ENSF-381: Full Stack Web Development Laboratory

Ahmad Abdellatif and Novarun Deb
Department of Electrical & Software Engineering
University of Calgary

Lab 5

Objectives

Welcome to the ENSF381 course lab! Below are the detailed instructions for creating and understanding CSS. The objective of this lab is to provide students with practical experience in creating dynamic and interactive web applications by combining frontend development techniques with backend data sources. We will learn to update and style webpage elements using the DOM, implement event handling with addEventListener, and create mock APIs.

Groups

Lab instructions must be followed in groups **of two students**.

Submission

You must submit the complete source code, ensuring it can be executed without any modifications. Also, if requested by the instructor, you may need to submit the corresponding documentation file (e.g., word and image). Only one member of the group needs to submit the assignment, but the submission must include the names and UCIDs of all group members at the top of the code.

Deadline

Lab exercises must be submitted by **11:55 PM on the same day as the lab session**. Submissions made within 24 hours after the deadline will receive a maximum of 50% of the mark. Submissions made beyond 24 hours will not be evaluated and will receive a grade of zero.

Academic Misconduct

Academic Misconduct refers to student behavior which compromises proper assessment of a student's academic activities and includes: cheating; fabrication; falsification; plagiarism; unauthorized assistance; failure to comply with an instructor's expectations regarding conduct required of students completing academic assessments in their courses; and failure to comply with exam regulations applied by the Registrar.

For more information on the University of Calgary Student Academic Misconduct Policy and Procedure and the SSE Academic Misconduct Operating Standard, please visit: https://schulich.ucalgary.ca/current-students/undergraduate/student-resources/policies-and-procedures

Exercise 1: DOM Manipulation with Event Listeners

Objective: In this exercise, we will learn to dynamically update and style elements on a webpage using the Document Object Model (DOM**).** Also, we will explore how to attach event handlers using the addEventListener method for better modularity and flexibility in your JavaScript code.

AddEventListener Method

We learned how to define event handlers directly in HTML attributes, such as using onkeypress or onclick. There are other methods of defining event handlers, using the addEventListener method in JavaScript. There are several benefits of using addEventListener:

1. Separation of Concerns**:** Keeps your HTML clean by defining the behaviour directly in your JavaScript code.
2. Multiple Handlers**:** Allows you to attach multiple event listeners to the same element for different events or functionalities.
3. Dynamic Addition and Removal**:** Enables you to dynamically add or remove event handlers as needed.

Example:

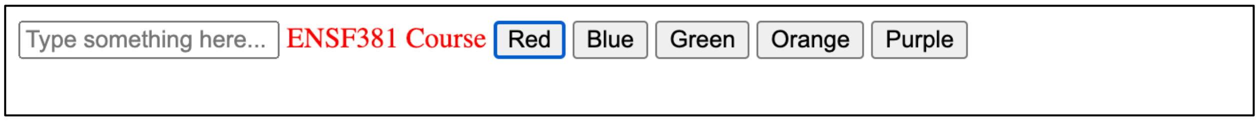
```
// Selecting an element
const button = document.getElementById('exampleButton');

// Adding a click event listener
button.addEventListener('click', () => {
    alert('Button was clicked!');
});
```

In this lab, we will apply the addEventListener method to handle keypresses for the textbox and clicks for the buttons.

Requirements

In this exercise, you need to create an interactive webpage with a textbox, a label, and five buttons. The textbox will allow users to input text. When the **Enter key** is pressed, the text will be displayed inside the label, and the textbox will be cleared. Each button should change the font color of the label to a specific color when clicked. The colours for the buttons are as follows: Red**,** Blue**,** Green**,** Orange, and Purple. You are required to use the addEventListener method to attach event handlers for both keypress and click events, and use style.color to modify the font color of the label dynamically.



