Full Stack Development Lab Instructions
ENSF381: Lab06

RestFul API

Created by: Mahdi Jaberzadeh Ansari (He/Him)
Schulich School of Engineering
University of Calgary
Calgary, Canada
mahdi.ansari1@ucalgary.ca

Week 8, February 26/28, 2024

Contents

1 Introduction

1.1 Objectives

In Lab 6, you will gain hands-on experience with the Restful API structure. This lab aims to provide practical skills in the following areas:

- **Working with API mock ups:** You will learn to use https://mockapi.io/ as a tool for modeling APIs. This platform serves as a foundation for understanding the basics of API development and modeling APIs before actual development.
- **Testing APIs:** The lab focuses on understanding the use of Postman for API development. You will learn the fundamental concepts of API construction, including designing endpoints, structuring requests and responses, and understanding the principles of RESTful APIs.
- **Webpage Creation:** You will enhance your knowledge by creating a webpage that interacts with your APIs. This includes fetching data from the API, processing it, and displaying it on the webpage, thereby integrating back-end and front-end development skills.

1.2 Prerequisites

1. Basic understanding of computer operations.
2. A web browser (Chrome, Firefox, Safari, etc.) to view your HTML file.
3. GitLab account.

1.3 Forming groups

- In this lab session, you can work with a partner (groups of three or more are not allowed).
- The main goal of working in a group is to learn:
  - how to do teamwork
  - how to not be a single player
  - how to not be bossing
  - how to play for team
  - how to tolerate others
  - how to behave with colleagues
  - how to form a winner team
  - and ...
- Working with a partner usually gives you the opportunity to discuss some of the details of the topic and learn from each other. Also, it will give you the opportunity to practice one of the popular methods of program development called pair programming. In this method, which is normally associated with the "Agile Software Development" technique, two programmers normally work together on the same workstation (you may consider a Zoom session for this purpose). While one partner, the driver, writes the code, the other partner, acting as an observer, looks over his or her shoulder, making sure the syntax and solution logic are correct. Partners should switch roles frequently in such a way that both have equivalent opportunities to practice both roles.
- If you decided to work with a partner:
  - Choose a partner that can either increase your knowledge or transfer your knowledge. (i.e., do not find a person with the same programming skill level!)
  - Please submit only one lab report with both names. Submitting two lab reports with the same content will be considered copying and plagiarism.

1.4 Before submission

- For most of the labs, you will receive a DOCX file that you need to fill out the gaps. Make sure you have this file to fill out.
- All your work should be submitted as a single file in PDF format. For instructions about how to provide your lab reports, study the posted document on the D2L called How to Hand in Your Lab assignment.
- Please note that if it is group work, only one team member must submit the solution in D2L. For ease of transferring your marks, please mention the group member's name and UCID in the description window of the submission form.
- If you have been asked to write code (HTML, CSS, JS, etc.), make sure the following information appears at the top of your code:
  - File Name
  - Assignment and exercise number
  - Your names in alphabetic order
  - Submission Date:

  Here is an example for CSS and JS files:

```
<!--
============================================================
Name        : lab6_exe_D.html
Assignment  : Lab 6, Exercise D
Author(s)   : Mahdi Ansari, William Arthur Philip Louis
Submission  : May 21, 2030
Description : API Usage.
============================================================
-->
```

- Some exercises in this lab and future labs will not be marked. Please do not skip them, because these exercises are as important as the others in learning the course material.
- In courses like ENSF381, some students skip directly to the exercises that involve writing code, skipping sections such as "Read This First," or postponing the diagram-drawing until later. That's a bad idea for several reasons:
  - "Read This First" sections normally explain some technical or syntax details that may help you solve the problem or may provide you with some hints.
  - Some lab exercises may ask you to draw a diagram, and most of the students prefer to hand-draw them. In these cases, you need to scan your diagram with a scanner or an appropriate device, such as your mobile phone, and insert the scanned picture of your diagram into your PDF file (the lab report). A possible mobile app to scan your documents is Microsoft Lens, which you can install on your mobile device for free. Please make sure your diagram is clear and readable; otherwise, you may either lose marks or it could be impossible for TAs to mark it at all.
  - Also, it is better to use the Draw.io tool if you need to draw any diagram.
  - Drawing diagrams is an important part of learning how to visualize data transfer between modules and so on. If you do diagram-drawing exercises at the last minute, you won't learn the material very well. If you do the diagrams first, you may find it easier to understand the code-writing exercises, so you may be able to finish them more quickly.
- **Due Dates:**

- You must submit your solution until 11:59 p.m. on the same day that you have the lab session.
- Submissions until 24 hours after the due date get a maximum of 50% of the mark, and after 24 hours, they will not be evaluated and get 0.

1.5 Academic Misconduct/Plagiarism

- Ask for help, but don't copy.
  - You can get help from lab instructor(s), TAs, or even your classmates as long as you do not copy other people's work.
  - If we realize that even a small portion of your work is a copy from a classmate, both parties (the donor and the receiver of the work) will be subject to academic misconduct (plagiarism). More importantly, if you give exercise solutions to a friend, you are not doing him or her a favor, as he or she will never learn that topic and will pay off for this mistake during the exam or quiz. So, please do not put yourself and your friend in a position of academic misconduct.
  - You can use ChatGPT, but please note that it may provide similar answers for others too, or even the wrong answers. For example, it has been shown that AI can hallucinate, proposing the use of libraries that do not actually exist [1]. So, we recommend that you imagine ChatGPT as an advanced search engine, not a solution provider.
  - In order to find out who is abusing these kinds of tools, we will eventually push you toward the incorrect responses that ChatGPT might produce. In that case, you might have failed for the final mark and be reported to administration.
  - If we ask you to investigate something, don't forget to mention the source of your information. Reporting without reference can lead to a zero mark even by providing a correct answer.

1.6 Marking Scheme

- You should not submit anything for the exercises that are not marked.
- In Table 1, you can find the marking scheme for each exercise.

