School of Electrical Engineering and Computer Science (SEECS)

Fundamentals of Computer Programming (CS-110)

Course Instructor: Ms. Momina Moetesum
Lab Engineer: Ms. Shakeela Bibi

School: SEECS
Semester: 1st

Dated: 30th December, 2024
Section: BESE 15-A

Final Deliverables (Github repository)

https://github.com/iPythonezta/StudentSync.git

#	Name	Reg No
1	**Huzaifa Azhar (TL)**	**522638**
2	Asim Shoaib	504888
3	Sheikh Abdullah Bin Zahid	532852

Project Aim: StudentSync

Scheduling conflicts are common among students who have to manage makeups, quizzes, and assignments. Someone from the class (usually CRs) have to manually compile schedules, assignments, and deadlines, missing something or forgetting about it altogether. Our project proposes a solution by developing an effective Student Management System to integrate all class schedules and tasks into one safe space.

This is not a simple calendar or planner project; instead, the platform focuses on centralizing and streamlining task management and academic workflows within a class. The system delineates two roles: Admins—(This can be CRs from the class)—and Users—(Students of the class). While Admins have special permissions to manage tasks and schedules, Users can view the information and interact with it. Our goal was to build a solid platform that ensures not a single deadline or crucial update is missed.

The project also addresses other common challenges faced by students, such as forming groups for assignments, tracking academic performance across various subjects, and organizing important notes and questions for different courses.

Key terminology

1. **JWT (JSON Web Token):** A compact, URL-safe means of representing data to be transferred between two parties. It ensures secure user authentication by verifying tokens for each session.
2. **RESTful API:** A set of web service protocols based on HTTP, allowing seamless communication between client and server.
3. **Crow Framework:** A modern C++ web framework designed for handling RESTful APIs and web applications efficiently.
4. **SQLite:** A lightweight, serverless database engine used for managing structured data in local or embedded environments.
5. **CSV (Comma-Separated Values):** A plain-text file format used to store tabular data (e.g., student marks, quiz questions).

6. **Role-Based Authorization:** A method of restricting system access based on user roles, ensuring users only access features relevant to their role.
7. **Aggregate Calculator:** A feature in our project that calculates students' overall performance based on marks entered for quizzes, assignments, and exams.
8. **Insomnia:** Insomnia is a powerful and user-friendly API testing tool which helps us to send HTTP requests such as GET, POST, DELETE, and others to RESTful endpoints, inspect responses, and validate API functionality.

Key Features

Our project integrates several essential features to support student and class management workflows:

1. User Authentication and Role Management:

- Accounts can be created with role-based authorization.
- Admins have access to additional features for managing schedules and student data.
- Users (students) have controlled access to view and interact with relevant information.
- Security is enforced using JWT (JSON Web Token) authentication assisted by SQLITE user database.

2. Task and Event Calendar:

- Admins can create, retrieve, and delete scheduled events and tasks, including assignment deadlines, quizzes, and makeups.
- Users can view these tasks and stay updated on important deadlines.
- Tasks and events are stored in the SQLite database.
- The Task and Event Calendar serves as the homepage after logging in.

3. Aggregate Calculator:

- Manage subject-wise marks for quizzes, assignments, mids, and final exams.
- Automatically calculate aggregates based on predefined weightages.

- ◆ Marks entered are stored in CSV files for future use.|

4. **Group Formation:**
   - ◆ Automatically form student groups for collaborative projects.
   - ◆ Uses a grouping system where students give their preferences, and the system generates optimal groups.

5. **Quiz Bank Management:**
   - ◆ Admins can store and manage questions and answers in .csv files.
   - ◆ Facilitates the creation of quizzes for exam preparation.

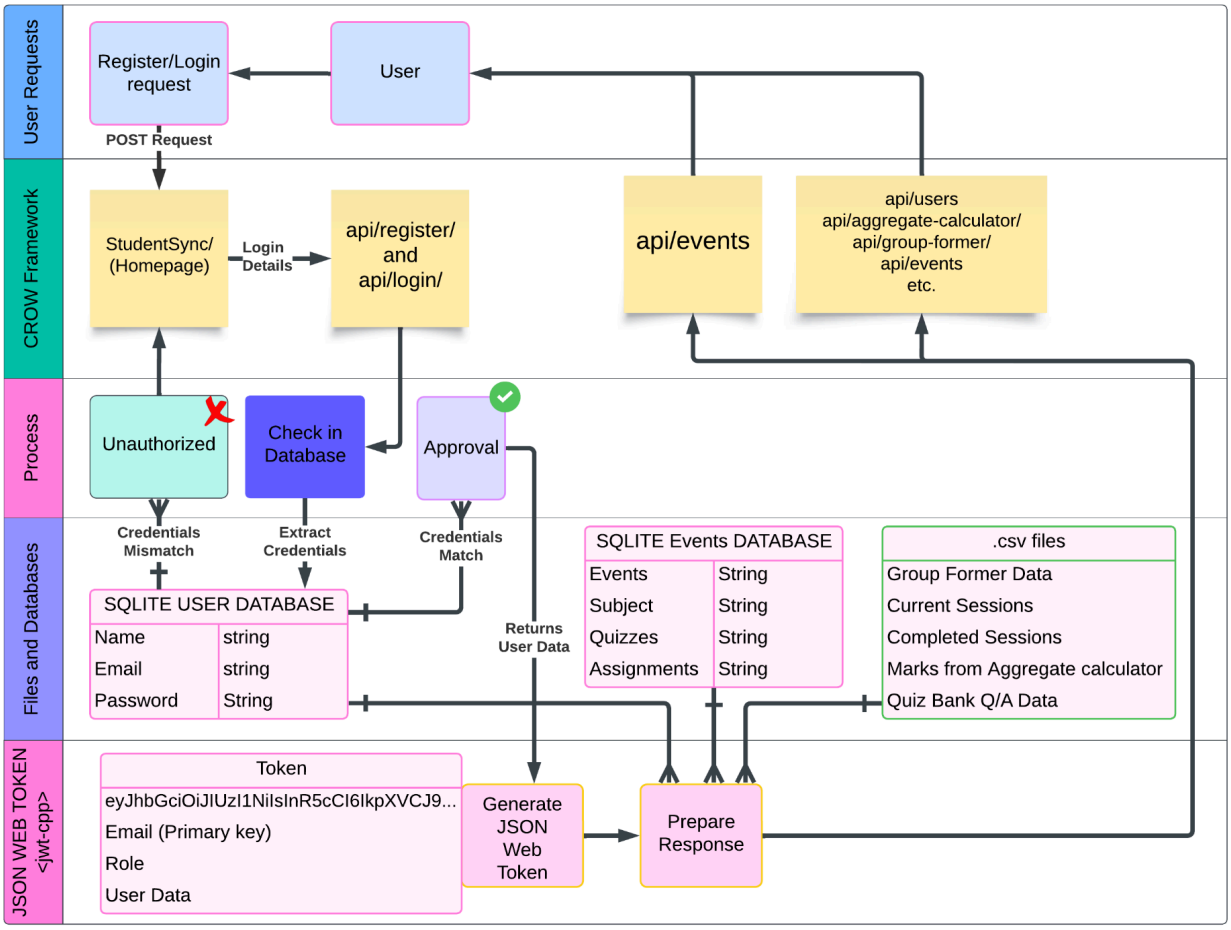
6. **File Handling:**
   - ◆ Stores and retrieves student marks, preferences, and quiz data using CSV files.

7. **Web API with Crow Framework:**
   - ◆ Provides a RESTful interface for interaction between the front-end and back-end.

Contrasting final project with the High-Level Design report:

Feature	Design in High-Level Design Report	Final Implementation	Notes
Authentication & Authorization	- Two roles: Admin and User. - Admins can manage users and system settings. - Users can access tasks and quizzes.	- Implemented with JWT for secure authentication. - Roles and access restrictions implemented.	Matches the design.
Task Calendar & Alerts	- Central calendar for deadlines. - Notifications for upcoming tasks via email or push.	- Calendar is implemented as the homepage. - Tasks and events stored in SQLite. - No notification system.	Core feature matches, but notifications are absent.
Quiz Bank	- Shared repository for quiz questions. - Students can view and submit questions. - Admins manage the bank.	- Quiz questions handled via CSV files. - Students contribute questions, and admins approve them.	Implemented but relies on CSV files instead of a database.
Group Former	- Admin-initiated group sessions. - Students submit preferences. - Algorithms handle grouping.	- Group formation feature works as designed. - Preferences and groups stored in CSV files.	Matches functionality, but data is stored in CSV files instead of SQLite.
Aggregate Calculator	- Students input marks for activities. - Admins set weightages. - Aggregates are calculated.	- Aggregates calculated with marks and weightages. - Data stored and retrieved using CSV files.	Functionality matches, but uses both CSV files for storing marks and SQLite database for storage of weightages.
Data Design	- SQLite database for users, tasks, quizzes, and group data.	- SQLite used for users and tasks. - CSV files used for quizzes, marks, and groups.	Partially implemented, with mixed use of SQLite and CSV files.
Notification System	- Notifications for upcoming deadlines.	- Not implemented.	Missing feature.
Frontend Integration	- Dynamic frontend using React.js.	- Frontend implemented with basic HTML and CSS.	Simplified implementation.

