OPENCLASSROOMS

Student Name: Yehia Tarek

Submitted To: Harish

Mentored By: Manish Podival

Report:

Enhance an existing project

Agenda

Introduction

detailed description of the code structure

Fixed Bugs

Test Cases

Performance

Introduction:

todo-List is a web app application that help you to increase your productivity you could add your tasks organize them and update them.You could list what incomplete and what is already completed. This easy to do with the simple and easy user interface.
The Application created using vanilla javascript and use MVC architectural design pattern.It's design Model-viewer-Controller that separate view from model form controllers. To make the code more clear easier to understand and use.
MVC has been used here to add dynamical functionality: user can add new task without reloading the webpage (like Single Page Application – SPA) similar to ajax functionality when we use server-side.

MVC Components are:
Model as a central component manages data logic and methods. Here are created prototypes to manage a local storage object and main app functionality like adding, updating and deleting task of the list.

View as an output represent or displays and manages the user interactions with the application, manipulates DOM structure and represents its functionality.

Controller as a third part of the MVC pattern connects model and view by converting inputs from the View for the Model component.

Detailed description of the code structure

Controller Object– controls interactions between Model and View.

Parameters: model object and view object.

Prototypes:

- setView (loads and initialize the view),
- showAll (displays all items in the todo-list),
- showActive (renders uncompleted tasks),
- showComplited(renders completed tasks),
- addItem (creates new todo task, saving it in the local storage by adding ID),
- editItem (starts editing mode of todo task by matching with the correct ID),
- editItemSave (successfully edits item and save the changing by using matched ID),
- editItemCancel (cancels the item editing mode),
- removeItem (removes item from to-do-list and storage by using its ID as a parameter),
- removeCompletedItems (removes all completed tasks),
- toggleComplete (gives ID and updates the state of completeness of task in the storage),
- toggleAll (change the state of completeness of the tasks: on/off),

Model Object– creates new Model instance and connects it with the storage.

Parameters: storage object.

Prototypes:
- create (creates a new todo model and saves it in the storage),
- read (finds and returns a model in storage, if the query isn't given, returns everything),
- update (updates a model, every action based on unique ID),
- remove (removes a model from storage),
- removeAll (removes all data from storage),
- getCount (counting active, completed and total tasks by finding the in the storage).

View Object – manipulates DOM structures attached to user interaction. It has two simple entry points:
- bind (takes a todo application event and registers the handler),
- render (renders the given command with the options).

#Storage Object – manages data storage by using the local session storage.

#Helpers - a bunch of helper methods for querying the selectors and encapsulating the DOM.

#Template – delivers template function to display list items, change button states, escape characters.

Fixed Bugs:

1- syntax error on todo-list-project/todo-list-app/js/controller.js line 95
2- missing ID on line 16 <input class="toggle-all" id="toggle-all" type="checkbox">
index.html

Test Cases:

1-'Should show entries on start-up'
the 'todo' array should be empty, when the application starts

```
    it('should show entries on start-up', function () {
        var todo = {title: 'my todo'};
        var todo2 = {title: 'my todo2'};
        var todo3 = {title: 'my todo3'};
        setUpModel([todo, todo2, todo3 ]);

        subject.setView('#/');

        expect(model.read).toHaveBeenCalled();

        expect(view.render).toHaveBeenCalledWith('showEntries', [
            {title: 'my todo'},
            {title: 'my todo2'},
            {title: 'my todo3'}
            ]);
    });
```

2- ‘should show active entries'
the completed tasks which are set to false (completed = false);

```
    it('should show active entries', function () {
        var todoArray = [
            {title: 'my todo', completed: true},
            {title: 'my todo2', completed: false},
            {title: 'my todo3', completed: true}
        ];
        setUpModel(todoArray);
        subject.setView('');

        expect(view.render).toHaveBeenCalledWith( 'toggleAll', Object({ checked: false }));
    });
```