Table 1: Marking scheme

Exercise	Marks
A	5 marks
B	15 marks
C	15 marks
D	15 marks
Total	50 marks

1.7 Complains

- Your grades will be posted one week following the submission date, which means they will be accessible at the subsequent lab meeting.
- Normally, the grades for individual labs are assessed by a distinct TA for each lab and section. Kindly refrain from contacting all TAs. If you have any concerns regarding your grades, please direct an email to the TA responsible for that specific lab.

[1] https://perma.cc/UQS5-3BBP

2 Exercise A (Creating a repository)

2.1 Initialize the Project

Like last time, we are going to clone an existing project from a remote repository. Cloning is particularly useful when you want to create a local copy of a remote repository to work on. Follow these steps:

- Create a new repository on GitLab named `Lab6`. Remember to:
  - Check the box for `Initialize repository with a README file`. This file serves as an introduction and overview of your project.
  - Select `MIT License` for your project's license. This is a permissive license that allows others considerable freedom with your code while still crediting you.
- After creating the repository, it should contain two files: a `README.md` and a `LICENSE` file. The README.md is where you can write about your project, and the LICENSE file contains the license text.

2.2 Using a Git Client for Git Operations

While previously we focused on using Git commands directly and SouirceTree, this section will focus on a Git client of your choice.

1. **Downloading and Installing a Git client:**
   - Visit the Git - GUI Clients website, review all possible items, and download one of the Git client software applications based on your preferences.
   - Please discuss the pros and cons of each client application and document the reasons that you selected your choice.
   - Follow the installation instructions to set up your preferred client on your system.
2. **Cloning a Repository using your Git client:**
   - Open your Git client and connect it to your GitLab account. You may need to generate an SSH key and add it to your GitLab account for authentication.
   - After connecting successfully to your GitLab account, use the 'Clone' feature to clone your `Lab6` repository from GitLab to your local machine. Choose a suitable destination on your local machine for the repository.
   - The process eliminates the need for command-line operations, providing a graphical interface for all actions.
   - After a successful cloning, you should be able to see the `README.md` and `LICENSE` files, along with any other files that were in the remote repository.

2.3 Copy/Paste initial materials

1. Download the lab content from D2L and unzip the materials.
2. Copy and paste materials related to this lab into the `Lab6` folder. Ensure only web-related materials (e.g., *.html or *.css file; **no *.docx file**) are copied. Include the `images` folder, if present.
3. Commit and push the initial contents to GitLab, using your Git client with this comment: 'Add initial contents'.

2.4 Inviting Collaborators to Your GitLab Repository

Collaborating with others on a project is a key aspect of software development. GitLab allows you to add collaborators to your repository. Here's how you can invite your colleagues to the repository you created in the previous steps:

- First, log in to your GitLab account and navigate to the repository you want to share, `Lab6` in this case.
- On your repository's page, locate the `Settings` tab near the top of the page and click on it. This will take you to the repository settings.
- In the settings menu, on the left side of the page, you will find a section called `Members`. Click on it.
- On the `Members` page, you will see an option to `Invite a member`. Click on this option.
- GitLab will prompt you to enter the GitLab username or email address of the person you want to invite. Enter the username or email of your colleague.
- Once you enter the username or email, GitLab will suggest the matching account. Click on the account to select it.
- After selecting the collaborator, you can set the role for them. For most cases, the `Developer` access level is sufficient. This allows them to push changes to the repository but doesn't allow actions like deleting the repository.
- Click on `Invite` to send the invitation. Your colleague will receive an email from GitLab with a link to accept the invitation.
- Inform your collaborator to check their email and accept the invitation to collaborate on your repository.

Once your colleague accepts the invitation, they will have access to the repository and can start collaborating with you on the project. This process is crucial for teamwork in software development, allowing multiple people to work together seamlessly on the same codebase. Ask your colleges to clone the repository on their local machines with your Git client.

Please note: *In the subsequent sections, each exercise should be completed by a single individual. It is imperative to rotate the role of the developer. This means that if Person X performs the steps in Exercise B, then Exercise C must be undertaken by Person Y, and this pattern should continue accordingly. We will check the commits on GitLab to give you marks.*

2.5 Deliverable

1. Add the address of your GitLab repository to your answer sheet file. Please make sure the repository is public and do not delete it until the end of the semester.
2. Make a screenshot from the History tab of your Git client application. It must at least show two commits to your GitLab repository in this step. Add the screenshot to your answer sheet file.
3. Write down your reason for selecting your preferred Git client.

3 Exercise B (Working with mockapi.io)

3.1 Objectives

MockAPI.io is an excellent tool for creating mock APIs quickly. The following steps will help you simulate a full-fledged RESTful API for student registration. Remember, for a real-world application, you would need a back-end service with database connectivity, security measures like authentication and authorization, and robust error handling. MockAPI is primarily for prototyping and testing front-end applications without a real back-end.

3.2 Read This First

Web services are software systems designed to support interoperable machine-to-machine interaction over a network. They have various formats and protocols, each with its own set of standards and usage. Let's briefly overview some of the common types of web services and then focus on RESTful services, including an explanation of CRUD and listing endpoints.

3.2.1 Types of Web Services

1. **SOAP (Simple Object Access Protocol):**
   - A protocol for exchanging structured information in the implementation of web services.
   - Uses XML to encode its calls and relies on application layer protocols, typically HTTP or SMTP, for message negotiation and transmission.
   - Known for its high security and extensibility.
2. **REST (Representational State Transfer):**
   - An architectural style for designing networked applications.
   - Relies on a stateless, client-server communication model.
   - Uses standard HTTP methods and is designed for use with lightweight data-interchange formats like JSON.
3. **XML-RPC and JSON-RPC:**
   - Remote procedure call (RPC) protocols that use XML and JSON, respectively, to encode calls.
   - Allow for communication between a client and a server.
   - Simpler than SOAP, focusing on executing functions remotely rather than passing messages.
4. **GraphQL:**
   - An open-source data query and manipulation language for APIs.
   - Provides a more efficient, powerful, and flexible alternative to REST.
   - Allows clients to define the structure of the data required, and the same structure of the data is returned from the server.