Backend Working



The Program Works in layers:

1. User Requests and API Endpoints

Diagram:

➔ User actions (Register/Login) are sent as POST requests to endpoints like /api/register/ and /api/login/.

➔ Other requests interact with endpoints such as /api/events/, /api/aggregate-calculator/, and /api/group-former/.

Code:

➔ The program defines RESTful API endpoints using the Crow framework. Our backend server is designed to listen for requests at these endpoints. Upon receiving a request, the server processes it and returns the relevant response along with the data (if necessary).

➔ Specifically:

- **Register/Login:** Handlers are defined for POST /register and POST /login, where user credentials are checked or saved in the SQLite database.
- **Other endpoints:** Handlers like /api/events and /api/group-former etc, manage events and group formation.

---

2. Role of SQLite Databases

Diagram:

➔ One database containing:

- **USER DATA:** Stores name, email, and password for authentication.
- **Events DATA:** Tracks events, subjects, quizzes, and assignments.
- **Subject DATA:** Keeps record of all the subjects that the class is studying. It stores information like the Subject Name, Credit Hours,

➔ Data includes:

- Marks stored and manipulated for aggregate calculation.
- Event and task management records.

Code:

➔ The program uses SQLite databases:

- sqlite3_exec function is used for executing SQL statements to support CRUD operations on tables storing user credentials and events.
- Event details such as quizzes and assignments are extracted and saved using this function.

➔ Marks from the aggregate calculator and group formation data are stored in .csv files, as depicted.

---

3. Authentication and Authorization

Diagram:

- ➔ The backend server generates a unique JSON Web Token (JWT) as a response to a successful login request.
- ➔ Token is generated by encoding the user's email and role (admin/normal) using cryptographic algorithms and a secret key.
- ➔ To interact with the application further, the user is required to **pass the JWT** in the **Authorization header** of each subsequent request. The token is typically sent as a Bearer token, like this:

  Authorization: Bearer <Server Generated Token> (The frontend Interface takes care of this)

- ➔ The backend server, upon receiving the request, **decodes and verifies** the token using the same secret key. This allows the server to authenticate the user's identity and role. If the token is valid, the server grants access to the requested resource based on the user's role (e.g., admin or normal).
- ➔ Invalid credentials lead to an "Unauthorized" state.

Code:

- ➔ JWT is implemented via a library that generates tokens during login
- ➔ Tokens encode:
  - ◆ Email as the primary key.
  - ◆ User role (admin or user).
  - ◆ Relevant user data.
- ➔ Login flow verifies credentials against the SQLite USER DATABASE. Incorrect matches result in an "Unauthorized" respo

---

4. Homepage and Task/Event Management

Diagram:

➔ After login, users land on the **StudentSync Homepage** displaying tasks and events.
➔ Tasks and events are queried from the database and presented based on user permissions.

Code:

➔ Event and task management functionality is implemented.
➔ Admins can add, update, or delete events, while users can only view them.
➔ Task and event data is retrieved from the SQLite Events DATABASE for display.

---

5. Group Former and Aggregate Calculator and Quiz Bank etc.

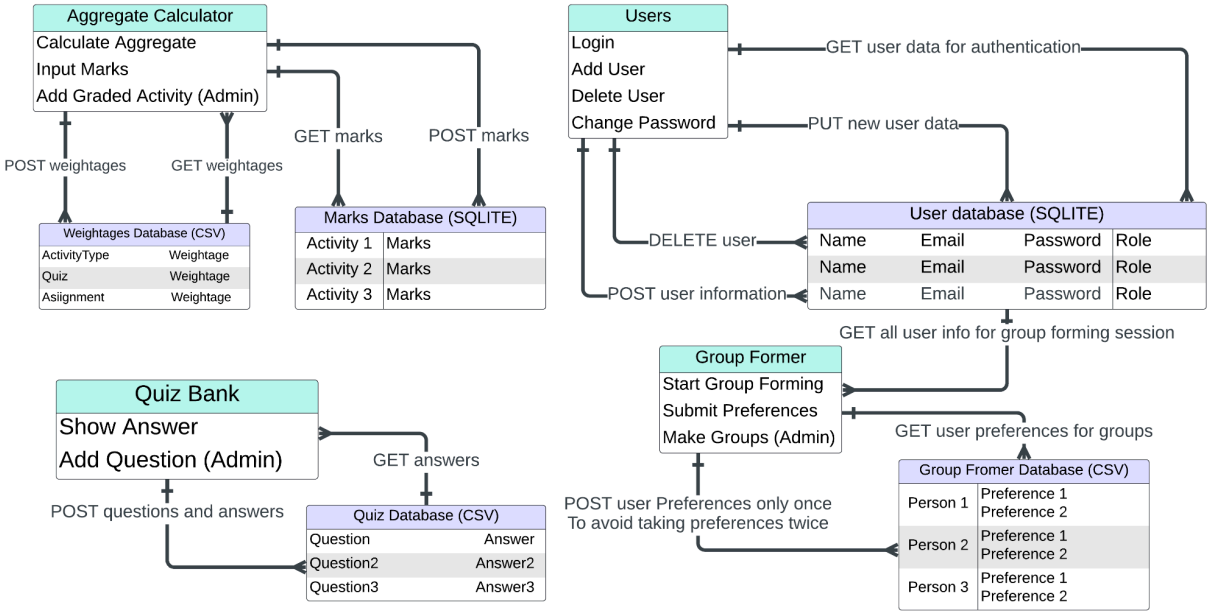
Diagram:

➔ Group formation, Quiz bank and aggregate calculation features are accessible via APIs.
➔ Marks and preferences are stored in .csv files.

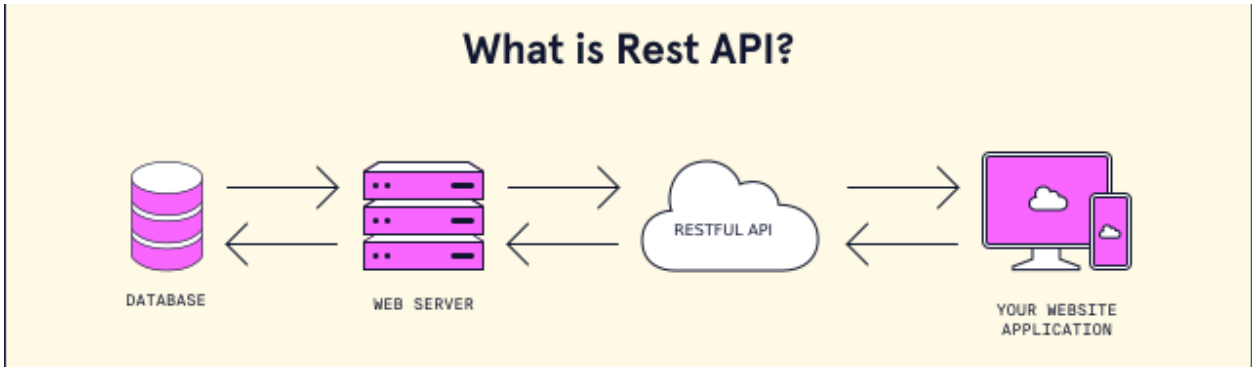
Code:

➔ Group formation uses algorithms based on user preferences.
➔ The aggregate calculator computes weighted scores and writes them to .csv files.