3-'should show completed entries'
the completed tasks which are set to true (completed = true)

```
    it('should show completed entries', function () {
        var todoArray = [
            {title: 'my todo', completed: true},
            {title: 'my todo2', completed: false},
            {title: 'my todo3', completed: true}
        ];
        setUpModel(todoArray);
        subject.setView('');

        expect(view.render).toHaveBeenCalledWith('clearCompletedButton', Object({ completed: 2, visible: true }) );
    });
});
```

4-'should show the content block when todos exists'
create a list of the tasks, when they exist

```
it('should show the content block when todos exists', function () {
    setUpModel([{title: 'my todo', completed: true}]);

    subject.setView('');

    expect(view.render).toHaveBeenCalledWith('contentBlockVisibility', {
        visible: true
    });
});
```

5-'should highlight "All" filter by default'
sets 'all' as default, takes total count, even if it's empty;

```
it('should highlight "All" filter by default', function () {
    var todo = {title: 'my todo'};
    setUpModel([todo]);

    subject.setView('#/');
    expect(view.render).toHaveBeenCalledWith('contentBlockVisibility', Object({ visible: true }));
});
```

6-'should toggle all todos to completed'
updates all tasks as completed (model component);

```
it('should toggle all todos to completed', function () {
    var todo = [
        {id: 42, title: 'my todo', completed: true},
        {id: 43, title: 'my todo2', completed: false},
        {id: 44, title: 'my todo3', completed: true},
        {id: 45, title: 'my todo4', completed: false}
    ];

    setUpModel(todo);

    subject.setView('#/completed');

    expect(view.render).toHaveBeenCalledWith('contentBlockVisibility', Object({ visible: true }));
    expect(view.render).toHaveBeenCalledWith('setFilter', 'completed');
});
```

7-‘should update the view'
updates the status as completed (view component);

```
    it('should update the view', function () {
        var todo = [
            {id: 42, title: 'my todo', completed: false},
        ];

        setUpModel(todo);

        subject.setView('#/');
        view.trigger('itemToggle', {id: 42, completed: true});
        expect(view.render).toHaveBeenCalledWith('updateElementCount', 1);
        expect(view.render).toHaveBeenCalledWith('toggleAll', Object({ checked: false }) );
        expect(view.render).toHaveBeenCalledWith('elementComplete', Object({ id: 42, completed: true }) );
    });
});
```

8-'should add a new todo to the model'
adds new task to the list;

```
it('should add a new todo to the model', function () {
    setUpModel([{id: 42, title: 'my todo'}]);
    subject.setView('');

    expect(view.render).toHaveBeenCalledWith( 'updateElementCount', 1);
    expect(view.render).toHaveBeenCalledWith( 'showEntries',  [ Object({ id: 42, title: 'my todo' }) ] );
});
```

9-'should remove an entry from the model'
removes todo task (model component), empty array.

```
    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));
    });
```

Performance

Competitor website: http://todolistme.net/
(go to performance folder)
Our website: https://yehia67.github.io

Comparison Conclusion

	Performance	Accessibility	Best Practices	SEO
Our project for Desktop	97%	48%	93%	75%
Competitor project for Desktop	37%	38%	71%	78%
Our project for Mobile	98%	48%	93%	60%
Competitor project for Mobile	51%	38%	71%	64%

From Numbers it’ obvious our project is much better:
after checking the audit report I found the Following reasons:-
1-Don’t have advertisements on our website.
2-Use less features.
3-Use Vanilla Javascript.
4-Simple CSS not SASS.
5-The competitor have unused css styles on his website.
6-Have more features then our app.
7-More complicated Database our databse use only strings.

What we need to make our website more the 97% preformance for Desktop and for mobile:-
1-Use minify javascript files.
2-Properly Size Images.
3-Pre-connect to required origins.
4-Serve images in next-gen formats.

All the detailed description of performance is available at performance/ directory.