‘todo-list-app’

About the app

‘todo-list-app’ is an online application created to help user manage their daily tasks. Simple functionality let user add new tasks to the list, mark them as completed when done (either separately or all of them at the same time). User can navigate between ‘All’, ‘Completed’ or ‘Active’ tasks.
App uses MVC architecture.

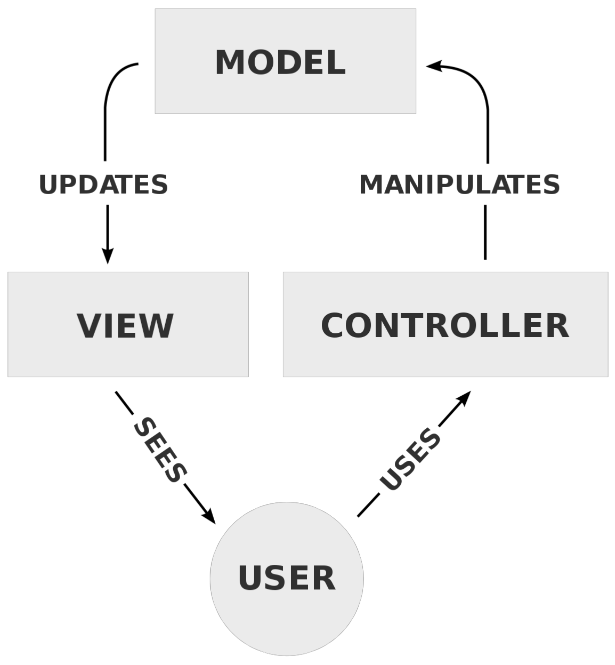
About the MVC

MVC is an architectural pattern used to develop user interfaces that divides an application into three interconnected parts. (model - view - controller)

Model - centre component of the pattern, is responsible for managing the data of the application.
It receives user input from the controller.

View - presentation of the model in a particular format

Controller - responds to the user input and performs interactions on the data model objects.



Files in the JS folder

Here is the list of files in the app's folder

app.js - sets up new list. This file is the starting point of the entire application

controller.js - have a control between view and model.

Here are the methods from the file
• setView - loads and initialises the view
• showAll - displays list on tasks
• showActive - displays all active tasks
• showCompleted - displays all completed tasks
• addItem - adds new items by handling the DOM
• editItem - triggers editing mode
• editItemSave - finishes editing mode
• editItemCancel - cancels the editing mode
• removeItem - removes item from the DOM and from the storage
• removeCompletedItems - removes all completed items from DOM and the storage
• toggleComplete - gives the ID and updates the state in the storage
• toggleAll - update the state in storage of all items
• _updateCount -
• _filter -
• _updateFilterState -

helpers.js - contains some methods, deals with DOM
It gets element by CSS selectors and by tag names, deals with addEventListeners,

model.js - abstracts how single 'todo' item is stored in the storage. Provides CRUD methods to manage data.

Here are the methods from the file
• create - creates new todo model
• read - reads data from storage
• update - updates a model by giving it an ID
• remove - removes a model from storage
• removeAll - removes all data from storage
• getCount - returns a count of all todos

store.js - creates a new client side storage object by use of Store function.

Here are the methods from the file
• find - finds items based on a query given as a JS objects
• findAll - retrieves all data
• save - saves given data to the database
• remove - removes an item from the Store based on its ID
• drop - drops all storage and start fresh

view.js - manipulates DOM structure
Its has two entry points
• **Bind** ('newTodo', 'removeCompleted', 'toggleAll', 'itemEdit', 'itemRemove', 'itemEditDone', 'itemEditCancel')
• **Render** ('showEntries', 'removeItem', 'updateElementCount', 'clearCompletedButton', 'contentBlockVisibility', 'toggleAll', 'setFilter', 'clearNewTodo', 'elementComplete', 'editItem', 'editItemDone')

template.js - template function to display items, change states etc.
Provides HTML templates which are used by the view