Data sources and destinations for Endpoints:



What an API endpoint actually does:

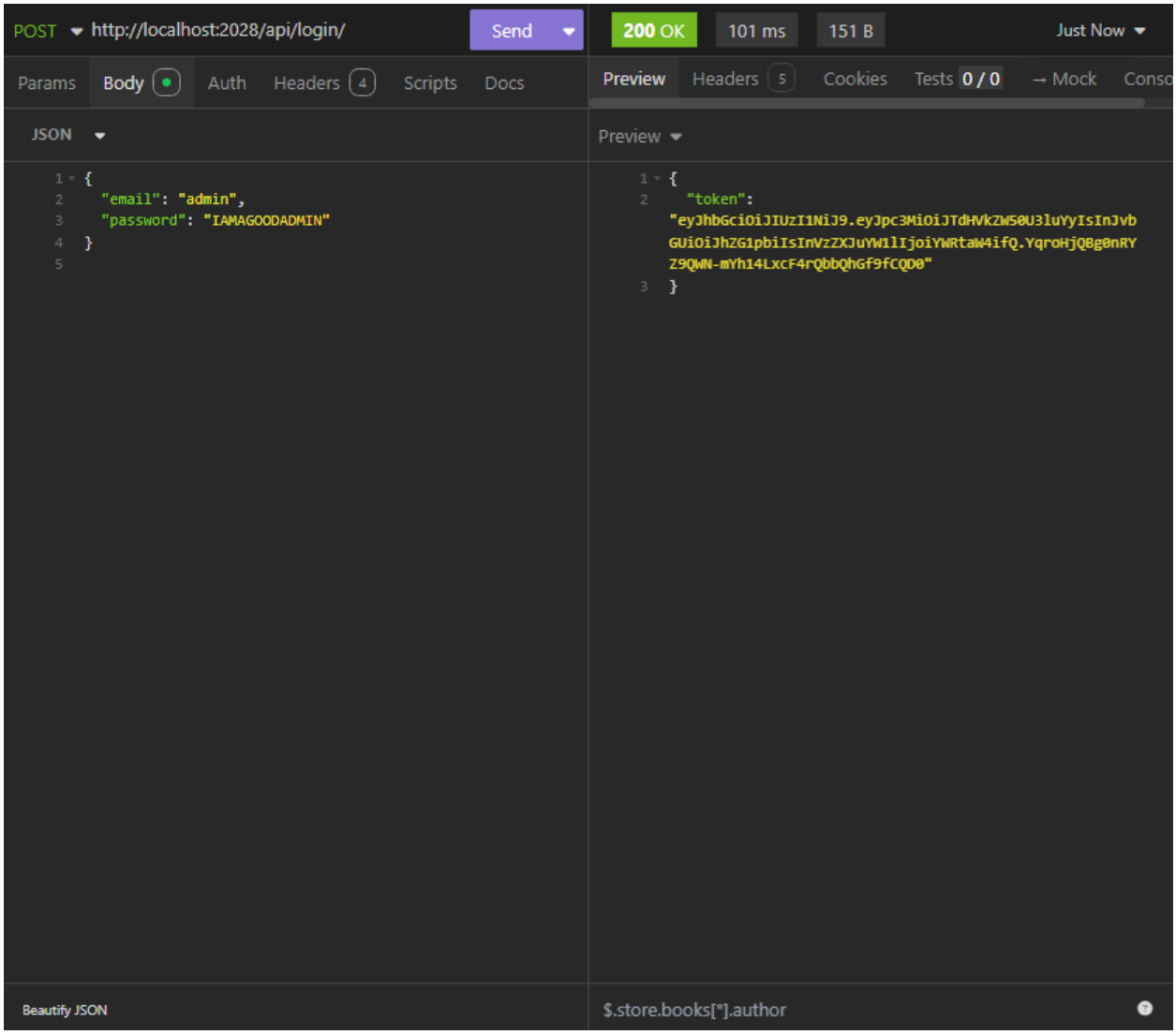


Testing the Endpoints coded in the main.cpp file

POST Endpoints:

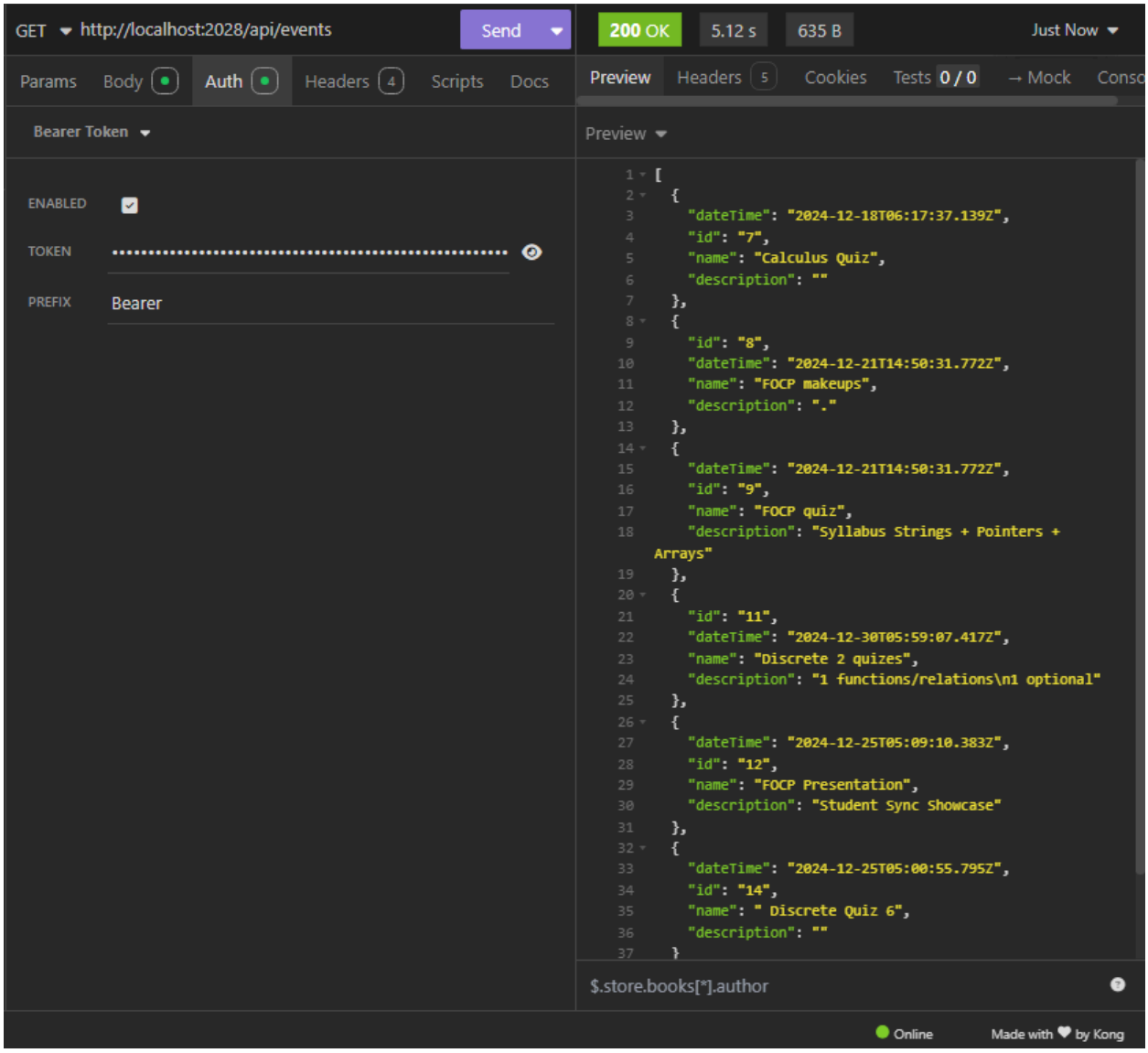
➔ **User Management:**

- ◆ /api/login/ - User login and token generation.
- ◆ /api/users/remove-user/ - Delete a user from the system.
- ◆ /api/users/make-admin/ - Promote a user to the admin role.