Create an HTML file with the following structure:

```
<!DOCTYPE html>
<html>
<head>
    <title>Lab5 - Exercise 1</title>
</head>
<body>
    <input type="text" id="textInput" placeholder="Type something
here..."/>
    <label id="textLabel"></label>
    <button id="redButton">Red</button>
    <button id="blueButton">Blue</button>
    <button id="greenButton">Green</button>
    <button id="orangeButton">Orange</button>
    <button id="purpleButton">Purple</button>
</body>
<script type="text/javascript">
    <!-- Your JavaScript code will go here -->
</script>
</html>
```

JavaScript Implementation

Write JavaScript code in the <script> section of the HTML file using the addEventListener method to:

1. **Textbox to Label**:
   - Add an event listener to detect when the **Enter key** is pressed in the textbox.
   - Update the label with the entered text and clear the textbox.
2. **Button Clicks**:
   - Add event listeners to each button.
   - Change the label's font color to the corresponding color when a button is clicked. The buttons will have the following functionality:
     - Red Button: Change to red.
     - Blue Button: Change to blue.
     - Green Button: Change to green.
     - Orange Button: Change to orange.
     - Purple Button: Change to purple.

Submission:

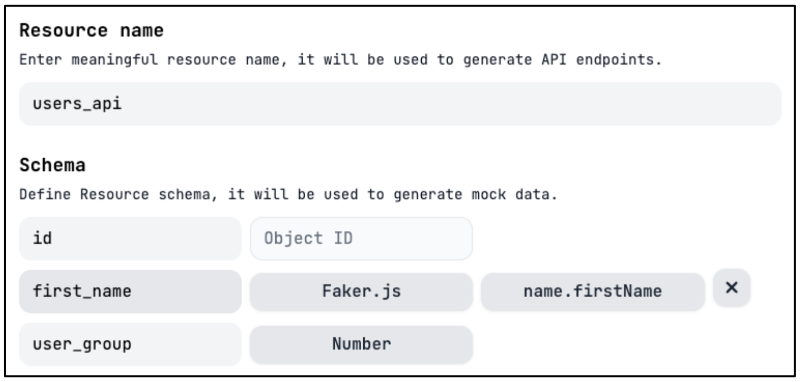
1- Fill out the Answer_sheet.docx, including:
   1. The names and UCIDs of all group members.
   2. Write the section number in the textbox, change its color to red, and include a screenshot showing the output of Exercise 1.

2- Create a new GitHub repository and upload the code for Exercise 1 to the new repository.

Exercise 2: Working with MockAPI.io

Objective: MockAPI.io is an online tool for quickly creating mock APIs. It simplifies the process of generating fake data and endpoints, making it particularly useful for prototyping applications. The objective of this exercise is to familiarize students with creating and interacting with mock APIs using MockAPI.io

1. Open your browser and navigate to MockAPI.io: https://mockapi.io/
2. Click on "Sign Up" to create an account using your email address.
3. Log in into your account and click on "Create Project" by clicking on + to create a new project.
4. Name your project: "FullStackLab"**.**
5. Inside the FullStackLab project, click "Create resource" to add a new API.
6. Delete all fields such as "id" and "name", except the "ID".
7. Set the API name to "users_api".
8. Add two fields for the API:
    - first_name (Faker.js) and set the random value to "name.firstName". Faker.js is a library that generates random, realistic data for testing. Using name.firstName as the random value ensures the first_name field is populated with realistic first names.
    - user_group (number type), which represents a category or group assigned to users, such as their role, type, or status. This field can be used to organize and manage users based on their role.



Resource name

Enter meaningful resource name, it will be used to generate API endpoints.