Debugging

Bugs fixed

- **Typo in controller.js on line 95**

```
Controller.prototype.adddItem = function(title) {
    var self = this;
    if (title.trim() === "") {
      return;
    }
    self.model.create(title, function() {
      self.view.render("clearNewTodo");
      self._filter(true);
    });
  };
```

has been changed to

```
Controller.prototype.addItem = function(title) {
    var self = this;
    if (title.trim() === "") {
      return;
    }
    self.model.create(title, function() {
      self.view.render("clearNewTodo");
      self._filter(true);
    });
  };
```

- **Lack of label to the input in the index.html on line 20**

```
 <input class=“toggle-all" type="checkbox" />
        <label for="toggle-all">Mark all as complete</label>
```

has been changed to

```
<input class="toggle-all" id="toggle-all" type="checkbox" />
        <label for="toggle-all">Mark all as complete</label>
```

- **Useless logging statement in controller.js on line 167**

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

- **Potential conflict between duplicate IDs - store.js on line 87**

```
 // Generate an ID
    var newId = “”;
    var charset = "0123456789";
```

has been changed to

```
 // Generate an ID
    var newId = Date.now();
    var charset = "0123456789";
```

Jasmine tests

- **should show entries on start-up**

```
it("should show entries on start-up", function() {
   var todo = { title: "my todo" };
   setUpModel([todo]);
   subject.setView("#/");
   expect(view.render).toHaveBeenCalledWith("showEntries", [todo]);
 });
```

- **should show active entries**

```
it("should show active entries", function() {
     var todo = { title: "my todo", completed: false };
     setUpModel([todo]);
     subject.setView("#/active");
     expect(view.render).toHaveBeenCalledWith("setFilter", "active");
   });
```

- **should show completed entries**

```
it("should show completed entries", function() {
     var todo = { title: "my todo", completed: true };
     setUpModel([todo]);
     subject.setView("#/completed");
     expect(view.render).toHaveBeenCalledWith("setFilter", "completed");
   });
 });
```

- **should highlight "All" filter by default**

```
it('should highlight "All" filter by default', function() {
    var todo = { title: "my todo" };
    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(){
    var todos = [
      { title: "my todo", completed: false },
      { title: "my todo2", completed: true }
    ];
    setUpModel(todos);
    subject.setView("#/active");
    expect(view.render).toHaveBeenCalledWith("setFilter", "active");
  });
```

- **should toggle all todos to completed**

```
 describe("toggle all", function() {
    it("should toggle all todos to completed", function() {
      var todos = [
        { title: "my todo", completed: false },
        { title: "my todo2", completed: true },
        { title: "my todo3", completed: false }
      ];
      setUpModel(todos);
      subject.setView("#/completed");
      expect(view.render).toHaveBeenCalledWith("setFilter", "completed");
    });
```

- **should update the view**

```
 it("should update the view", function() {
      var todo = { id: 1, title: "my todo", completed: false };
      setUpModel([todo]);
      subject.setView("");
      expect(view.render).toHaveBeenCalledWith("updateElementCount", 1);
    });
  });
```

- **should add a new todo to the model**

```
describe("new todo", function() {
    it("should add a new todo to the model", function() {
      var newTodo = {
        title: "my todo",
        completed: false,
        checked: false,
        visible: false
      };
      setUpModel([newTodo]);
      subject.setView("");
      expect(view.render).toHaveBeenCalledWith("updateElementCount", 1);
      expect(view.render).toHaveBeenCalledWith("showEntries", [
        Object({
          title: "my todo",
          completed: false,
          checked: false,
          visible: false
        })
      ]);
    });
```