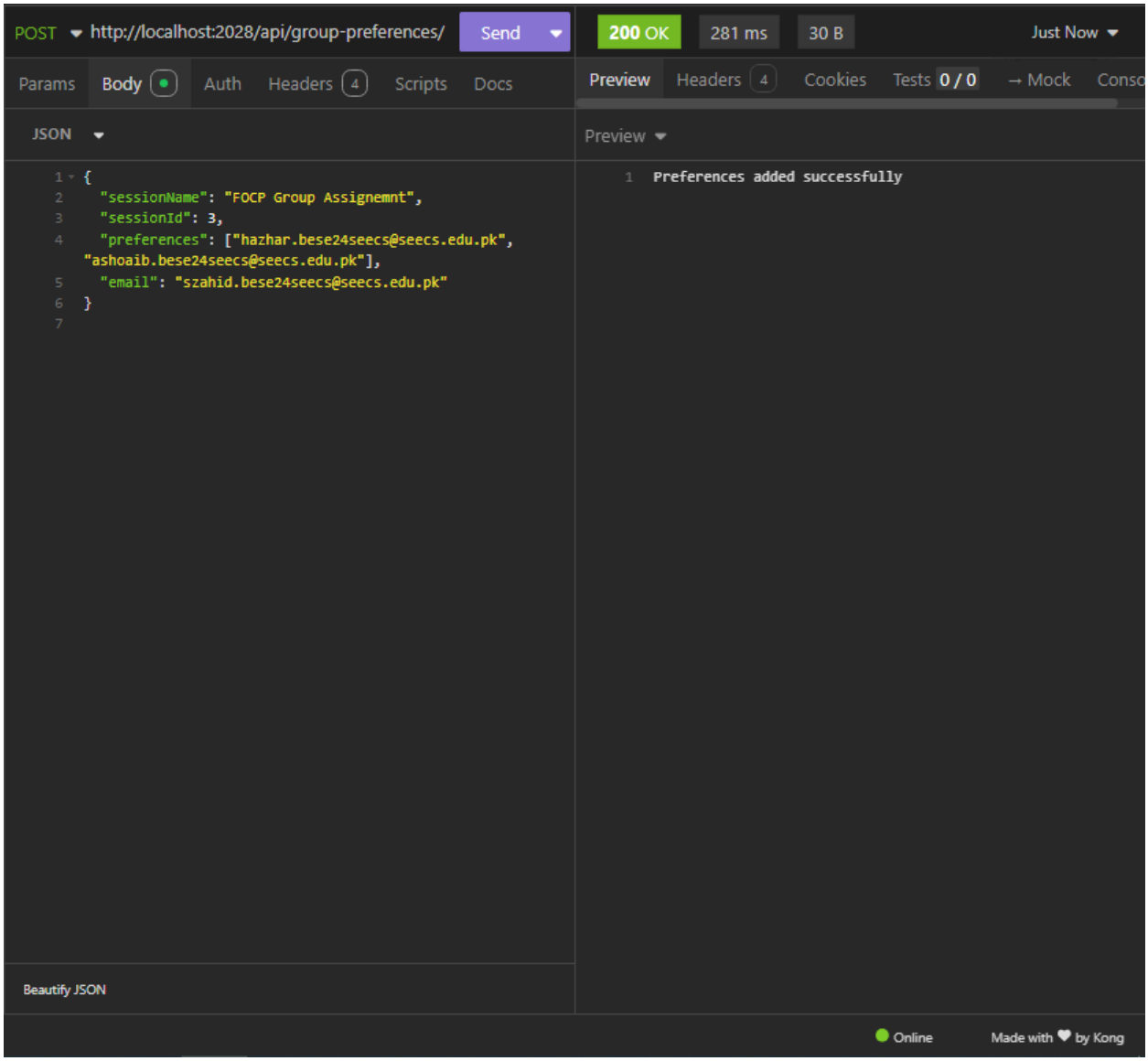
➔ **Events and Tasks:**

- /api/events/ - Create a new event.
- /api/subjects/<int>/add-task/ - Add a task (quiz, assignment, etc.) to a specific subject.



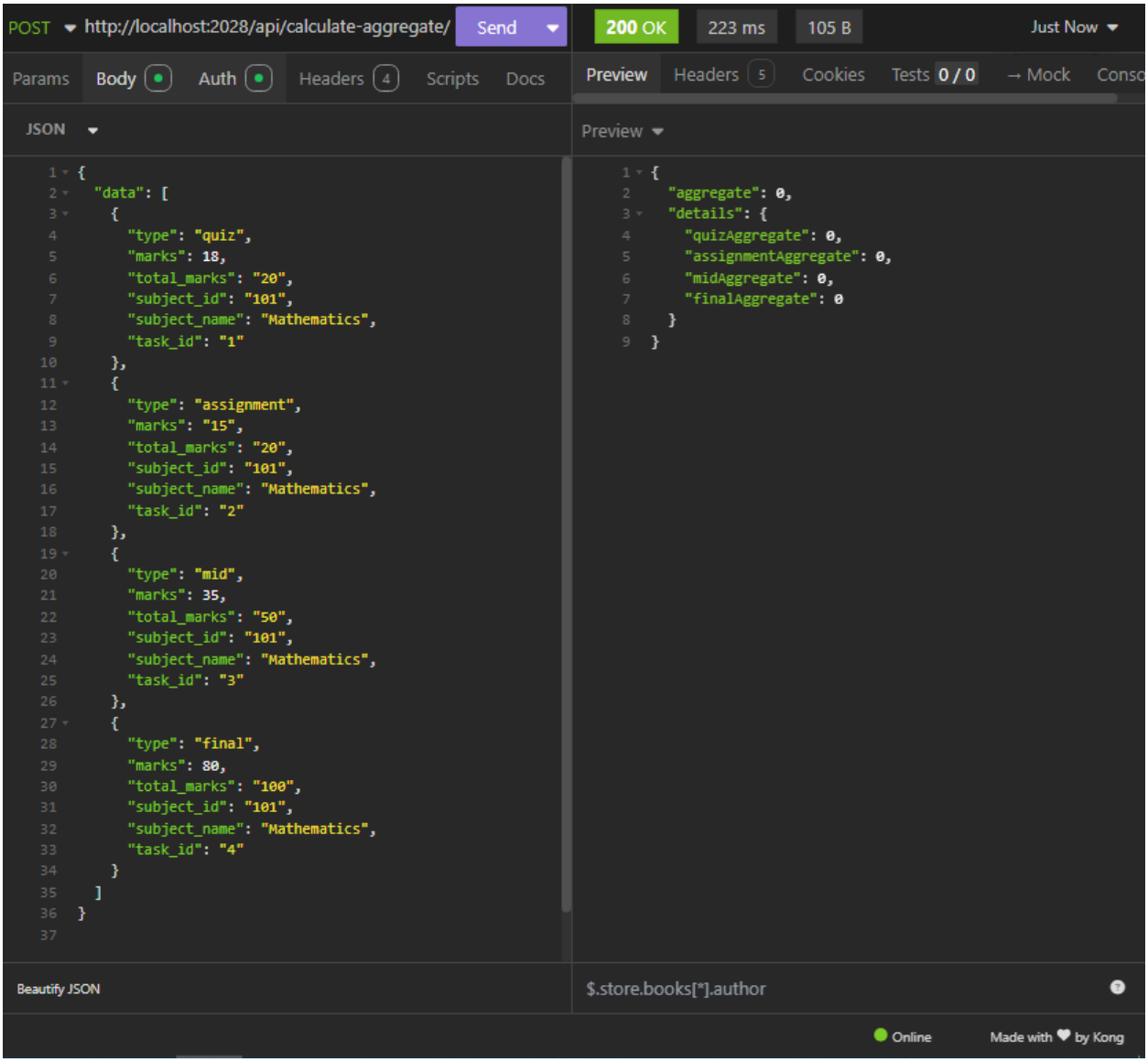
➔ **Group Management:**

- ◆ /api/group-preferences/ - Save group preferences for a session.
- ◆ /api/form-groups/ - Form groups based on user preferences.
- ◆ /api/grouping-sessions/ - Create a new grouping session.



