users to read.
8. [id] attributes on the page are not unique.
9. <frame> or <iframe> elements do not have a title. Screen reader users rely on frame titles to describe the contents of frames.
10. Image elements do not have [alt] attributes. Informative elements should aim for short, descriptive alternate text.
11. Form elements do not have associated labels. Labels ensure that form controls are announced properly by assistive technologies, like screen readers
12. <html> element does not have a [lang] attribute. If a page doesn't specify a lang attribute, a screen reader assumes that the page is in the default language that the user

chose when setting up the screen reader. If the page isn't actually in the default language, then the screen reader might not announce the page's text correctly.

13. Does not use HTTPS. All sites should be protected with HTTPS, even ones that don't handle sensitive data. HTTPS prevents intruders from tampering with or passively listening in on the communications between your app and your users.
14. Uses document.write(). For users on slow connections, external scripts dynamically injected via document.write() can delay page load by tens of seconds.
15. Includes front-end JavaScript libraries with known security vulnerabilities. Some third-party scripts may contain known security vulnerabilities that are easily identified and exploited by attackers. Library Version - jQuery@2.2.4 / Vulnerability Count - 2.
16. Does not have a <meta name="viewport"> tag with width or initial-scale. Add a viewport meta tag to optimize your app for mobile screens.
17. Image elements should use alt attributes

Comparison:

Topic	Description	Our App	Competitor's App
Loading	Page Loading time	372 ms	2320 ms
	DOM content Loaded time	372ms	3380 ms
	Total amount of memory used	44.5 KB	2.8 MB
	Amount of Data transferred	3.9 KB	24 KB
	No of requests sent	12	69
User Actions	Add a new task to list	4199 ms	4507 ms
	Remove a new task from list	2636 ms	2401 ms
	Mark one task as completed	2442 ms	2651 ms
	Mark all task as completed	3267	No option

Our app:

Advantages:

1. Short loading time
2. Low memory usage
3. Short time of performing functionalities
4. Clean, readable, and commented code
5. Simple and minimal design
6. Can select and unselect all the task in one click

Disadvantages:

1. Limited functionality
2. Local storage is used to save data.
3. Data can't be synced

Competitor's app:

Advantages:

1. Tasks can be sorted (Normal, Alphabetical, Random and Top 3)
2. Tasklist can be printed
3. Drag and drop tasks option available
4. User can create multiple lists
5. User can create categories
6. Data can be synced

Disadvantages

1. Long loading time
2. High memory usage
3. Long time of performing functionalities
4. No option to select and unselect all the task in one click

Conclusion:

Todo-list-app is way more performant comparing to our competitor's app. We can make this app more efficient by using dedicated CSS and adding a storage system. From this project, I have learned about MVC approach and writing tests which will help me to work on large projects.