3.2.2 Focusing on RESTful Services

RESTful web services use HTTP requests to perform CRUD operations (Create, Read, Update, Delete) on data modeled as resources. These operations correspond to HTTP methods:

1. **Create (POST):**
   - Used to create a new resource.

- Data sent with the request is used to create the new entity.
- Typically returns a status code of 201 (Created).

2. **Read (GET):**
   - Retrieves information about a resource.
   - Does not change the state of the resource.
   - Considered safe and idempotent (multiple identical requests will have the same result).
3. **Update (PUT or PATCH):**
   - `PUT` is used to update or create a resource entirely, while `PATCH` is used for partial updates.
   - The client sends the updated data as part of the request.
   - Typically returns a 200 (OK) or 204 (No Content) status code if successful.
4. **Delete (DELETE):**
   - Removes the specified resource.
   - Typically returns a 200 (OK) or 204 (No Content) status code if successful.
5. **Listing (GET with Query Parameters):**
   - A variation of the GET method where a list of resources is returned.
   - Often supports query parameters for filtering, sorting, and pagination.

3.2.3 RESTful Endpoint Design

When designing RESTful services, endpoints (URLs) represent resources. A well-designed REST API has intuitive and predictable URLs that make it easy to understand and use. For example:

- `GET /students` might retrieve a list of students.
- `POST /students` would create a new student.
- `GET /students/{id}` would get the details of a specific student.
- `PUT /students/{id}` would update the details of a specific student.
- `DELETE /students/{id}` would delete a specific student.

In REST, resources and their states are manipulated using these standard HTTP methods, making the API predictable and easy to understand. This simplicity and use of standard HTTP methods have made RESTful services a popular choice for public APIs. In the following, we are going to design a mock API for students and their courses.

3.3 Steps

3.3.1 Sign Up and Log In

1. **Visit** MockAPI.io and **sign up** for an account if you don't already have one.
2. **Log in** to your account.

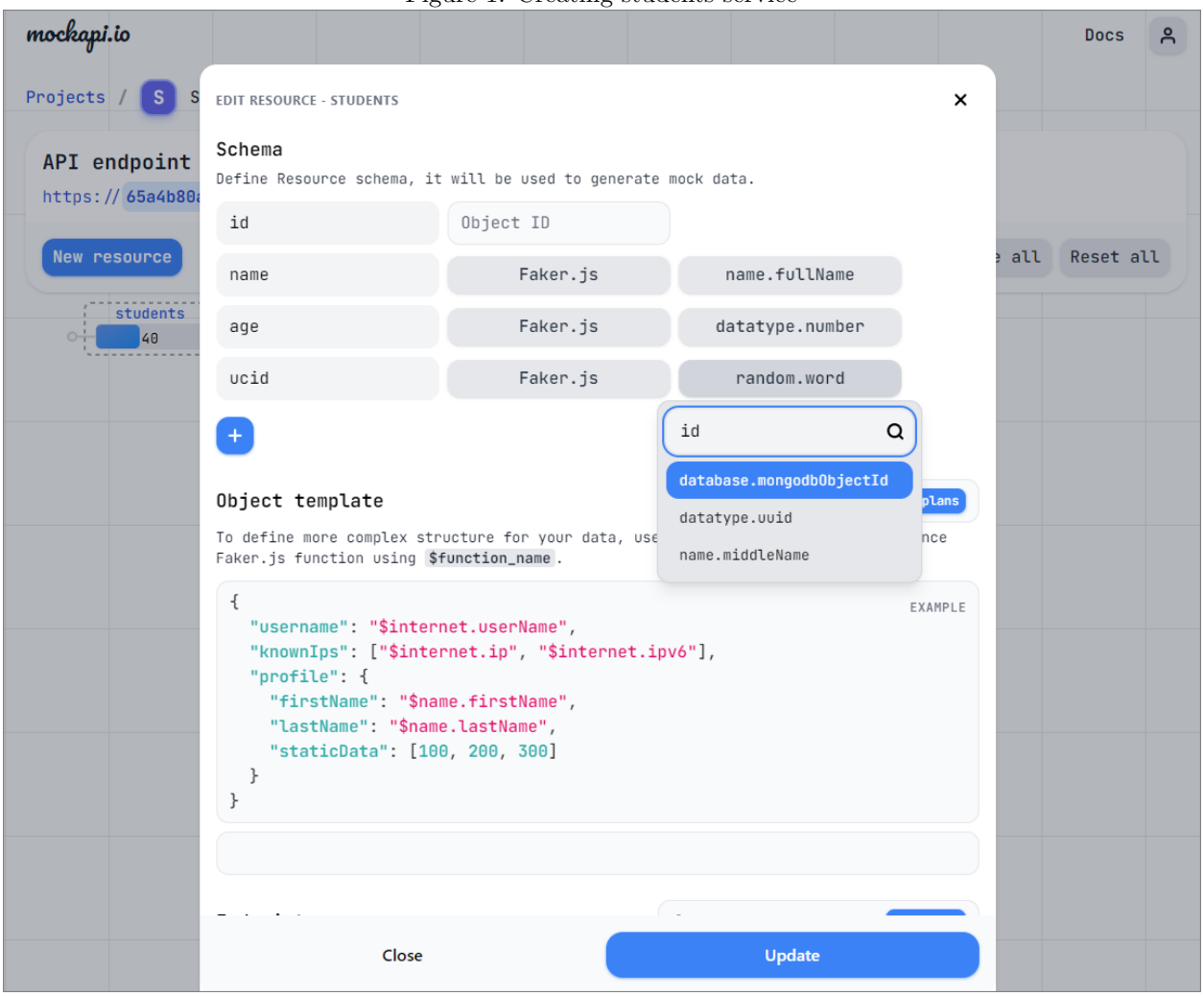
3.3.2 Create a New Project

1. **Navigate** to the Dashboard.
2. **Click** on the 'Create a Project' button.
3. **Enter** a name for your project, like 'Student Registration'.
4. **Click** 'Create'.