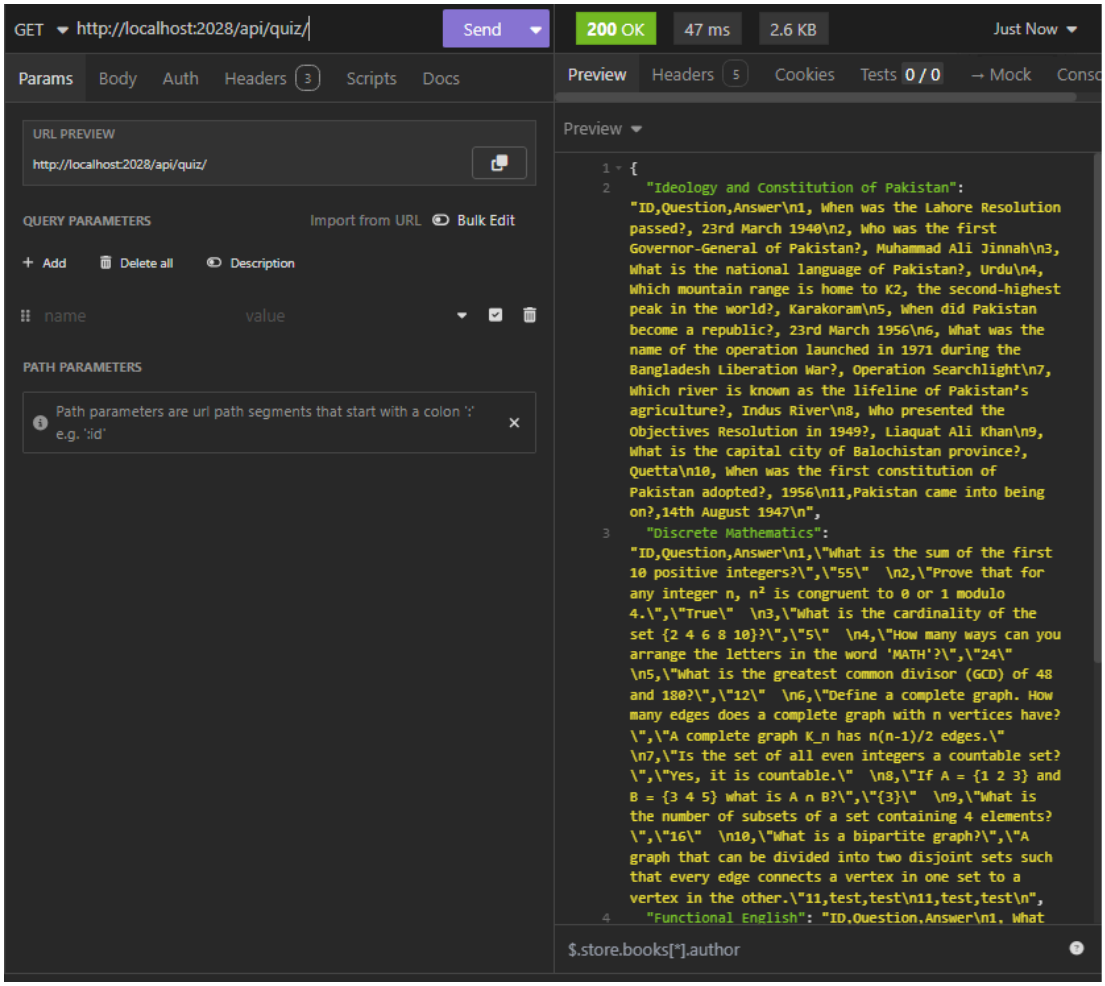
➔ **Aggregate Calculator:**

- ◆ /api/calculate-aggregate/ - Calculate student aggregate marks.



➔ **Quiz Bank:**

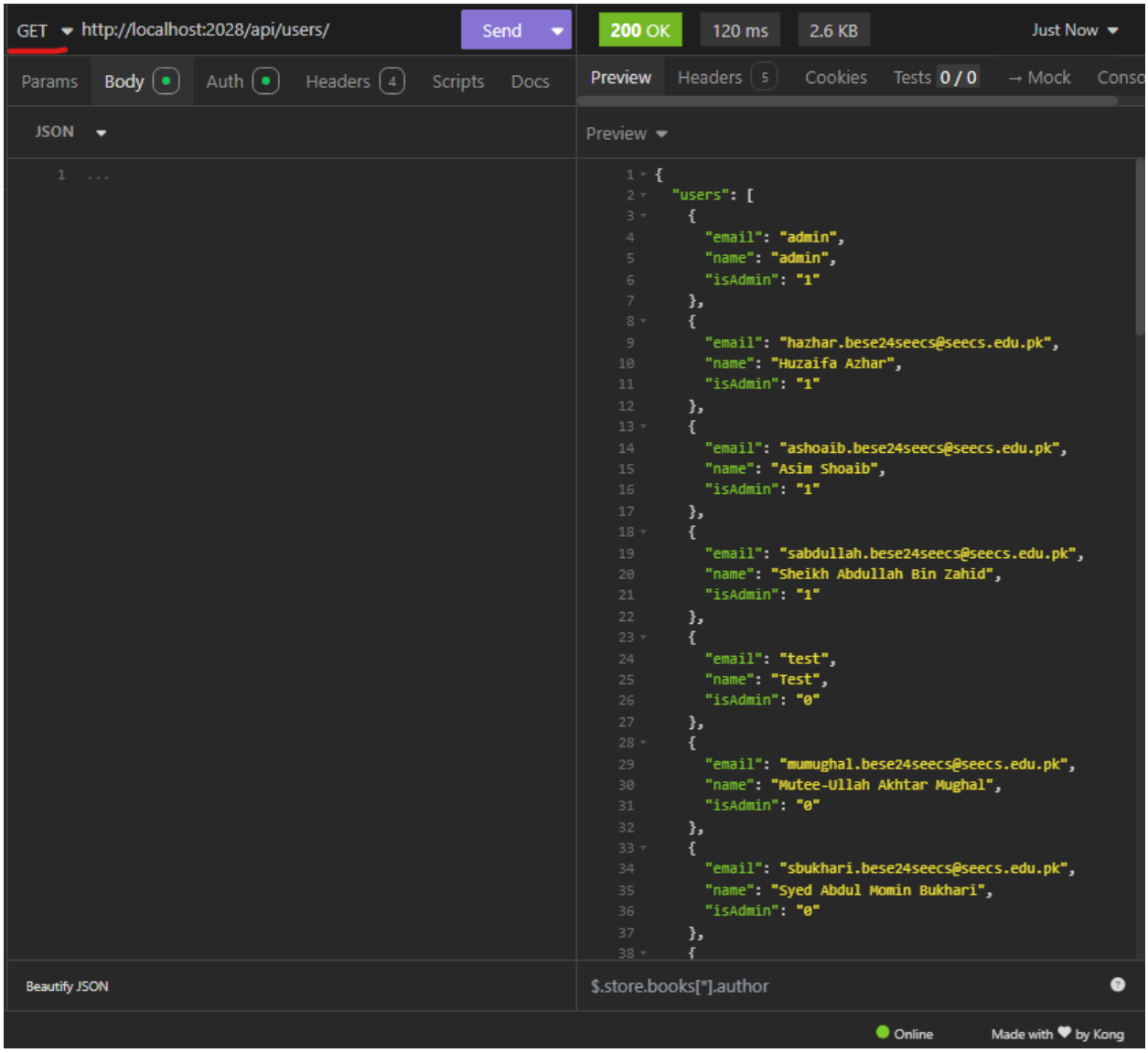
- ◆ /api/quiz-bank/ - Adds an ‘id, question, answer” 3-tuple into the database.



GET Endpoints:

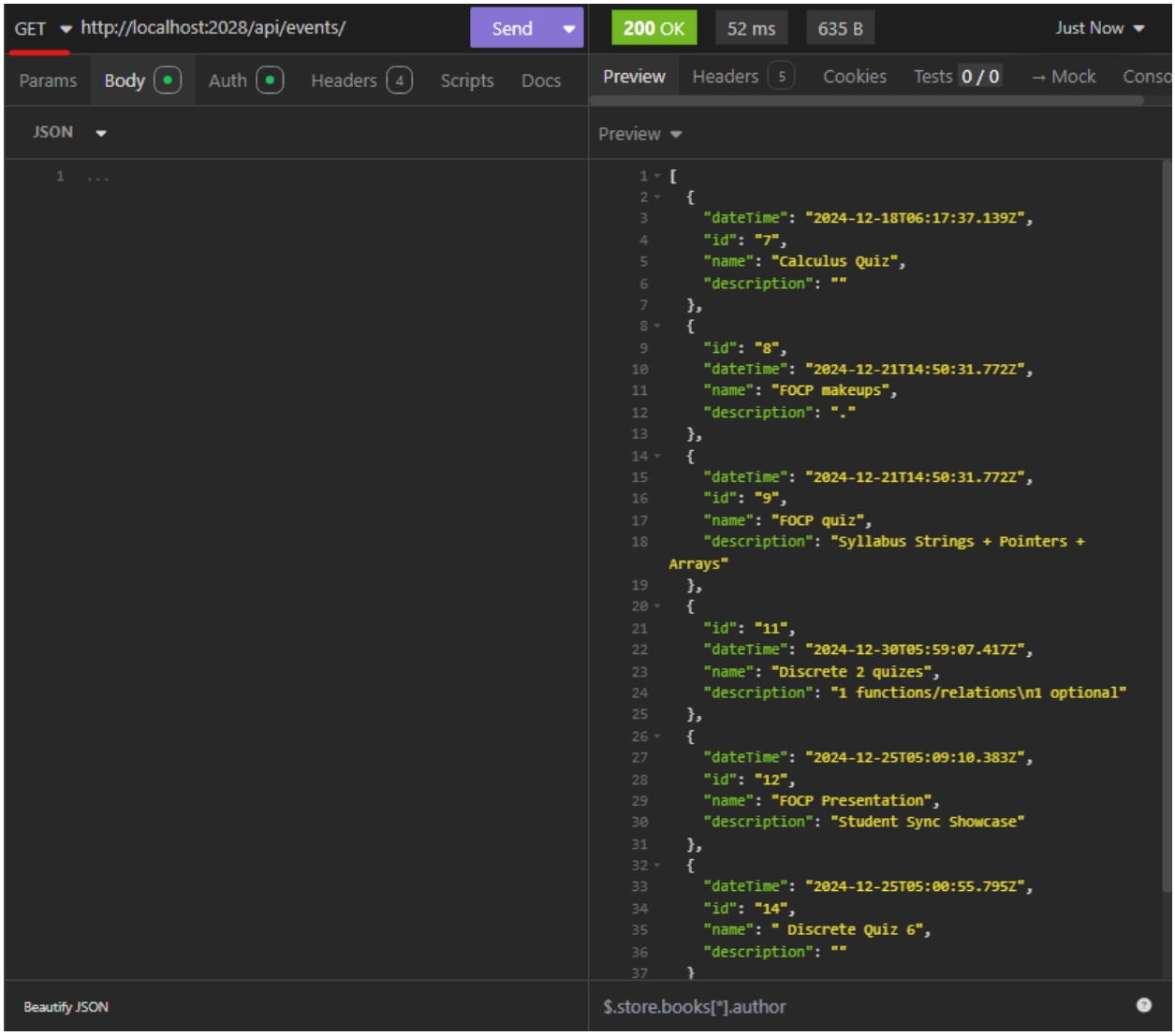
➔ **User Management:**

- /api/user/ - Retrieve logged-in user's details.
- /api/users/ - Retrieve all user details.



