SENG 401
FINAL REPORT – StudySync
TEAM 38

Team Members:
Adhil Ashraf, Anfaal Mahbub, Al Farhana Siddique, Arvind Krishnaa, Mehvish Shakeel, Varshiny Gogulanathan

Purpose:

The primary goal of StudySync is to empower students in their academic journeys by providing a comprehensive platform designed to optimize their study routines, foster collaboration, and support their mental well-being. By integrating various features, StudySync aims to address the challenges students face in managing their academic responsibilities and navigating their courses effectively.

Context:

StudySync is developed based on extensive research into the challenges students encounter in their academic pursuits, as well as feedback gathered from prior related work and direct input from students. Through this process, key areas where students could benefit from additional support and tools have been identified and targeted for improvement.

Features such as the ability to create, edit, and delete posts tailored to specific courses are carefully crafted to meet the unique needs and pain points of students. This flexibility allows users to engage with peers, seek assistance with coursework, form study groups, and share valuable insights, thereby fostering a supportive and collaborative academic environment.

The overarching objective of StudySync's design is to enhance student satisfaction, promote academic success, and improve retention rates. By focusing on delivering a user-centric experience and continually refining the platform based on user feedback and emerging needs, StudySync aims to serve as a valuable resource for students throughout their academic journeys.

Detailed Design:

Initial Plan

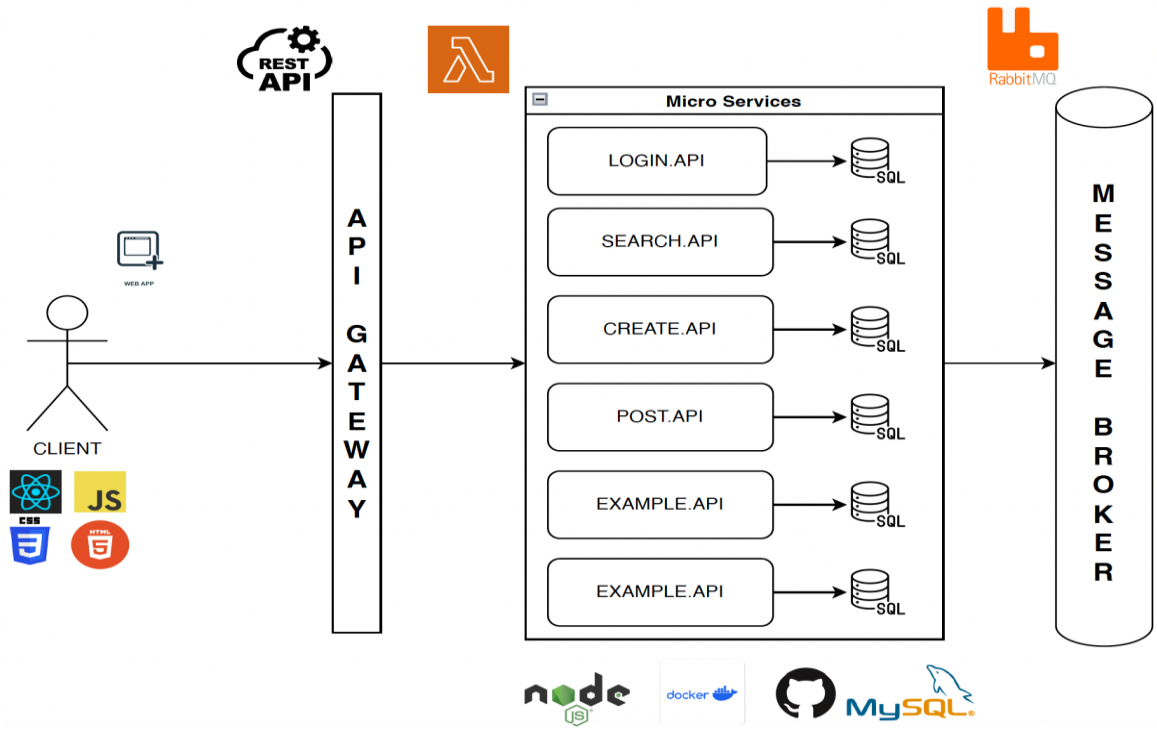
Overall System Architecture:
Our StudySync will adopt a microservices-based architecture, providing modularity, scalability, and maintainability. The front-end will be developed using JavaScript and React.js. At the back-end, we will use Node.js with Express.js and MySQL for structured data storage. Real-time updates will be facilitated through a Pub/Sub pattern using a message broker (e.g., RabbitMQ). We also plan to integrate third-party APIs for mental health tips and room availability.