3.3.3 Create an API Endpoint

1. Inside your project, **click** 'New resource'.
2. **Enter** the endpoint name. For example, 'students'.
3. Create the following list of attributes as shown in Figure 1.
   - id (auto-generated object id)
   - name (string: full name)

Figure 1: Creating students service

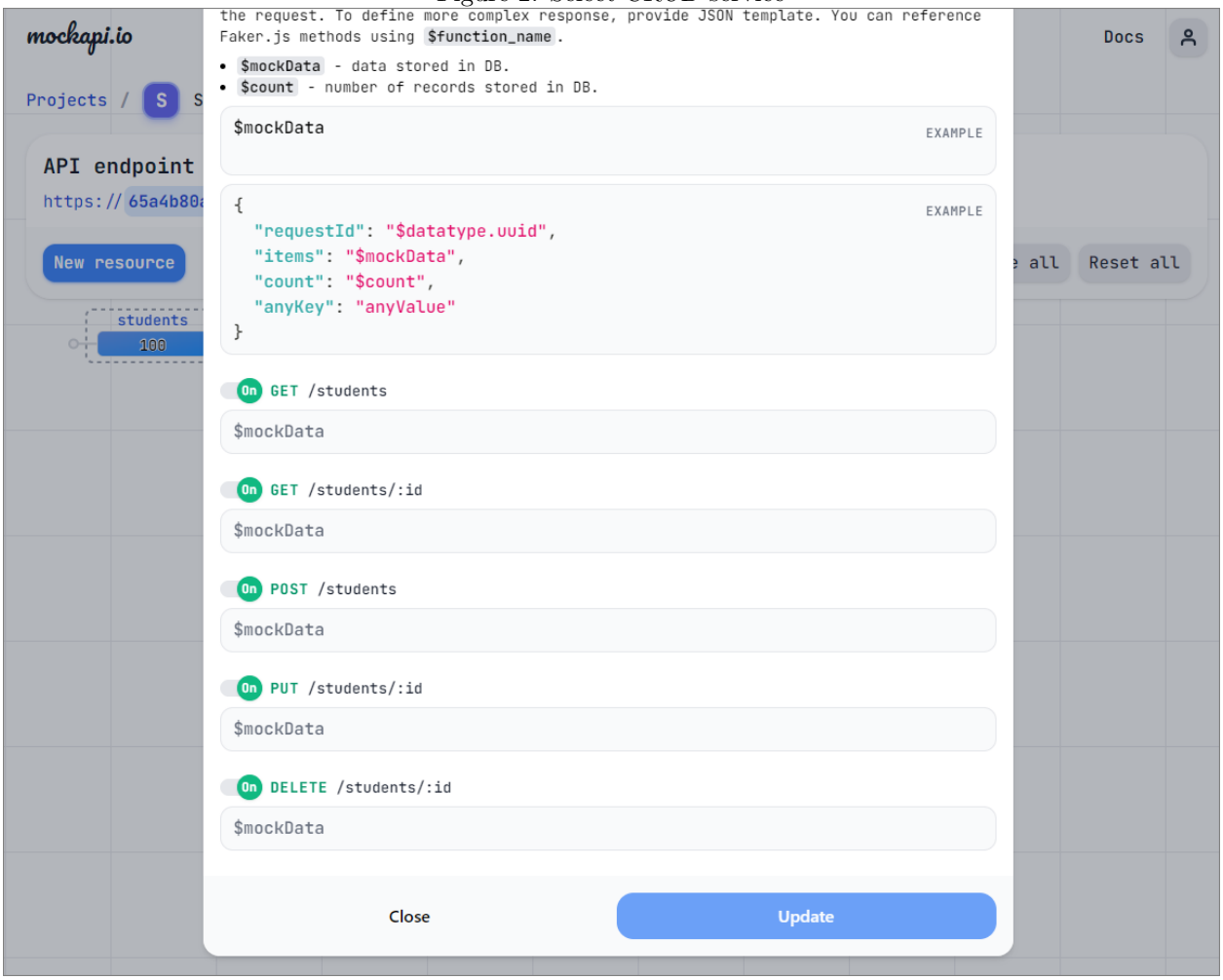


- age (integer: number)
- ucid (string: uuid or MongoDB object id)

4. Scroll down and see how all methods that we need for 'students' services are defined, as shown in Figure 2.

5. **Save** the endpoint.

Figure 2: Select CRUD service



3.3.4 Create courses service

1. Follow the same steps as you did for the 'students' service and create a 'courses' service.

2. This new service has the following items, as shown in Figure 3.

   - id (auto-generated object id)
   - name (string: word)
   - unit (integer: number)