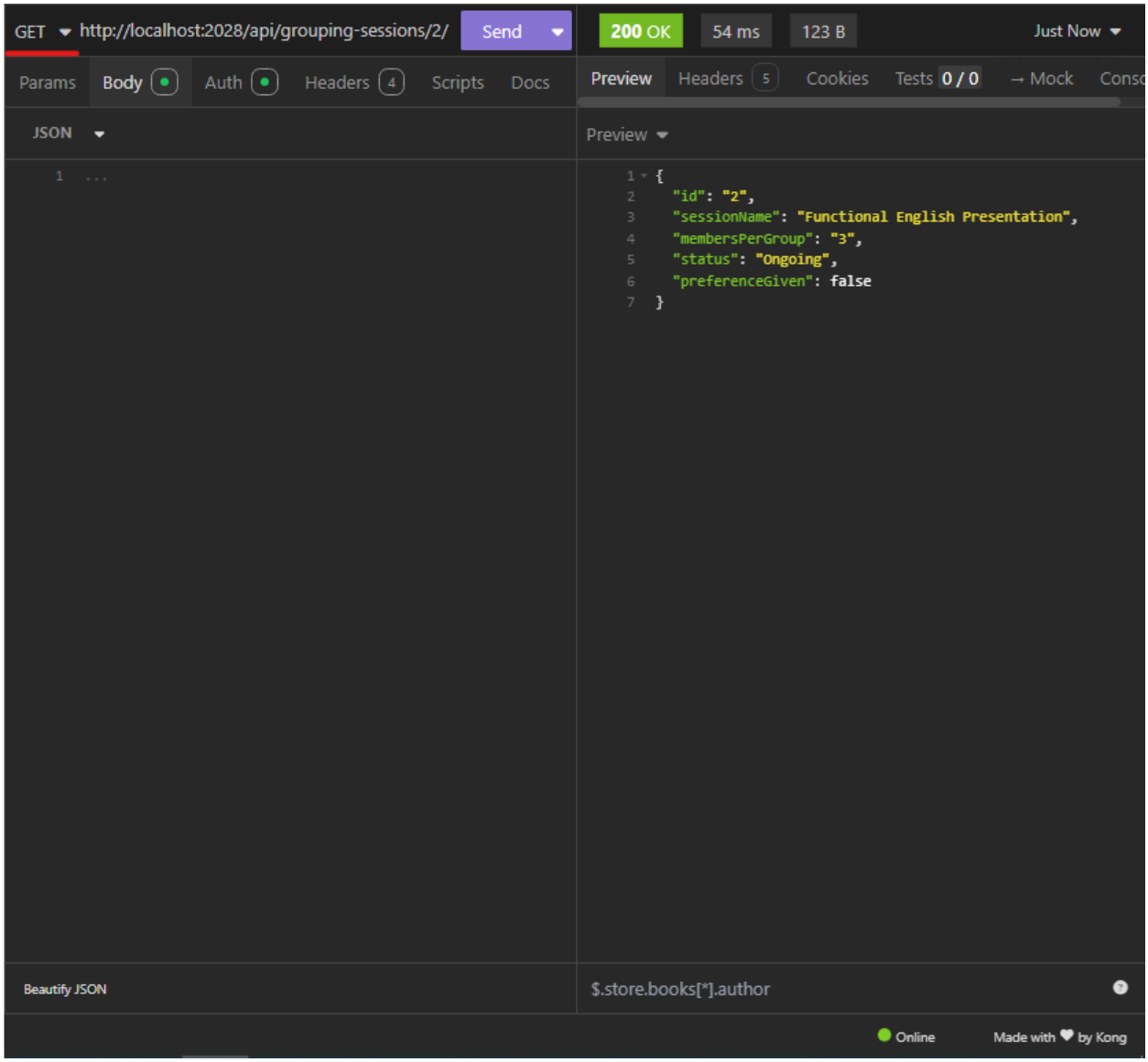
➔ **Subjects and Events:**

- /api/events/ - Retrieve all events.
- /api/subjects/ - Retrieve all subjects.
- /api/subjects/<int>/ - Retrieve specific subject details, including tasks (quizzes, assignments, etc.).



➔ **Group Management:**

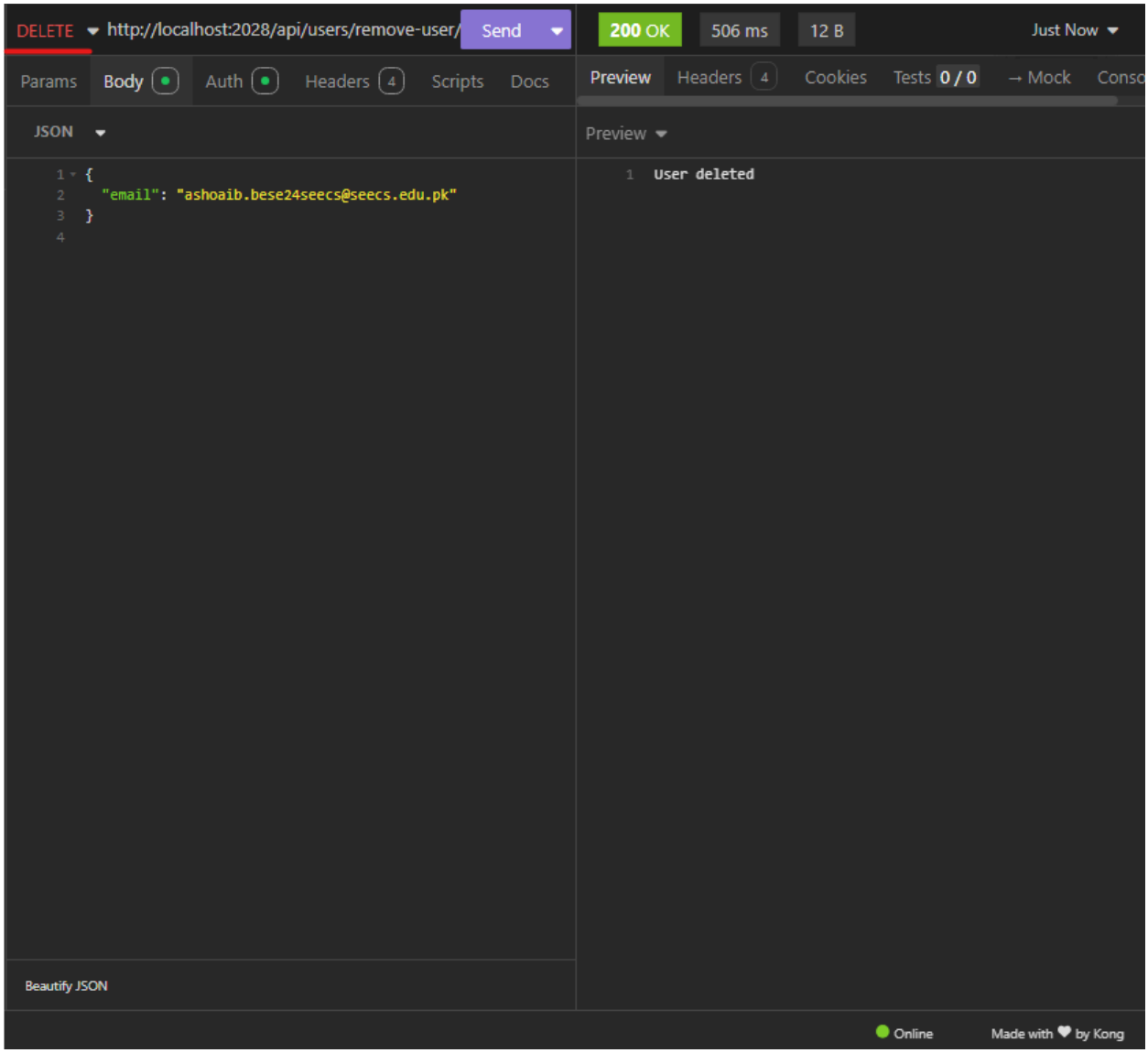
- ◆ /api/grouping-sessions/ - Retrieve all grouping sessions.
- ◆ /api/grouping-sessions/<int>/ - Retrieve details of a specific grouping session.



