Project: Todos-list-app

Technical documentation

Author: Aleksandr Gribko

**1. Project description**

**1.1 Architecture**

The project is based on the MVC architecture.

MVC stands for Model-View-Controller and is a programming pattern that separates this free elements of application:

- Controller with help of which a user manipulates the model,

- Model, once being manipulated, updates the view,

- View, what a user sees after each manipulation.

MVC is being used to make each element independent and reusable.

* + 1. **Controller**

It takes the model instance and the view instance and acts as the controller between them.

The controller loads and initiates the view with possible parameters ‘ ‘, ‘active’ or ‘completed’.

Also in the controller all actions on the todos list are designated to be triggered on user’s manipulations on the program (e.g. addItem, editItem, removeItem, toggleAll).

* + 1. **Model**

In the model element a new todo is created and stored with a particular ID.

Methods:

*.read* finds a model in storage to make action on it.

*.update* updates a model by giving it an ID, data to update, and a callback to fire when the update is complete.

*. remove* removes a model from storage (even *.removeAll*).

*. getCount* returns numbers of active, completed and total todos to be used in the view.

* + 1. **View**

The core of the view element is the render method that takes as

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

* + 1. **Additional elements**

The project has additional files that stores necessary elements and connectors.

*App.js* sets up a new todo list.

*Template.js* stores the template for every todo.

*Store.js* Creates a new client side storage object and will create an empty collection if no collection already exists.

*Helpers.js* has a list of wrappers: event listeners, query selectors.

**1.2 Bugs**

During analysis two bugs have been found.

1. A typo.

Line 95 in controller.js had a typo of triple ‘d’:

*‘Controller.prototype.adddItem = function (title) {…}*

1. Potential conflict between duplicate IDs.

Line 77 in store.js is the function that stores a todo with a random ID. An ID is generated as a random number of 6 digits. There is a little possibility, that an ID will be duplicated. To prevent this I have added a Boolean ‘*checkForDuplicate’* false by default. Then by use of *do{…} while* *(checkForDuplicate)* loop and *for..of* loop I am checking if a random ID hasn’t been already assigned.

*let checkForDuplicate = false*

*do{*

*checkForDuplicate = false*

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

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

*}*

*for (let todo of todos){*

*if(todo.id === newId){*

*newId = ''*

*checkForDuplicate = true*

*break*

*}*

*}*

*} while (checkForDuplicate)*

3) Incorrect display of todos’ check.

In the index.css lines 192 and 196 hex color system had to be changed to rgb to work correctly.

**1.3 Tests**

In order to check the correct f of JavaScript in the Project behavior-driven development framework Jasmine has been used.

Additional to already existing tests some additional text where written in order to check that:

|  |  |
| --- | --- |
| * Entries are shown on start-up, | var todo = { title: "my todo" }; *//create a new todo*  setUpModel([todo]); *// create new model with the todo*  subject.setView(""); *//set view at the starting page*  expect(view.render).toHaveBeenCalledWith("showEntries", [todo]); *// expect to see all entries* |
| * Active entries are shown, | setUpModel([{ title: "my todo, fompleted:false" }]); *// create new model with a new todo*  subject.setView("#/active"); *//set view at the ‘active’ page*  expect(view.render).toHaveBeenCalledWith('setFilter', 'active'); *// expect to see active entries* |
| * Completed entries are shown, | var todo = { title: "my todo" , completed:true }; *//create a new todo*  setUpModel([todo]); *// create new model with the todo*  subject.setView("#/completed "); *//set view at the ‘completed’ page*  expect(view.render).toHaveBeenCalledWith('setFilter', 'completed'); *// expect to see completed entries* |
| * Filter “All” is highlighted by default, | setUpModel([{ title: "my todo" }]); *// create new model with a new todo*  subject.setView(""); *// set view at the ‘default’ page*  expect(view.render).toHaveBeenCalledWith('setFilter', ''); *// expect to see unfiltered entries*  expect(view.render).toHaveBeenCalledWith('contentBlockVisibility', {visible: true}); *// expect content to be visible* |
| * Filter “Active” is highlighted when switching to active view, | setUpModel([{ title: "my todo" }]); *// create new model with a new todo*  subject.setView("#/active"); *//set view at the ‘active’ page*  expect(view.render).toHaveBeenCalledWith('setFilter', 'active'); *// expect to see active entries* |
| * All todos are toggled to completed, | setUpModel([{ id: 1, title: "my todo", completed: false }]); *// create new model with a new todo (uncompleted)*  subject.setView(""); *//set view at the ‘default’ page*  subject.toggleAll(true); *// toggle all entries*  expect(view.render).toHaveBeenCalledWith('elementComplete', {id:1, completed:true}); *// expect to see the todo as completed* |
| * The view is updated when toggle all, | setUpModel([{ title: "my todo", completed: false }]); *// create new model with a new todo (uncompleted)*  subject.setView(''); *//set view at the ‘default’ page*  subject.toggleAll(true); *// toggle all entries*  expect(view.render).toHaveBeenCalled();*// expect to update the view* |
| * A new todo should be added to the model, | setUpModel([]) *// create new empty model*  subject.setView("") *//set view at the ‘default’ page*  subject.addItem('a new todo'); *// add a new todo*  expect(model.create).toHaveBeenCalledWith("a new todo", jasmine.any(Function)); *// model of the todo should be created* |
| * An entry should be removed from the model, | setUpModel([{ id: 42, title: "my todo", completed: true }]); *// create new model with a new todo*  subject.setView(''); *//set view at the ‘default’ page*  subject.removeItem(42) *// remove the todo*  expect(model.remove).toHaveBeenCalledWith(42, jasmine.any(Function)) *// expect the todo to be removed* |
| * A completed entry should be removed from the model | setUpModel([{ id: 42, title: "my todo", completed: true }]); *// create new model with a new todo (completed)*  subject.setView(""); *//set view at the ‘default’ page*  view.trigger("removeCompleted"); *// remove completed clicked*  expect(model.read).toHaveBeenCalledWith(  { completed: true }, jasmine.any(Function)); *// has to search for completed todos*  expect(model.remove).toHaveBeenCalledWith(42, jasmine.any(Function)); *// has to remove completed todo* |