Figure 3: Creating courses service

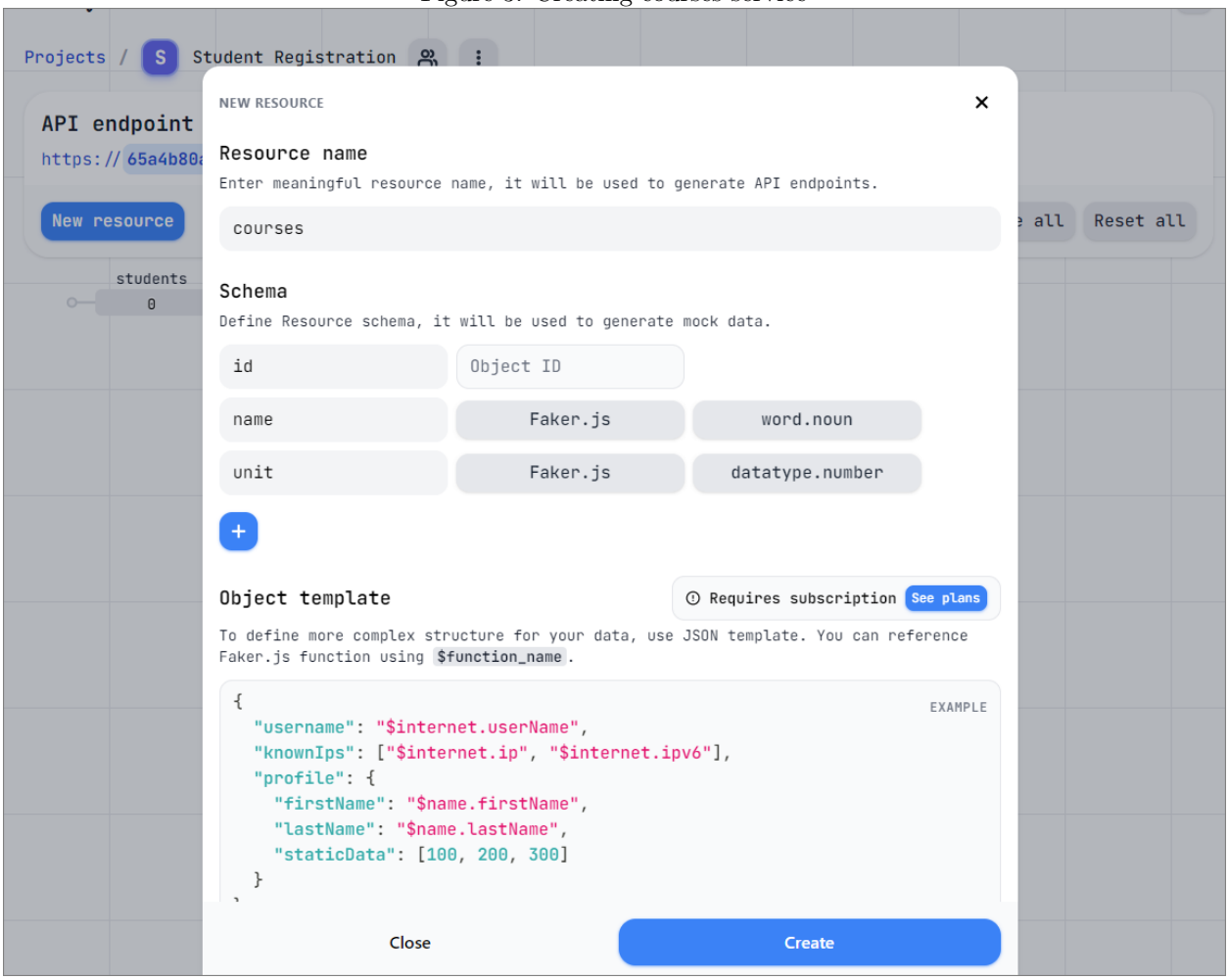
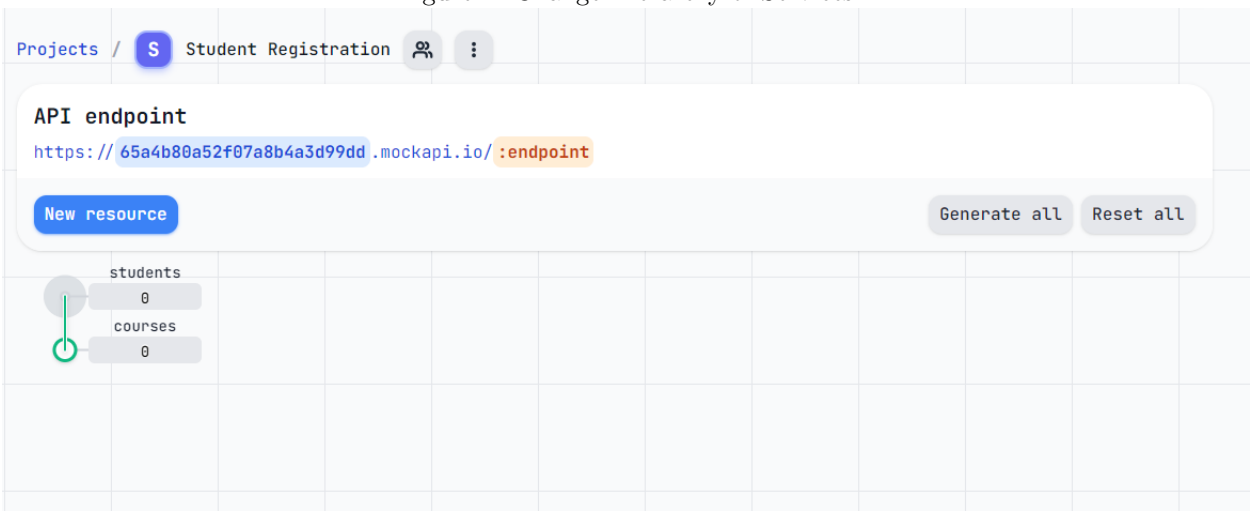