DELETE Endpoints:

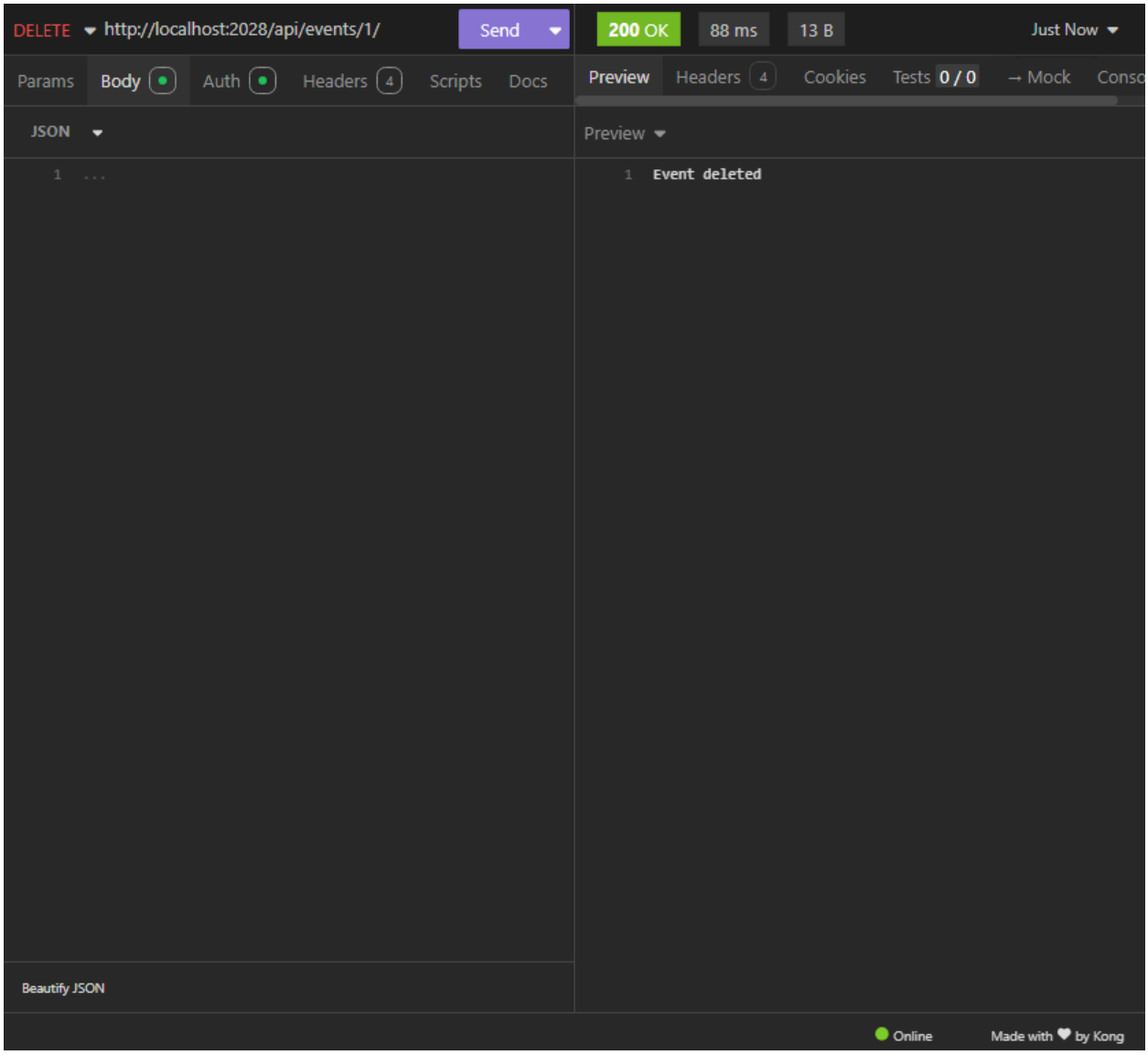
➔ **User Management:**

- ◆ /api/users/remove-user/ - Delete a user from the system.



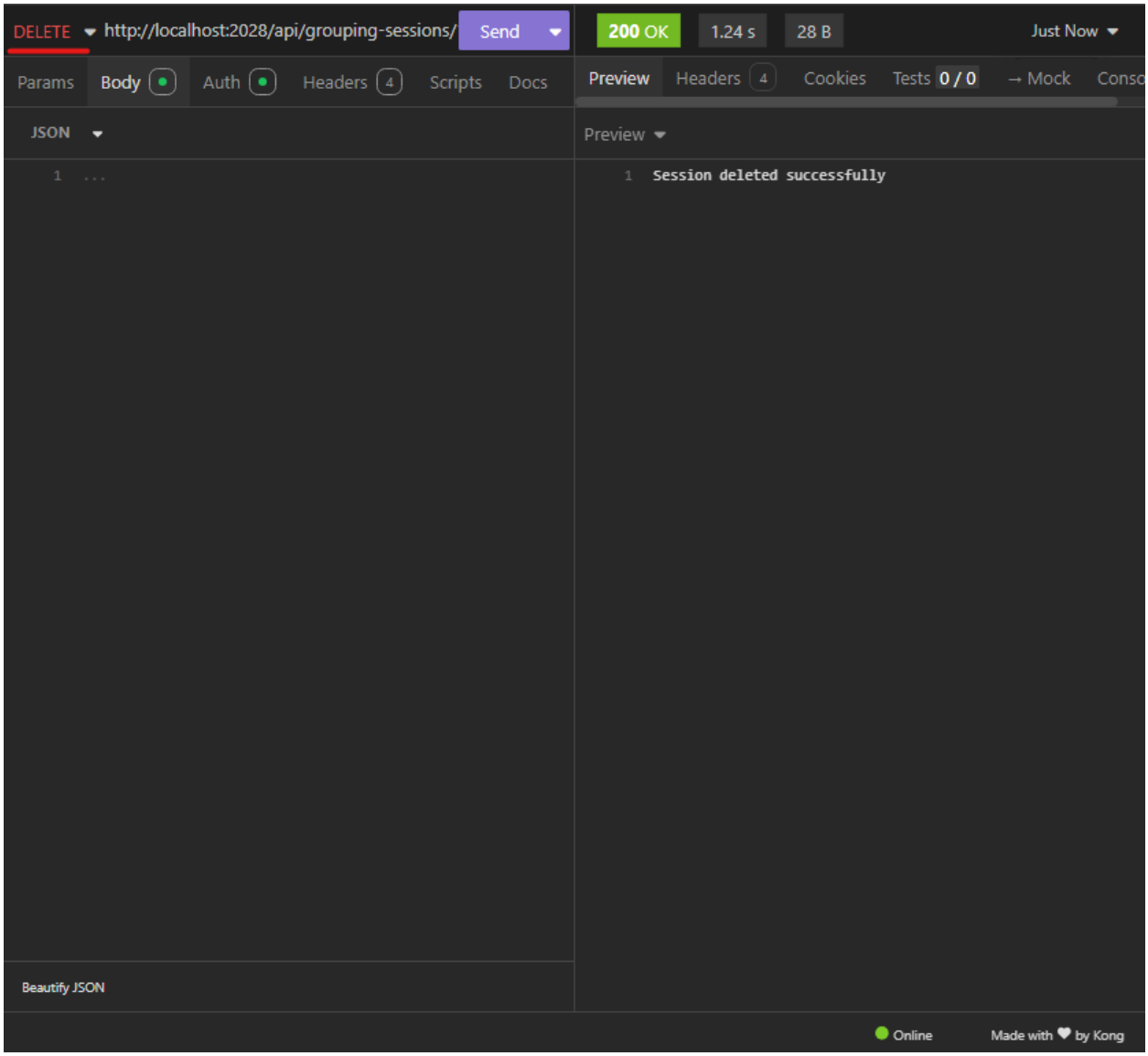
➔ **Subjects and Events:**

- /api/subjects/<int>/ - Delete a specific subject.
- /api/subjects/activity/<int>/ - Delete a specific task (quiz, assignment, etc.) from a subject.
- /api/events/<int>/ - Delete a specific event.



