ASSIGNMENT DOCUMENTATION

Centre of Excellence

Kliment Markoski

1. Introduction

The task provided was to create a Simple Web Based to-do tracker in web frontend framework/library of my choice. The application should run in modern web browsers (Google Chrome, Microsoft Edge, Mozilla Firefox, …). You should use general web technologies such as HTML, CSS, JavaScript/Typescript. If you choose to do so, you can store the items in the supplied API, storing the items is not required – to-do items can be stored in memory of the web browser.

1.1 Application Expectations

The web application front-end should contain components or pages for performing the required actions:

- Add new to-do item
- View all to-do items
- Delete a to-do item
- Modify an existing to-do item

The user should not have to click around too much to perform a specific task.

1.2 Approach

As an initial approach, I will be creating this app using JavaScript (help using express and nodemon), and Node.js regarding the REST API because the provided ones don't seem to work, I will be creating a server as a backend which will be using REST APIs. This way I will ensure that the task is fully covered.

*Note: I will be referring to the to-dos as *activities* in this documentation

2. Creation of Frontend

Without any reason starting with the frontend feels more comfortable for this task. I plan to create two separate *views* of *one page.* The first view will consist of the gadgets needed to add an activity. The second view will be intended to display all the activities with the possibility of editing them and deleting them. Changing from one view to another will be achieved by clicking on certain elements of the page.

2.1. Creation of views

My student instinct says I should put all the gadgets in the same HTML and work on one JavaScript file to achieve simplicity. However, this approach might be simple, but it can be messy. I will try to divide the code into classes and create a module script to take advantage of the import/export syntax. I will create a second server for my frontend (One will be for the backend for the REST API).

Starting, I will create a front-end folder with an HTML file. Every route I will have will be leading to this HTML file. This way I will avoid multiple pages and page refreshes and I will have a nice, divided code for the two views.

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/html">
<head>
    <meta charset="UTF-8">
    <title>To Do List</title>
    <link rel="stylesheet" href="./static/css/code.css">
    <link rel="icon" type="image/x-png" href="./static/js/option.png" >
</head>
<body>
    <nav class="nav">
        <a href="/" class="nav__link" data-link>New Activity</a>
        <a href="/list" class="nav__link" data-link>List Activities</a>
    </nav>
    <div id="app"></div>
    <script type="module" src="static/js/code.js"></script>
</body>
</html>
```

As I mentioned, I will be using a module script. The views will have their own classes which will inject HTML into the parent HTML. Specifically, it will be injected in the ** ** with the **id=”app”.** The main JavaScript class, code.js, will primarily consist of the routing mechanism for navigating through the views. The actual functionalities of the views will be implemented in the view classes themselves.

2.2 Code.js - navigation

In this file we have the routing logic. In short, the logic is all based on this attribute “**data-link”**, once you click if that href contains this attribute it will stop the refresh of the page and insert

the new HTML through the function ***getHTML()*** that both views have. This function contains the **innerHTML** that needs to be injected into the main HTML. Providing all the necessary gadgets and their functionalities. Additionally, this class contains a small code for a **form** validation upon submitting.

3. Creation of Backend

Regarding the REST APIs they will have the mentioned structure from the assignment,

```
{
    name: String,
    description: String,
    completed: boolean
})
```

The ***ID*** and the ***timestamp*** are additionally added because they are generated automatically and not inputted by the user.

Creation of the backend is in the backed.js file. In this file, we have the creation of the server and the **GET, POST, PUT, DELETE REST** APIs and **GET/:id**. Simply created with request and response to the server. I tested them and developed them with the help of POSTMAN.

3.1 Gadgets functionalities/backend

The gadgets are divided into two groups, belonging to the two views, respectively. I have created an abstract class with a constructor and a method **getHTML().**

3.3.1 newActivity view

This view consists of the form that has the input for the *name* and *description* and a checkbox for the *completed* attribute. Finally, a simple submit button. Once the fields are filed in (added validation so they cannot be empty) and the button is pressed the form will trigger the method **addActivity()** that will take the values and send them to the other server, the backend. It uses the **POST** REST API

3.3.2 listActivities view

The second view is slightly larger because it contains the edit and the delete functionalities. Upon navigating to this view first the **list()** function is called that will display previously added activities. This function uses the **GET** REST API to obtain all the activities from the backend server. The way it works is it goes through each activity from the backend and it creates a ** **

for it. This ** ** contains all the gadgets that are needed for one activity to exist on the parent html.

Regarding the edit and delete functionalities. The **editActivity()** function will simply find the activity on the backend server and replace the values. To achieve this, I use the **GET/:id** and **PUT** REST API, while the **deleteActivity()** will simply remove it from the list entirely. It uses the same two REST APIs.

4. Expending the backend to a database

Once everything is in order and working, I start working on replacing the server with a connection to a database. This way once the backed server is turned off, I do not need to worry about the date being lost. I will be using a NoSQL database, namely MongoDB. Why did I choose this database? The first reason is that I am working with JSON files. Choosing NoSQL database is the best option in this assignment. The second reason is that there is not that much information that I am storing and not a lot of relationships to be kept. If I were to extend the assignment to log in multiple users at the same time, then considering a relationship database such as PostgreSQL does not seem that bad.

I achieved this by writing another backend file named *mongoDB.js*, which contains the same code with some extensions and simplifications as the *backend.js*.

5. Errors and Improvements

Avoiding presenting errors and incorrect implementations in a presentation is a clever idea. I do like to be honest and keep the documentation as detailed as I can.

- When it comes to errors, there are some improvements to the navigation. My prediction is that I missed some **await function** in the implementation of the code.
- Regarding the editing I did not add a validation form to the edit. Which means once you open an edit from an activity and remove the text from the fields you can store it as an empty object.
- Moving to a database from the server resulted in the need to change the code for the **editActivity()** and **deleteActivity()** functions. Which I did not have time check or to correct what was needed. This means If you run the app to store the activities in the DB it will crash upon edit or delete.