SC-T-213-VEFF - Web Programming I

Project Assignment 2 – Client-side Application

Reykjavik University | Deadline: **18th February 2021, 23:59**

The topic of this assignment is: **Writing a complete client-side application**.

1 Overview

In this assignment, you will develop a web application that handles planning boards, similar to the boards you can see in applications like Trello[1]. You will communicate extensively with an existing server-side application that manages these boards. In our case, there are different **boards** that can have **tasks**.

In Figure 1 you see an example of three different boards ("Backlog", "Ongoing", and "Done"). The "Planned" board has two tasks ("Prepare exam draft" and "Discuss exam organisation").

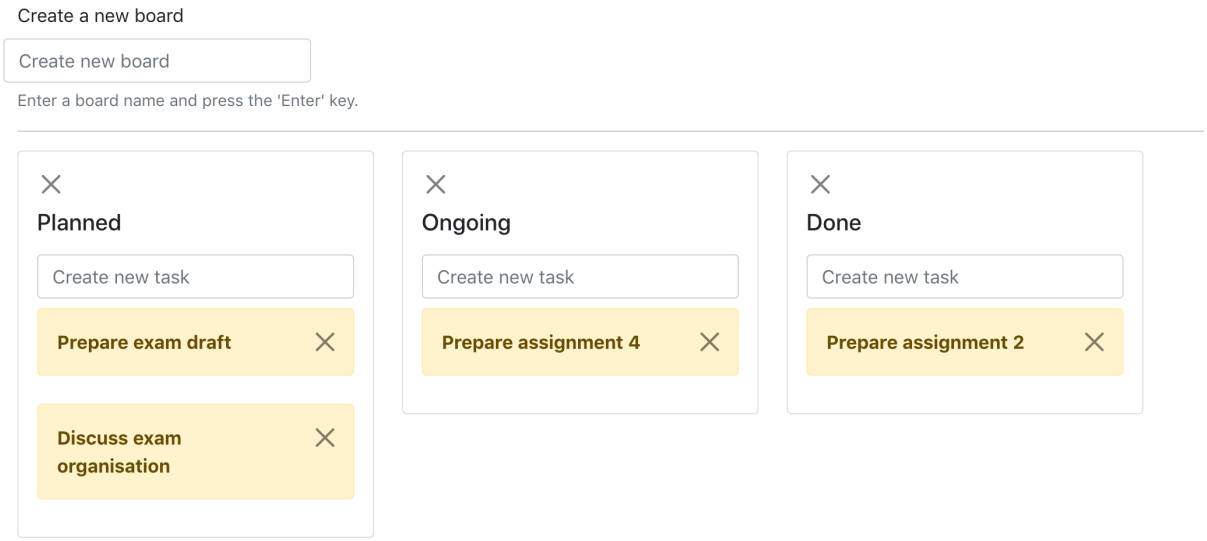

Figure 1: Example Application

This assignment has to be done individual - no group work is permitted.

In the following sections, the different parts of the application are outlined.

Note: If you are new to HTML, CSS and JavaScript, this task is quite challenging and might seem daunting for you. To get started, try to break the problem down into pieces and look into each problem separately. Specifically, we recommend to follow these steps:

1. Write static HTML/CSS code that shows sample boards/tasks, without any JavaScript, and without using the backend.

[1] https://trello.com

2. Add JavaScript so that the application works, but without using the backend (i.e., you just use the sample boards/tasks you created in step 1).

3. Try out the HTTP requests to the backend (e.g., using Postman or a separate HTML/JavaScript page).

4. Step by step, replace the sample content from step 1 with the actual operations from the backend.

2 Use Cases and Requirements

The focus in this assignment should be on the logic. Since we are not studying design, the look of the application is secondary and does not give any extra points. However, the application needs to have a basic usability. For instance, buttons to delete boards/tasks, and text fields to create them should be clearly visible. Similarly, boards and tasks need to be visible (but scroll bars are allowed).

The following seven cases shall be supported by the application:

1. When the page is loaded, all boards and tasks are received from the backend. Boards are then drawn as some kind of box that contains the board name, all its non-archived tasks (so that it is clearly visible which task belongs to which board), and an input field for creating a new task in that board. Boards and tasks have a button to delete/archive them. Furthermore, an input field to create new boards is drawn in the top of the page.

2. When the user enters a board name into the "Create a new board" input field and presses enter, a new board with the same name is created in the backend. If this is successful, the board is drawn on the web page. If it is unsuccessful, nothing happens. Multiple boards with the same name are allowed.

3. When the user enters a task name into any of the "Create new task" input fields and presses enter, a new task with the same name is created in the right board. If this is successful, the task is drawn in the right board box. If it is unsuccessful, nothing happens. Multiple tasks with the same name are allowed.

4. When the user clicks on any of the boards' delete/close buttons, the selected board is deleted in the backend. If this is successful, the board disappears from the page. If it is unsuccessful, nothing happens. Note that boards can only be deleted if they do not contain tasks (the backend takes care of this - you do not have to check).