30 existing tests bring to 0 failures.

**2. Audit**

First of all, the program is written in a difficult too detailed way using a new function and many callbacks for each little element. It results in a callback hell where in order to understand the structure of the program we have to go thought many functions and even separate files. What is designed in the end is a simple todo list with create-store-read-update-destroy functionalities.

In order to estimate the site performance we take a website of a competitor and make a comparison table.

|  |  |  |
| --- | --- | --- |
|  | Competitor: todolistme.net | We: todos-list-app |
| **Initial load** | | |
| Requests | 93-112 (66 without ads) | 13 |
| type | JS,  CSS,  PNG (page pic elements)  DOC (FB like and tweet button)  DOC (Google ads, Google analytics)  Fonts  Code (jQuery) | JS,  CSS |
| **Fast connections 4G** | | |
| DOM content load | 1.242 – 1.8 s (1.6 s without ads) | 362 ms |
| Load all materials | 2.573 – 3.5 s (3.08 s without ads) | 501 ms |
| **Slow connections 3G** | | |
| DOM content load | 18.98 s (15.05 s without ads) | 4.55 s |
| Load all materials | 30.67 s (26.16 s without ads) | 4.66 s |
| **Interacting with the site** | | |
| Add new task | Event handler (key press) – 64.84 ms;  Animation – 555 ms | Event handler (key press) – 82.8 ms;  No animation |
| Task done | Event handler (click) – 37.80 ms;  Animation 872 ms (disappear + appear in ‘done’) | Event handler (Tap) – 192 ms;  Animation 408 ms (disappear) |
| **Audit** | | |
| Overall result | Performance  53 / 65 (with/no ad) | Performance  100 |
|  | First Contentful Paint  1.4 s / 1.5 s  Speed Index  5.7 s / 5.1 s  Time to Interactive  9.5 s / 7.4 s  First Meaningful Paint  2.4 s / 1.8 s  First CPU Idle  8.5 s / 6.9 s  Estimated Input Latency  240 ms / 200 ms | First Contentful Paint  1.1 s  Speed Index  1.5 s  Time to Interactive  2.1 s  First Meaningful Paint  1.1 s  First CPU Idle  1.9 s  Estimated Input Latency  240 ms |
|  | | |
| Recommendations for better performance | - Change PNG to JPEG  - Improve render-blocking code  - Minify JS and CSS (delete comments, make inline)  - Don’t use external jQuery | - defer/async js files,  - Minify JS and CSS (delete comments, make inline) |

**Performance conclusion**

Competitor: todolistme.net is much slower than todos-list-app and the performance is falling significantly with slow connections. The main reasons for it are heavy pictures, advertisement and external jQuery.

Todos-list-app looks much more competitive from performance point of view. Its visual simplicity and lack of distracting elements makes UI very intuitive and clear. Some functionalities that take resources but do not bring much sense for such this kind of programs are FB like and tweet button.

Todos-list-app Estimated Input Latency is assessed as 240 ms that is quite long.

The main problem with Todos-list-app is Estimated Input Latency that had been assessed as 240 ms. To fix this Critical Requests Depth (defer/async, minimize js files) and main-thread work (reducing the time spent parsing, compiling and executing JS) has to be minimized. Script Parsing & Compilation took 731 ms.

When scaling up the app we should analyze the possible effect of the new features on the rendering. First, all scripts and materials that doesn’t play essential role in the first page load can be loaded asynchronously. Style sheets can be stored in separate files for different media. JS and CSS files in the production stage can be made inline. Media materials should be in the appropriate format and minimum necessary size.