User Interaction:
Users will access StudySync through a web interface. The platform will offer features such as an exam date scheduler, study group finder, mental health tips, course finder/feedback, and room finder. User authentication and authorization will be implemented using Passport.js.

Software Components Interaction:
The front-end components will communicate with back-end microservices through a RESTful API. Real-time updates will be achieved through the Pub/Sub pattern.

Data Flow:
User inputs and requests will flow from the front-end to the back end for processing and storage in the MySQL database. Real-time data updates, such as study group notifications and room availability, will flow through the message broker to relevant components.

Design Diagram:



Final Implementation:

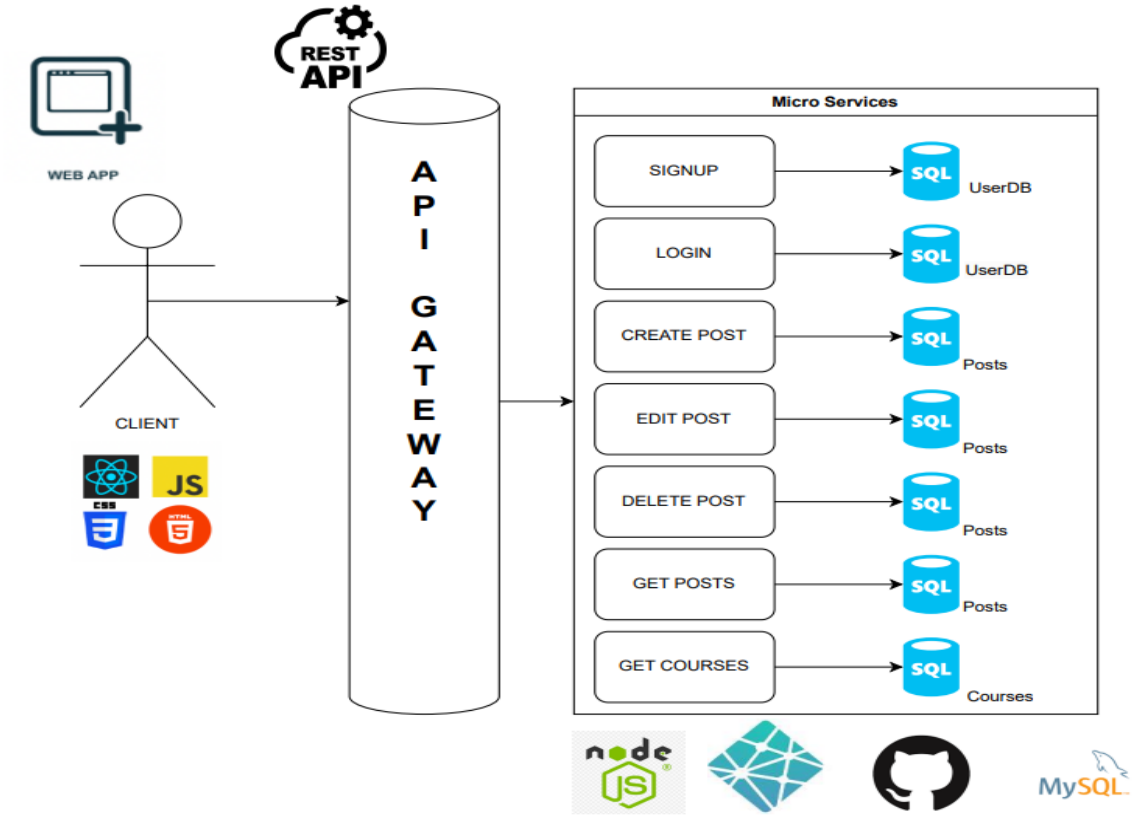
Overall System Architecture:
Our StudySync will adopt a microservices-based architecture, providing modularity, scalability, and maintainability. The front-end will be developed using JavaScript and React.js. At the back-end, we will use Node.js with Express.js and MySQL for structured data storage.

User Interaction:
Users will access StudySync through a web interface. The platform will offer features such as an inbuilt discusssion board, study group finder via posts, mental health tips, course finder/feedback. User authentication and authorization will be implemented using Passport.js.

Software Components Interaction:
The front-end components will communicate with back-end microservices through a RESTful API.

Data Flow:
User inputs and requests will flow from the front-end to the back end for processing and storage in the MySQL database.

Design Diagram:



Database Design:

DataBase: UserDB

Table: User

ID	F_name	L_name	Email	Program	Password	Year

DataBase: Posts

Table: Content

PostID	UserID	Title	Content	CourseID	Program

DataBase: Courses

Table: Biomedical Engineering

ID	Course_Name	Year

Table: Chemical Engineering

ID	Course_Name	Year

Table: Civil Engineering

ID	Course_Name	Year

- Similar tables exist for other engineering major

Why we did not go for an OOP approach with this project?