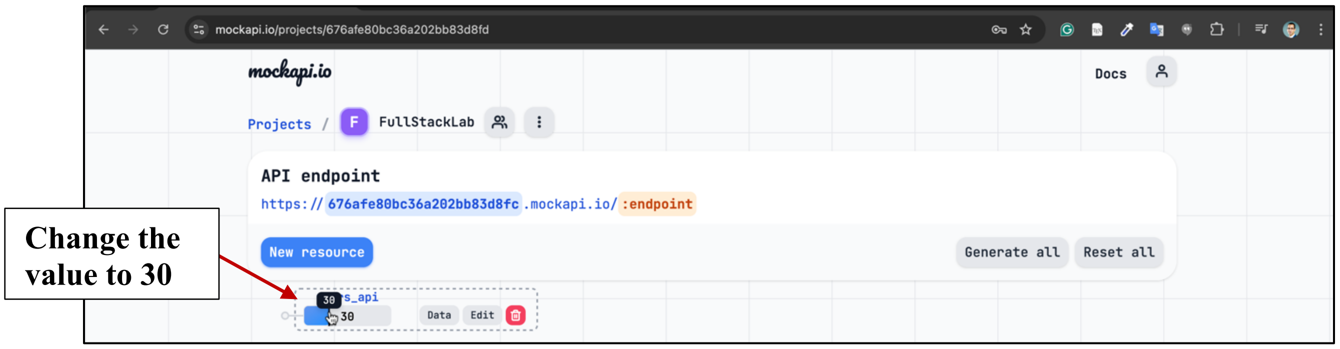
users_api

Schema

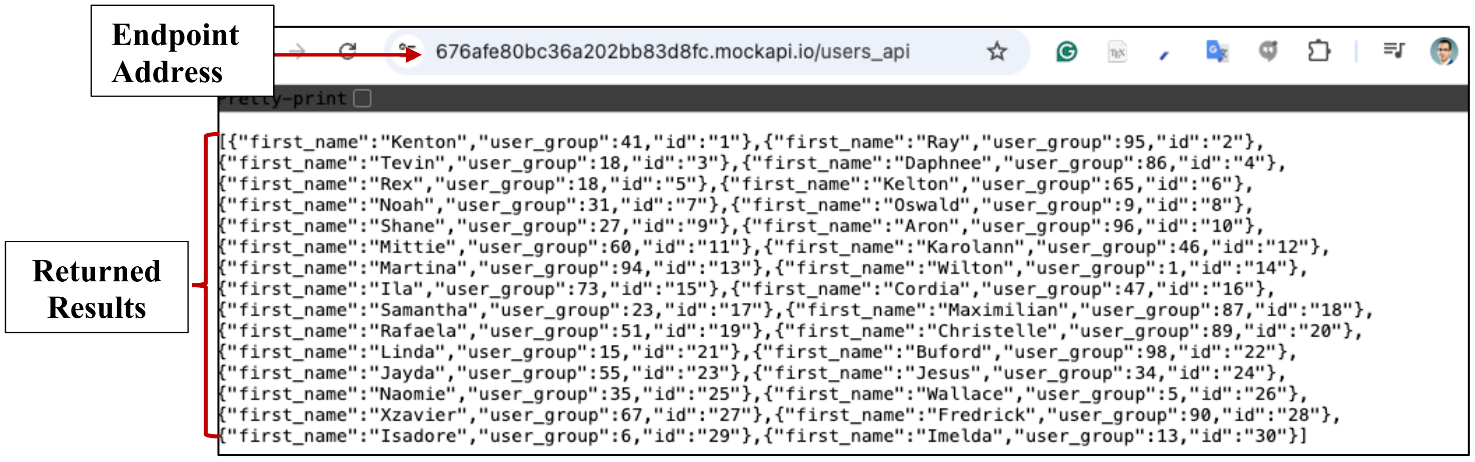
Define Resource schema, it will be used to generate mock data.

id	Object ID	
first_name	Faker.js	name.firstName
user_group	Number	

9. Click on the Create button. This process creates a new resource within the project, and it initializes the API with the specified name and fields.
10. MockAPI.io enables us to control the number of records returned in each API response, making it ideal for simulating real-world scenarios. In this step, we will configure the API to return 30 users per request. Each time the API is called, it will consistently return a fixed number of users (i.e., 30). Click on the section that displays the number of users, as shown in the picture, and update the value to 30:


Change the value to 30
mockapi.io/projects/676afe80bc36a202bb83d8fd
mockapi.io
Docs
Projects / F FullStackLab
API endpoint
https://676afe80bc36a202bb83d8fc.mockapi.io/:endpoint
New resource
Generate all
Reset all
users_api
30
Data
Edit