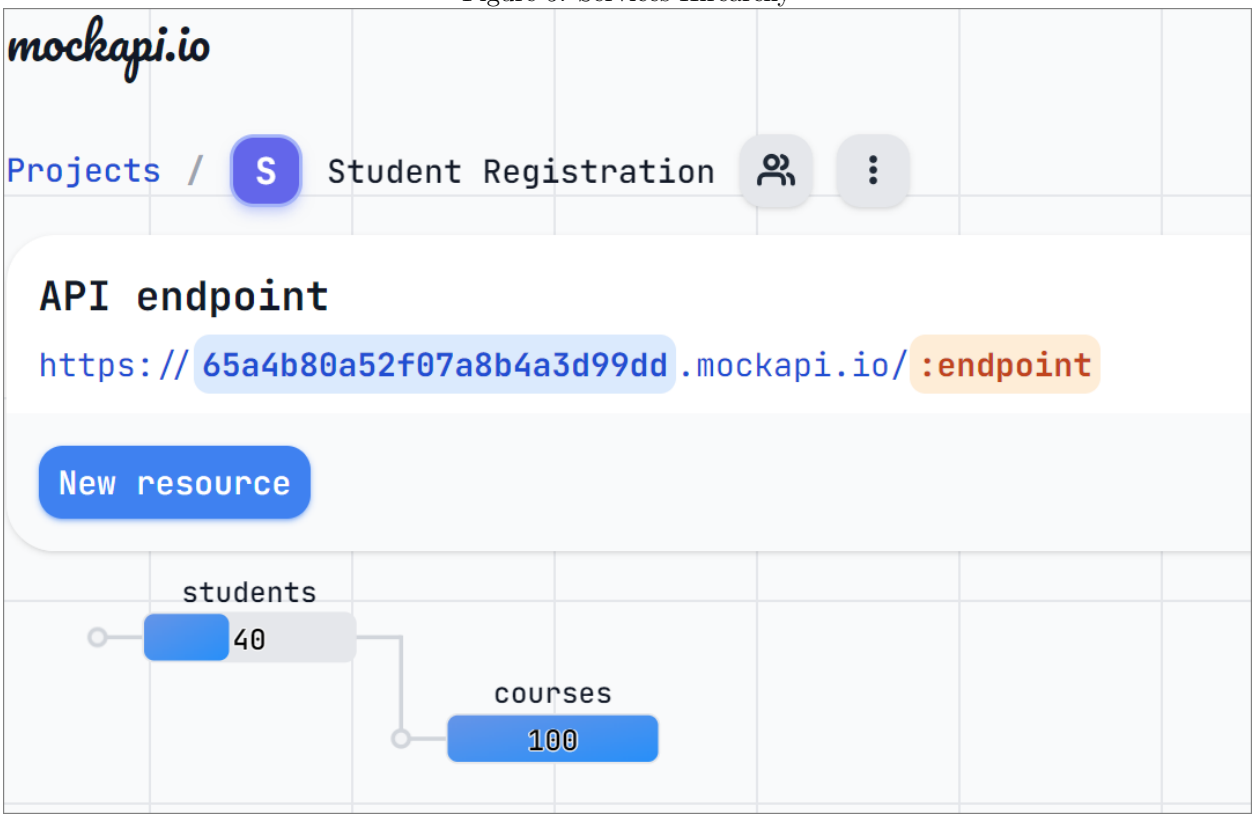
Figure 4: Change Hierarchy of Services



3.3.5 Change Hierarchy of services

1. After creating the 'courses' service, it will be shown at the same level as the 'students' service. As shown in Figure 4.
2. We want to access the courses of each student. To do so, drag the line from courses and connect it to students. If you need to see a video about how to change the hierarchy of services, check here.
3. After changing the hierarchy, you must have 'courses' under the 'students', as shown in Figure 5.

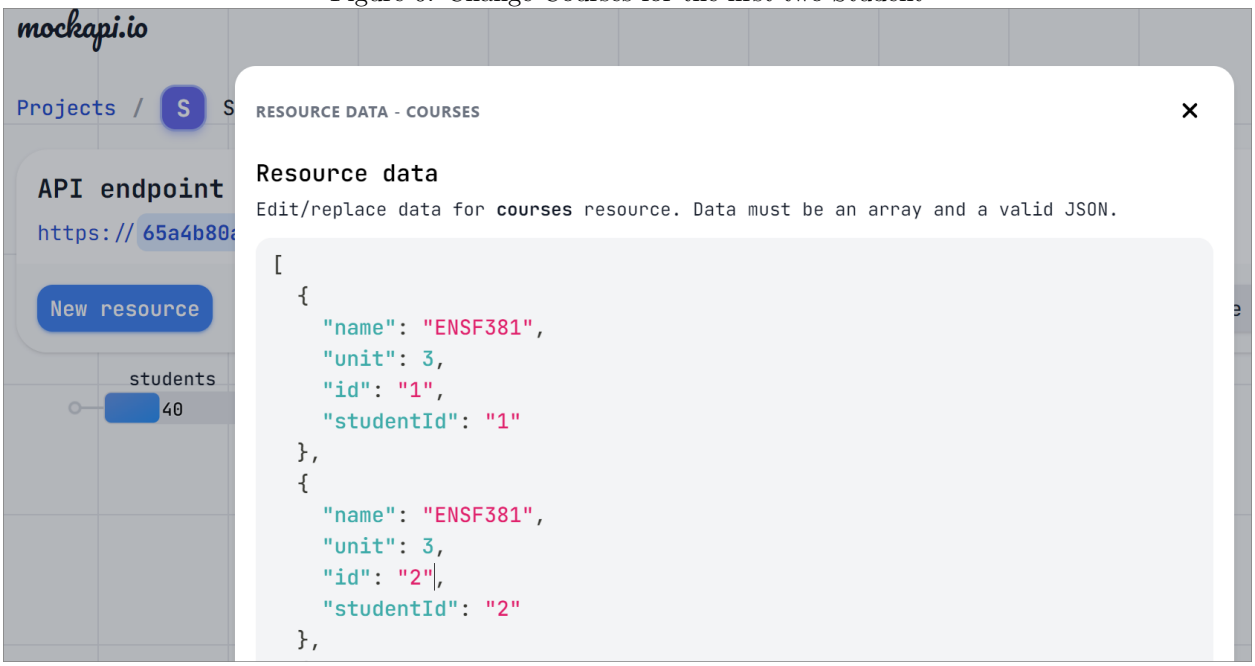
Figure 5: Services Hirearchy



3.4 Generating mock data

- For each field in the resource schema, we specified a Faker.js generator function. Check Figures 1 and 3.
- Move your mouse cursor over the 'students' bar and click as soon as it shows 40. It means you generated 40 random students. Do the same over the 'courses' bar and generate 100 courses. It means almost three courses per each student.
- At the end, it must show 40 on 'students' and '100' on courses, as shown in Figure 5.
- Please note that by changing the hierarchy, the generated data is reset.
- Hover your mouse over one of the services, then click on the 'Data' button. It opens a window that allows you to modify data. Edit the names of the first two students for the group members. Also, edit the 'courses' data and put **ENSF381** as one of the courses for each of the first two students.
- Figure 6 shows how I edited courses data for the first two students.

Figure 6: Change Courses for the first two Student



3.5 Get Courses Data

1. By clicking on the service's name, you can get the data for that service.
2. Figure 7 shows how I served the students data, and Figure 8 shows the list of courses for the student 1.