1. Scope of the Project: The project primarily focuses on frontend presentation and styling with HTML, CSS, and JavaScript for interactivity. Given the relatively small scope of the project and its straightforward requirements, opting for a simpler, more direct approach without heavy reliance on OOP was deemed sufficient.

2. Functional Requirements: The project involves rendering static content, implementing basic user interactions, and ensuring responsive design across different devices. These requirements do not necessitate complex data models or extensive business logic that typically benefit from an object-oriented architecture.

3. Skillset and Familiarity: The development team comprises individuals who are more proficient in procedural or functional programming paradigms. Given the team's familiarity and expertise with these approaches, it was decided to leverage their existing skills rather than introducing the overhead of learning and implementing OOP principles.

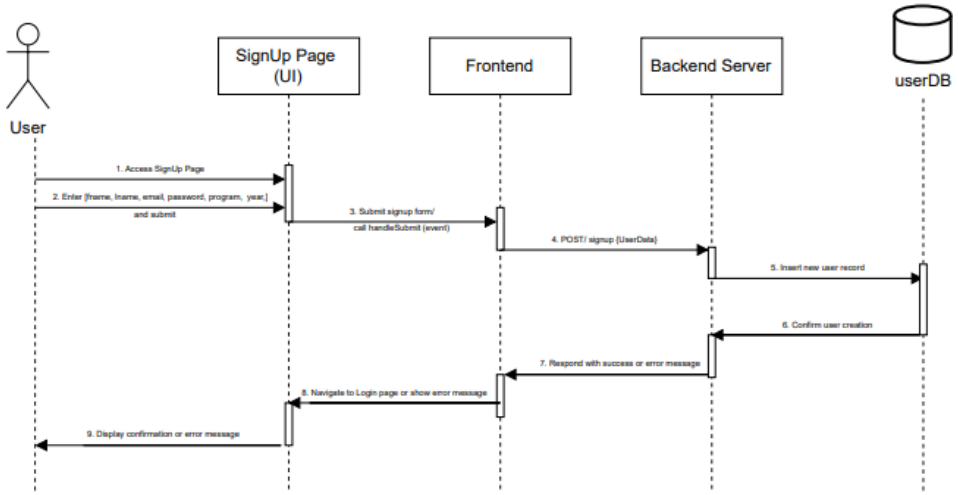
4. Project Constraints: The project is subject to constraints such as tight deadlines and limited resources. In light of these constraints, prioritizing simplicity and efficiency in the development process was essential. Opting for a straightforward approach without extensive OOP design allowed the team to meet project requirements within the given constraints.

5. Performance Considerations: Performance considerations, particularly rendering speed and interactivity, were key factors in the decision-making process. For frontend-focused projects like ours, the immediate priority is to deliver a responsive and visually appealing user experience. Complex object-oriented designs might introduce unnecessary overhead and potentially impact performance negatively.

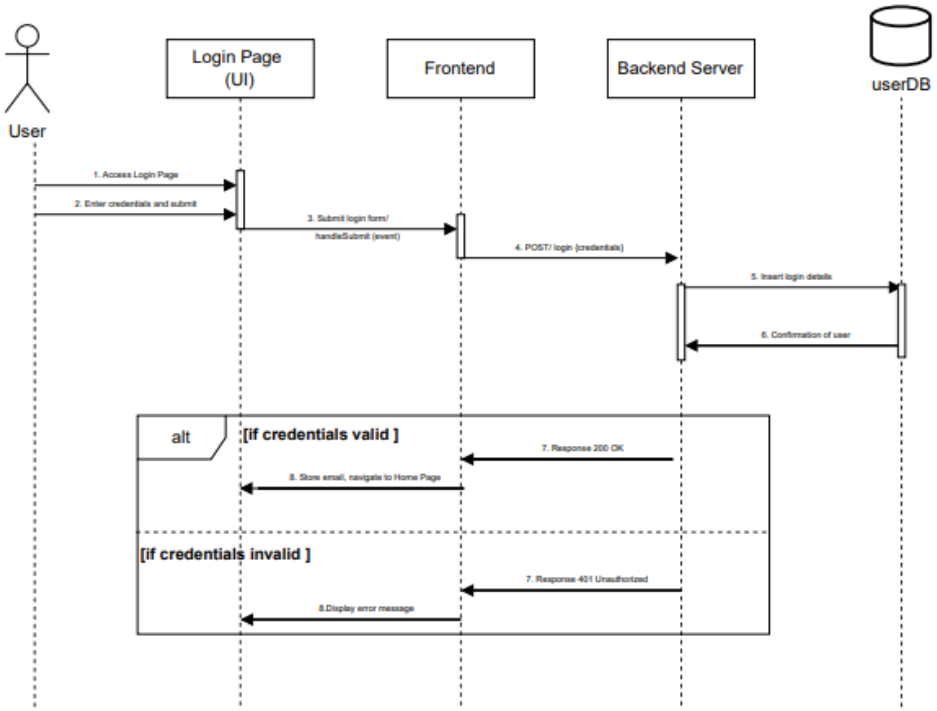
In conclusion, while Object-Oriented Programming offers numerous benefits in terms of code organization, modularity, and extensibility, it was not deemed necessary or appropriate for the current project. By prioritizing simplicity, efficiency, and the utilization of existing skillsets, the team was able to deliver a satisfactory solution that meets the project's objectives effectively.