5. When the user clicks on any of the tasks' delete/close buttons, the selected task is archived in the backend. If this is successful, the task disappears from the board. If it is unsuccessful, nothing happens.

6. When the user drags any task from one board to another, that task is moved in the backend. If this is successful, the task is moved to the destination board in the page. If it is unsuccessful, nothing happens.

In addition to the use cases, the following requirements shall be closely followed:

1. The application uses only one HTML file named *boards.html*.

2. The application shall never cause the HTML to reload.

3. You only need to react to your own changes in the backend. You do not need to update your page based on other changes (e.g., another user deletes/creates a board.[2]).

4. The HTML document shall validate as an HTML 5 document without errors in the W3C validator (https://validator.w3.org), using automatic detection for the document type.

5. The CSS document, if present, shall validate as correct CSS with the *CSS Level 3 + SVG* profile in the W3C CSS Validator (https://jigsaw.w3.org/css-validator).

6. All CSS code shall be placed in its own CSS file, all JavaScript code in its own JS file. No inline CSS or JavaScript is allowed (with the exception of JavaScript function calls from HTML events).

7. External CSS files (e.g., Bootstrap) are allowed.

8. External JS libraries/frameworks (e.g., jQuery) are only allowed in two cases. First, Axios is allowed (to perform AJAX HTTP requests). Secondly, JS libraries that are used as a part of an external CSS library (e.g., as is the case for Bootstrap) are allowed. In this case, you are not allowed to use these external JS libraries in your own code.

9. There are no restrictions on the ECMAScript (JavaScript) version.

You are free to design the application in any way you like, as long as it supports the named use cases, and closely follows the requirements.

3 Board Backend

For this assignment, we use an existing backend that keeps track of boards and tasks. We deployed the backend at 9 urls, one for each section. These are:

- https://veff-boards-h1.herokuapp.com/
- https://veff-boards-h2.herokuapp.com/
- https://veff-boards-h3.herokuapp.com/
- https://veff-boards-h4.herokuapp.com/
- https://veff-boards-h5.herokuapp.com/
- https://veff-boards-h6.herokuapp.com/
- https://veff-boards-h7.herokuapp.com/
- https://veff-boards-hmv.herokuapp.com/
- https://veff-boards-aku.herokuapp.com/

Whenever these websites are not available, it means that the backend is down and cannot be reached.

The backend supports a number of endpoints (basically "actions") that are relevant to this assignment. These are:

[2] Since many students will work on the same backend, this is likely to happen.

1. GET `/api/v1/boards`
   A GET request to this URL returns an array of all existing boards. Each board has an id, a name, and a description (irrelevant for this assignment).

2. GET `/api/v1/boards/:boardId/tasks`
   A GET request to this URL returns an array of all existing tasks for a given board with id **:boardId**. Each task has an id, a boardId field (which should have the same value as the boardId you provided), a taskName, a creation date, and an archived flag (we will "archive" tasks instead of deleting them).

3. POST `/api/v1/boards`
   A POST request to this URL creates a new board. You need to provide the attributes name and description in the request body (but description can be an empty string).

4. POST `/api/v1/boards/:boardId/tasks`
   A POST request to this URL creates a new task for a given board (if the board exists). You need to provide only the attribute taskName.

5. DELETE `/api/v1/boards/:boardId`
   A DELETE request to this URL deletes the board with the provided boardId. This is only possible if the board does not contain any tasks, or if all tasks are archived.

6. PATCH `/api/v1/boards/:boardId/tasks/:taskId`
   A PATCH request to this URL makes it possible to modify a task. To make modifications, you can provide any combination of the three attributes that can be changed in the request body: boardId, taskName, archived. For instance, if you send a PATCH request with the request body boardId: 2, the task is moved to the board with boardId 2. In this assignment, we will use this request to move tasks between boards (by providing the boardId attribute), and to archive tasks (by providing the archived attribute with value true).

In general, the different endpoints return 2xx status codes if successful, and 4xx/5xx status codes if unsuccessful. POST, DELETE, and PATCH methods return the created/deleted/modified object.

To call any of the endpoints, you send an HTTP request with the right HTTP method (verb) and the right request body to "your" backend url with the added endpoint path[3]. In the supplementary material, you find an example (named *axios.html*) on how this is done programmatically using the Axios library, for endpoints 1 and 4. You can adapt this example to send requests to the other endpoints of the given backend as well. Note that GET requests do not have a request body, while POST, DELETE, and PATCH require it. Additionally, it might be a good idea to start trying out the different requests using Postman or curl.

Submission

The lab is submitted via Canvas. Submit a zip file containing your *boards.html* file and all additional resources needed (e.g., CSS files, JS files, images).

[3]For instance, if you are in HMV and you want to get all boards, you need to send a GET request to `https://veff-boards-hmv.herokuapp.com/api/v1/boards`