Figure 7: Get list of students

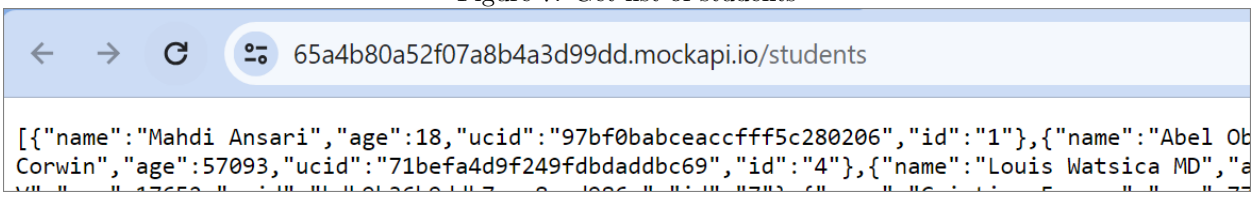
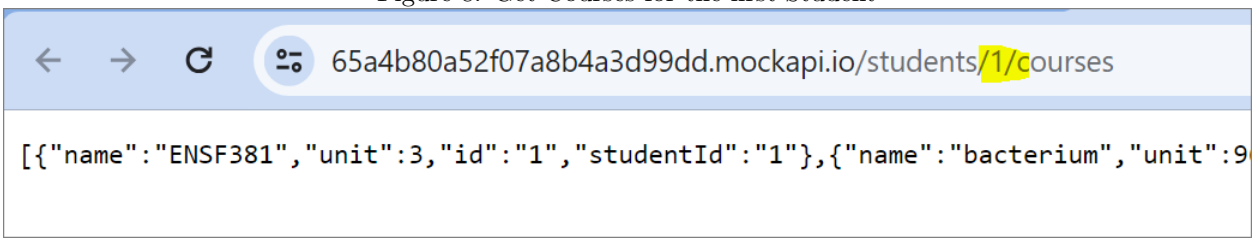


Figure 8: Get Courses for the first Student



3.6 Deliverable

1. Add the URL of your 'students' service to your answer sheet file.

2. Add a snapshot of the list of students to your answer sheet file (similar to Figure 7, but a complete list).

3. Add a snapshot of the courses of your second student to your answer sheet file (similar to Figure 8, the first course must be 'ENSF381' in the picture).

4 Exercise C (Testing APIs in Postman)

4.1 Objectives

Testing your API with Postman is an effective way to ensure that it behaves as expected. Postman allows you to simulate client requests to your API endpoints and observe the responses, which is crucial for back-end development and integration testing. Remember, always replace the URL in the requests with the actual endpoint provided by your MockAPI.io project.

4.1.1 Step 1: Install Postman

1. **Visit** the Postman website at https://www.postman.com/.
2. **Download** the Postman application suitable for your operating system.
3. **Install** Postman by following the on-screen instructions.

4.1.2 Step 2: Set Up Postman

1. **Open** Postman after installation.
2. **Create** a new account or log in if you already have one.
3. **Familiarize** yourself with the Postman interface.

4.1.3 Step 3: Create a New Collection

1. **Click** on the 'New' button (plus icon).
2. **Select** 'Collection' from the options.
3. **Name** your collection, e.g., "Student Registration API".
4. **Create** the collection by clicking 'Create'.

4.1.4 Step 4: Add Requests to Your Collection

For each of the following operations, you'll add a new request to your collection.

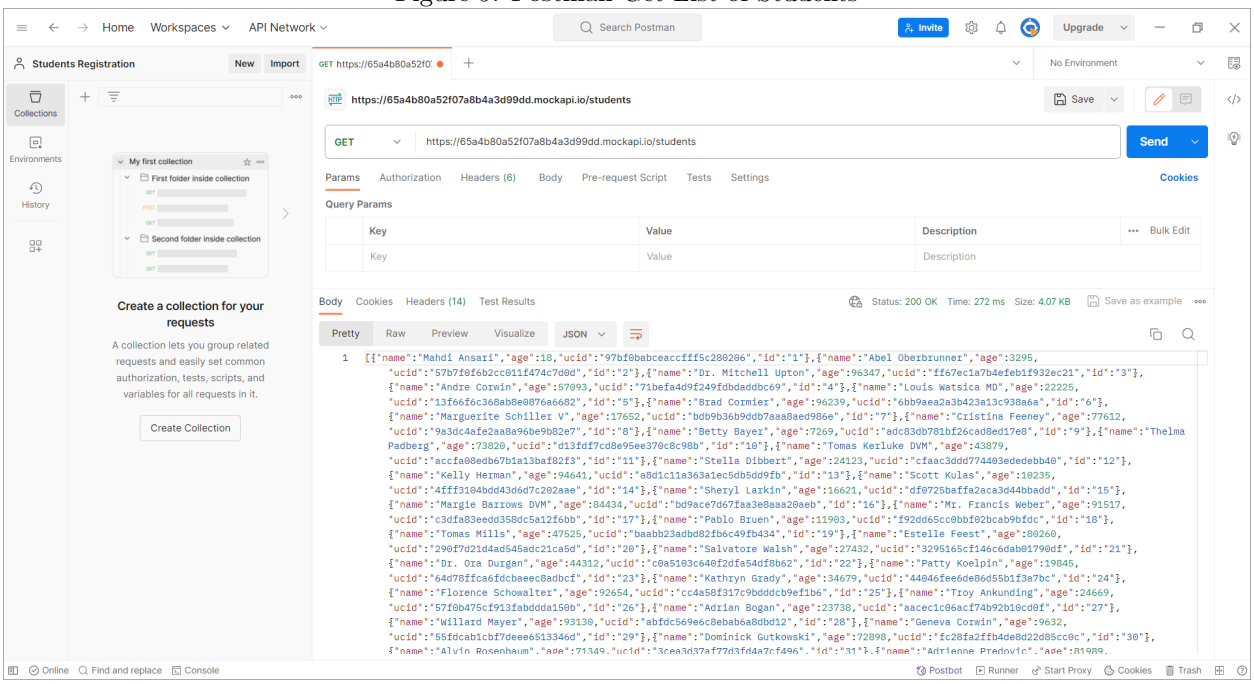
4.2 List (GET)

1. **Add** another 'GET' request for listing all students.
2. **Set** the URL to list students (e.g., `https://[your-mockapi-url]/students`).
3. **Save** the request.