Sequence Diagrams:

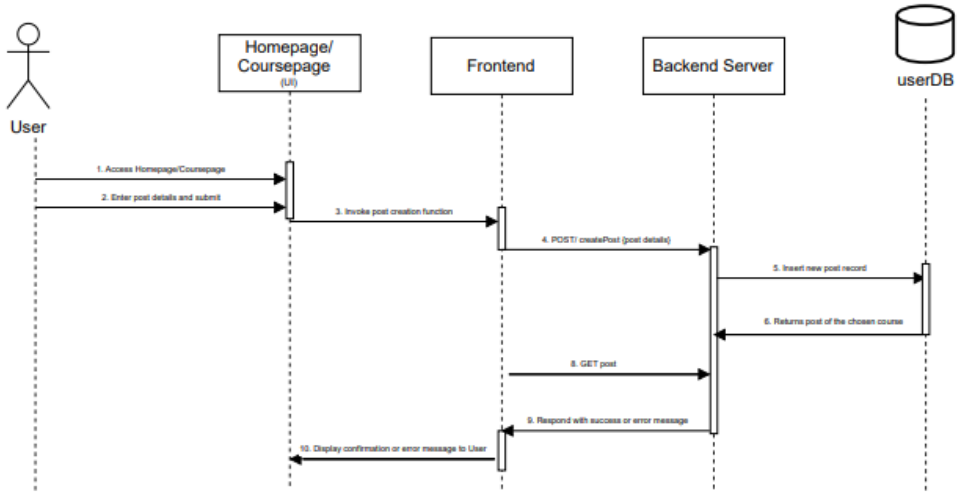
Scenario 1: When a user signs up



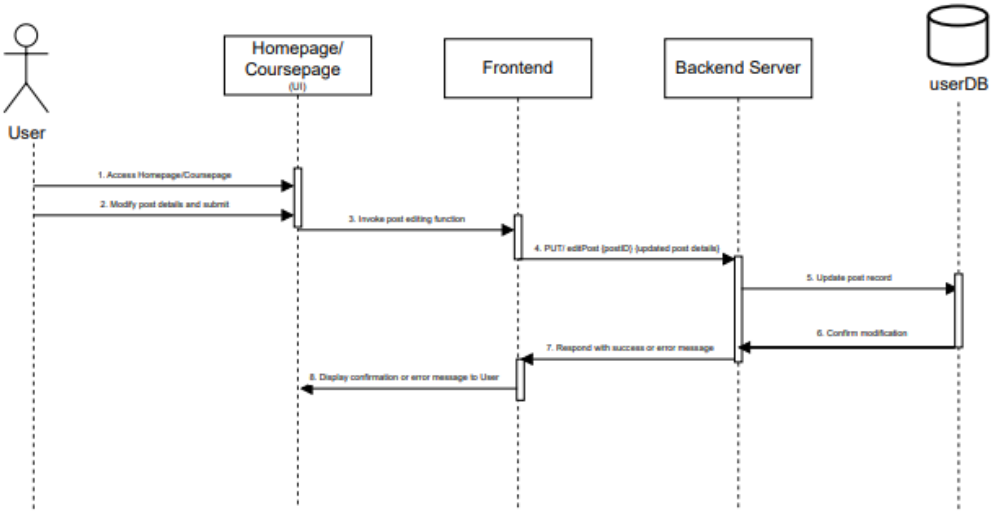
Scenario 2: When a user logs in



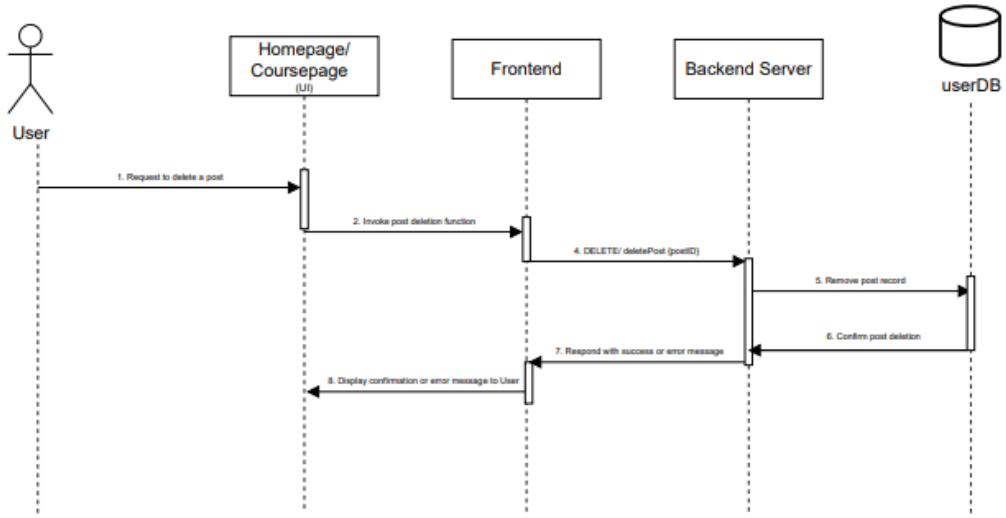
Scenario 3: When a user makes a post



Scenario 4: When a user edits a post



Scenario 5: When a user deletes a post



Implementation Plan:

Technologies We Intend to Use:
- Front-end: JavaScript, React.js, HTML/CSS
- Back-end: Node.js, Express.js, MySQL, AXIOS, CORS

How We Will Manage Source Code:
We will utilize Git and a version control system for source code management. Our repository on a platform like GitHub or GitLab will facilitate collaboration and version tracking.

How We Will Integrate Components:
We plan to implement containerization and orchestration for efficient component integration and scaling. Clear communication between microservices through the RESTful API will ensure seamless functionality.

How We Intend to Test the System:
Our testing approach includes the development and execution of unit tests using Selenium for the Frontend and CURL for the backend.

How We Will Deploy the System:
The system was to be deployed to production using container orchestration tools such as Docker to ensure scalability and ease of management. A robust deployment process with rollback mechanisms will be in place to handle potential issues. However, due to time constraints – the final version of the application was deployed on the local machine.

Testing Results:

(Actual Files for the Test Reports is provided on the GitHub – In the Documents Folder)

Backend:

Test Case ID Backend	Function Being Tested	Test Case	Input	Expected outcome	Actual Outcome	Result