11. After configuring the number of returned users, you can view the generated data. The output will look similar to the following:



```
[{"first_name":"Kenton","user_group":41,"id":"1"},{"first_name":"Ray","user_group":95,"id":"2"},
{"first_name":"Tevin","user_group":18,"id":"3"},{"first_name":"Daphnee","user_group":86,"id":"4"},
{"first_name":"Rex","user_group":18,"id":"5"},{"first_name":"Kelton","user_group":65,"id":"6"},
{"first_name":"Noah","user_group":31,"id":"7"},{"first_name":"Oswald","user_group":9,"id":"8"},
{"first_name":"Shane","user_group":27,"id":"9"},{"first_name":"Aron","user_group":96,"id":"10"},
{"first_name":"Mittie","user_group":60,"id":"11"},{"first_name":"Karolann","user_group":46,"id":"12"},
{"first_name":"Martina","user_group":94,"id":"13"},{"first_name":"Wilton","user_group":1,"id":"14"},
{"first_name":"Ila","user_group":73,"id":"15"},{"first_name":"Cordia","user_group":47,"id":"16"},
{"first_name":"Samantha","user_group":23,"id":"17"},{"first_name":"Maximilian","user_group":87,"id":"18"},
{"first_name":"Rafaela","user_group":51,"id":"19"},{"first_name":"Christelle","user_group":89,"id":"20"},
{"first_name":"Linda","user_group":15,"id":"21"},{"first_name":"Buford","user_group":98,"id":"22"},
{"first_name":"Jayda","user_group":55,"id":"23"},{"first_name":"Jesus","user_group":34,"id":"24"},
{"first_name":"Naomie","user_group":35,"id":"25"},{"first_name":"Wallace","user_group":5,"id":"26"},
{"first_name":"Xzavier","user_group":67,"id":"27"},{"first_name":"Fredrick","user_group":90,"id":"28"},
{"first_name":"Isadore","user_group":6,"id":"29"},{"first_name":"Imelda","user_group":13,"id":"30"}]
```

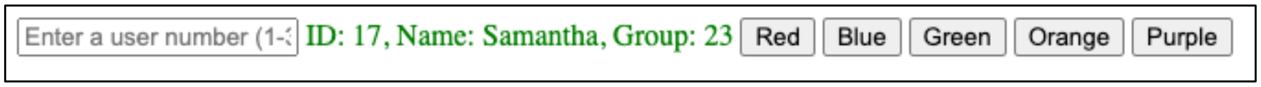
Submission: In the Answer_sheet.docx you edited for Exercise 1, include a screenshot showing the final output of in exercise 2.

Exercise 3: Displaying Dynamic Content

Objective: In this exercise, we will expand on Exercise 1 by integrating it with a mock API created in Exercise 2. You will fetch and display user information based on a user number entered in the textbox. Also, we need to maintain the functionality to change the font color of the displayed information using buttons.

1. Make a copy of the exercise1.html file and name it exercise3.html.
2. We will use the label that its ID is textLabel to display the fetched user information.
3. Write an async function retrieveData() to fetch all user data from the API. Use the same endpoint address for the users_api API that you created in Exercise 2.
4. When the Enter key is pressed, call the retrieveData() function to get all user data, iterate through the returned data, and find the user whose id matches the number entered in the textbox. You need to update event listeners for the keypress event for fetching and filtering data.
5. Populate the label step 2 with the selected user information in the following format using the for loop:

   ID: <id>, Name: <first_name>, Group: <user_group>.
6. If no user is found or the entered ID is invalid, display an appropriate error message (e.g., "No users were found with the provided User_ID.") in the label.
7. You need to update event listeners for all buttons to dynamically change the font color of all displayed information.



ID: 17, Name: Samantha, Group: 23

Submission:

1. In the Answer_sheet.docx you edited for Exercise 1, include **a single screenshot** for Exercise 3 that shows (see lab5_output.pdf):
   a. All the returned API values displayed in the developer console.
   b. The label populated with the appropriate user information based on the entered number.
2. Upload the code for Exercise 3 to the same repository you created for Exercise 1.
3. Compress both Exercise 1 and Exercise 3 into a single file and upload the compressed file to D2L. Also, upload the completed Answer_sheet.docx.