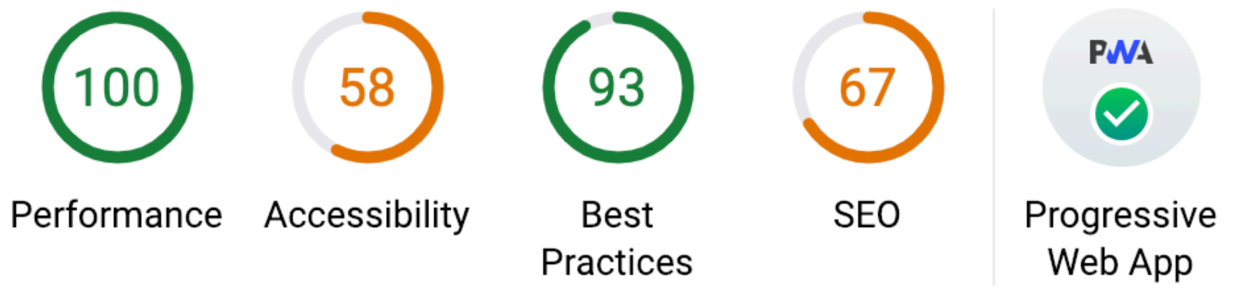
- **should remove an entry from the model**

```
describe("element removal", function() {
    it("should remove an entry from the model", function() {
      var todo = { id: 42, title: "my todo", completed: true };
      setUpModel([todo]);
      subject.setView("#/");

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

Audits

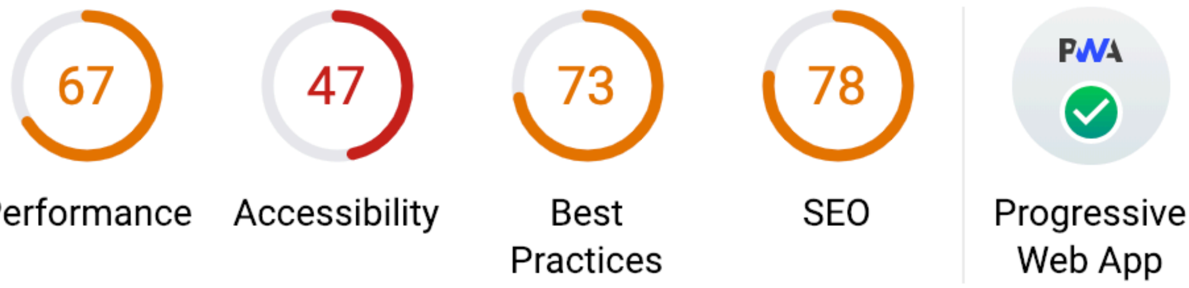
Todo-list-app



Cons / areas of improvement
Does not respond with a 200 when offline
No viewport meta tag found
Address bar does not match brand colours
Form elements do not have associated labels
Browser errors were logged to the console

Pro
Page load is fast enough on 3G
Uses HTTPS
Server response times are low (TTFB)
Uses efficient cache policy on static assets
Avoids enormous network payloads
JavaScript execution time (0.1s)
Uses HTTP/2 for its own resources
Links to cross-origin destinations are safe

Todolistme



Cons / areas of improvement
JavaScript execution time (5.0s)
Does not respond with a 200 when offline
Does not use HTTPS
Does not redirect HTTP traffic to HTTPS
No viewport meta tag found
Background and foreground colours do not have a sufficient contrast ratio.
[id] attributes on the page are not unique
Form elements do not have associated labels
<html> element does not have a [lang] attribute
Image elements do not have [alt] attributes

Pro
Links to cross-origin destinations are safe
Data can be saved
Has a lot option for user (sorting tasks, creating new lists)

Summary

	Todo-list-app	Todolistme
PERFORMANCE	100%	67%
Time to interactive	1.0s	7.7s
Estimated input latency	10ms	370ms
ACCESSIBILITY	58%	47%
BEST PRACTICES	93%	73%
SEO	67%	78%

Comparing those two apps 'todo-list-app' works much quicker, has a simple layout and better performance than 'todolistme' but its very limited in the functionality. 'todolistme' offers more in regards of adding new task, gives the option to sort tasks in four different ways. User is able to create new lists. 'todolistme' is much slower because of the google adds displayed on the page. Both apps could use a service worker so that the apps can work offline.