4.3 Read (GET)

1. **Add** a new 'GET' request to your collection.
2. **Enter** the URL to get a specific student (e.g., `https://[your-mockapi-url]/students/{id}`).
3. **Save** the request.

Figure 9: Postman Get List of Students



4.4 Create (POST)

1. **Add** a new request to your collection.
2. **Set** the method to 'POST'.
3. **Enter** the URL for creating a new student (e.g., `https://[your-mockapi-url]/students`).
4. **Go** to the 'Body' tab, select 'raw', and choose 'JSON' as the format.
5. **Enter** a sample JSON object for a new student:

```
{
  "name": "Donald Trump",
  "age": 20,
  "ucid": "90185632"
}
```

6. **Save** the request.

4.5 Update (PUT)

1. **Add** a new 'PUT' request.
2. **Set** the URL to update a student (e.g., `https://[your-mockapi-url]/students/{id}`).
3. **In** the 'Body' section, provide the updated student information in JSON format. For example, change its name.
4. **Save** the request.

4.6 Delete (DELETE)

1. **Add** a 'DELETE' request.
2. **Set** the URL for deletion (e.g., `https://[your-mockapi-url]/students/{id}`).
3. **Save** the request.

4.6.1 Step 5: Test Each Endpoint

1. **Select** each request in the collection.
2. **Replace** any placeholder like {`id`} with actual values as needed. Let's stick to the student with the id '1'.
3. **Click** 'Send' to execute the request.
4. **Observe** the response in the lower section of Postman and make a snapshot of the result of each request.
5. **Verify** that the response is as expected for each operation. Please also check your services in Mockapi.io and see how mock data are changed by each operation that you do. For example, after creating a new student it must shows 41 as the number of students.
6. Figure 9 shows how to get the list of students using Postman.

4.7 Deliverable

1. Add a snapshot of executing the request in step 4.2 to your answer sheet file.
2. Add a snapshot of executing the request in step 4.3 to your answer sheet file.
3. Add a snapshot of executing the request in step 4.4 to your answer sheet file.
4. Add a snapshot of executing the request in step 4.5 to your answer sheet file.
5. Add a snapshot of executing the request in step 4.6 to your answer sheet file.

5 Exercise D (Using APIs in webpages)

5.1 Objective:

Your task is to complete the JavaScript functions `fetchStudents` and `fetchCourses` in the provided `lab6_exe_D.html` file via D2L. These functions will interact with an API to fetch and display data about students and their courses.

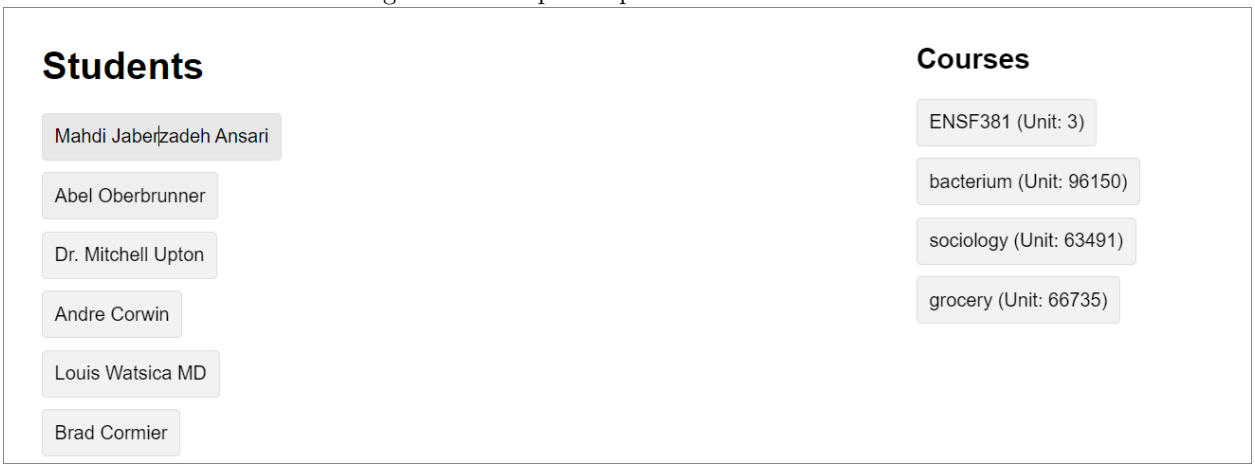
5.2 File Overview:

The `lab6_exe_D.html` file contains a basic web page setup with two main sections:

1. **Students List:** Displays a list of students.
2. **Courses List:** Shows courses for a selected student.

Figure 10 shows a sample output of the `lab6_exe_D.html` file, after pressing on the first student's name.

Figure 10: Sample output for lab6_exe_D.html



5.3 JavaScript Functions to Complete:

1. **Function `fetchStudents`:**
   - **Purpose:** This function retrieves a list of students from the API you created recently and displays them in the 'Students' section.
   - **How to Implement:**
     - Use `fetch` to make a GET request to `'https://[your-mockapi-url]/students'`.
     - Convert the response to JSON.
     - Iterate over the student data, creating a list item (<li>) for each student.
     - Set the text of each <li> to the student's name.
     - Add an `onclick` event to each <li> that calls `fetchCourses` with the student's ID.
     - Append each <li> to the `studentList` UL element.
2. **Function `fetchCourses`:**
   - **Purpose:** This function fetches and displays courses for a selected student in the 'Courses' section.

- **How to Implement:**
  - Use `fetch` to make a GET request to `'https://[your-mockapi-url]/students/${studentId}/courses'`.
  - Convert the response to JSON.
  - Iterate over the courses data, creating a list item (`<li>`) for each course.
  - Set the text of each `<li>` to the course name and unit.
  - Append each `<li>` to the `courseList` UL element.

5.4 Deliverable:

1. Once completed, save the `lab6_exe_D.html` file, commit it, and push it to your repository.
2. Make a snapshot of your sample output and attach it to your answer sheet file.

This assignment aims to help you understand how to fetch and manipulate data from an API using JavaScript and how to dynamically update the DOM based on user interactions and asynchronous data retrieval.