➔ **Group Management:**

- ◆ /api/grouping-sessions/<int>/ - Delete a specific grouping session.



Integration of the frontend with Backend

Our web application's frontend is built using **JavaScript**, with **React JS** as the primary framework. For handling user actions, the frontend communicates with the backend using the **axios library**. The results of these actions are dynamically reflected in the user interface.

For example, when a user opens the application, they are directed to the login page. Here, they fill out a form with their credentials. Upon clicking the submit button, the frontend sends a **POST request** to the backend (**/api/login/ endpoint**), including the username and password as parameters. The backend validates these

credentials and, if they are correct, responds with a **token** and a **success code**. The frontend detects the success code, stores the token in memory, and redirects the user to the **Dashboard**.

On the Dashboard, the frontend sends a **GET request** to the backend to retrieve all scheduled tasks and other user-specific data. This time, the stored token is included in the request's **Authorization header**, allowing the backend to authenticate the user and provide the necessary data.

Success, Challenges and Future Improvements

Successes:

Our overall project turned out to be a big success, where we were able to successfully implement most of the functionalities as planned. We were also able to achieve more than **90%** of the ***functional and non-functional requirements*** that we had defined for the project.

What we achieved:

Functional requirements

Here are the details of the functional requirements that we had defined, and how much were we able to achieve them.

1. User Authentication and Authorization

- Users are required to log in to access the system.
- The system differentiates access levels for admins and regular users to ensure appropriate permissions.

2. Task Calendar Management

- Users can view a calendar displaying deadlines and enable notifications for reminders. (**Partially implemented)**
    - Admins have the ability to add, or delete calendar entries as needed.

3. Group Formation

- Admins can initiate group-building sessions by specifying assignment details and desired group sizes.

- Students are allowed to submit team member preferences during these sessions.
- The system employs algorithms to form groups based on preferences or assigns users randomly if no preferences are submitted.

4. Quiz Bank Management

- Admins can add new quiz questions, as well as approve or reject submitted questions.
- Students can view the quiz bank and contribute by submitting their own questions for consideration. **(Feature Dropped)**

5. Aggregate Tracker

- Students can input marks for various activities and monitor their aggregate scores over time.

6. Database Operations

- CRUD operations (Create, Read, Update, Delete) are implemented to manage data such as users, quizzes, group preferences, and activities.
- The system utilizes SQLite as the database for efficient data storage and retrieval.

Non-Functional requirements

Here are the details of non functional requirements that we had defined earlier, and how much were we able to achieve them

1. Cross-Platform Compatibility

- The application is designed to operate seamlessly across multiple operating systems, including **Windows**, **Linux**, and **macOS**.

2. Cross-Browser Compatibility

- The application ensures smooth functionality on major browsers such as **Chrome**, **Edge**, **Safari**, and **Firefox**, without encountering significant issues.

3. Usability

- The user interface is intuitive and user-friendly, allowing users to easily navigate and perform tasks without confusion or difficulty.

4. Reliability

- The system maintains **database integrity** even during concurrent read and write operations, ensuring reliable performance under load. **(CSV databases do mess up sometime but overall it's ok, especially when it is supposed to be used by only the students of one class)**

Challenges Faced

As with any programming project, we encountered numerous challenges throughout the development of this project. While these challenges were often frustrating, they also proved to be rewarding and fun, as they pushed us to think critically and creatively. Below, we detail the obstacles we faced and the strategies we employed to overcome them successfully.

1. Identifying the right libraries

Once we finalized our project idea, our first challenge was selecting a reliable yet user-friendly library for coding a web server in C++. This proved to be somewhat difficult, as C++ is not a language commonly used for web development, limiting our options. After some research, we discovered Crow.h, which appeared to align well with our requirements. The example code provided on its website was both straightforward and easy to understand for us. So we decided to use Crow.h as our backend framework.

2. Installing the required libraries

Before this project, we had never installed an external C++ library, so we had no idea about the challenges waiting for us. While following the installation instructions on Crow.h's official website, we came across **vcpkg**, which they recommended for installing the library on Windows machines. None of us had ever used **vcpkg** before, so we did what any other student in our place would do—Google it.

That led us to its website and its own installation instructions. But installing vcpkg wasn't as straightforward as one might hope. Instead of a simple .exe file, we had to build it from source. This meant cloning its GitHub repository and then

following the instructions to compile it ourselves. It took us several hours of debugging, but we finally managed to install it. Once set up, vcpkg turned out to be incredibly useful, saving us hours of effort when installing other libraries and managing dependencies.

3. Learning the new libraries

Learning the syntax of libraries like **Crow.h** and **SQLite3.h** also proved to be a significant challenge for us. There were many instances where we found ourselves scratching our heads, either because we wanted to implement something with the libraries, but didn't know how, or because we were dealing with random bugs that popped up in our program out of nowhere.

For example, at one point, I needed to create an array of **crow::json::wvalue** objects. I spent a considerable amount of time trying to figure it out but couldn't find a solution, until I stumbled upon a GitHub thread: https://github.com/ipkn/crow/issues/265. That thread provided the guidance I needed to resolve the issue.

Throughout the project, we encountered numerous errors and challenges, but resources like GitHub, Stack Overflow, and GPT often came to our rescue, helping us overcome the obstacles.

4. **Last Minute Debugging**

The night before our final project presentation, one of our teammates pushed some changes to our GitHub repository, mostly formatting tweaks and added comments. [Commit Link: https://github.com/iPythonezta/StudentSync/commit/df11abd256fb6a96413bffd475b46f0852a07364].

With just 30 minutes left before our presentation, we decided to run the program to ensure everything was in order. To our shock, two key features—Quiz Bank and Aggregate Calculator—weren't working, even though they had been functioning perfectly the previous night. With no time to debug the entire code, we quickly turned to Git for help. Using the **`git checkout`** command, we reverted to the previous version of the code, and the errors disappeared.

This saved the day, and we were able to deliver a successful demo. Later, we discovered that while adding comments, our teammate had accidentally deleted some critical lines of code, causing the issues.

Future Improvements:

In the future we plan to integrate several improvements into our project. Here are some of the improvements that we have planned and are already working on.

1. Responsive mobile design – We will be making our frontend responsive, ensuring that it can be used easily on mobile devices as well
2. Aggregate Calculator Improvements– Currently the aggregate calculator doesn't support courses that have a lab component as well. In future we are looking to integrate this feature as well, where the aggregate of courses with lab components can be calculated easily as well.
3. Quiz Bank Improvements– Quiz bank can be made more functional by allowing normal users to submit questions as well. Additionally, we are considering integrating a "Create a Test" option, where users can generate quizzes using the questions that have already been uploaded. This will help them in preparing for exams more effectively.

Achieved Learning Outcomes

We learned a lot during the course of this project and here is a reflection on the learning outcomes that we achieved.

1. Understanding of the C++ language, its different programming constructs and data structures

Throughout this project, we wrote a significant amount of C++ code, which deepened our understanding of the language. We implemented various algorithms and functionalities using different constructs, enhancing our knowledge. We also encountered new data structures that we hadn't previously worked with, such as vectors, unordered_maps, and unordered_sets. Most of the time, we discovered these structures when faced with problems that required them. Learning them through hands-on experience turned out to be a valuable and rewarding process.

2. Solving Real World Problems.

Working on this project greatly enhanced our understanding of C++ and its application in solving real-world problems. While we had used loops and conditional statements throughout the course, they were mostly focused on printing random patterns or numbers, which didn't seem to have practical use at the time. However, this project changed that. Our loops were now performing meaningful

tasks, each with a specific function. This experience showed us how the concepts we learned during the course could be applied to solve real problems.

3. Building a program and its documentation using associated IDES and supplementary tools

This project also increased our knowledge about the IDEs we were using. E.g we learned to integrate the r **GIT** version control with Visual Studio and Visual Studio Code. Especially learning to merge our codes using these IDEs and Git was a really valuable skill that we learned.

Similarly we also learned about documenting our code using code comments as well as by maintaining a separate documentation. We also made a README file for our repo using markdown language, which is considered to be a must have documentation for any project

4. Performing effectively as a member of a team.

This project required all three of us to collaborate effectively on different tasks which played a role in developing the invaluable skill of working together as a team. We worked together on campus and collaborated online on Google Meets and Whatsapp calls as well.
Had we failed to work as a team, this project wouldn’t have been possible.