#TC1_B	Login	Valid Credentials	curl -X POST http://localhost:3003/login -d '{"email": "arvikrishna03@gmail.com", "password": "1234"}' -H "Content-Type: application/json"	logged in Successfully	logged in Successfully	Pass
#TC2_B	Login	Invalid Credentials	curl -X POST http://localhost:3003/login -d '{"email": "arvikrishna03@gmail.com", "password": "1234tdgd"}' -H "Content-Type: application/json"	failed to login	failed to login	Pass
#TC3_B	Login	non existent use	curl -X POST http://localhost:3003/login -d '{"email": "invalid@gmail.com", "password": "tdgd"}' -H "Content-Type: application/json"	failed to login	failed to login	Pass
#TC4_B	Sign up	existing user	curl -X POST http://localhost:3003/signup -d '{"fname": "Jane", "lname": "Smith", "email": "new@email.com", "program": "Software Engineering", "password": "abcdef", "year": "2"}' -H "Content-Type: application/json"	user already exits	user already exits	Pass
#TC5_B	Sign up	new user	curl -X POST http://localhost:3003/signup -d '{"fname": "Jane", "lname": "Smith", "email": "new@email.com", "program": "Software Engineering", "password": "abcdef", "year": "2"}' -H "Content-Type: application/json"	creates a new user	creates a new user	Pass
#TC6_B	Sign up	not all columns	arvindkri03/signup -d '{"fname": "Alice", "lname": "Johnson", "email": "alice@email.com", "program": "Engineering"}' -H "Content-Type: application/json"	invalid entry	invalid entry	Pass
#TC7_B	Get user courses	Valid user email	curl -X POST http://localhost:3003/user-courses -d '{"email": "arvikrishna03@gmail.com"}' -H "Content-Type: application/json"	invalid data	invalid data	Pass
#TC8_B	Get user courses	Invalid user email	curl -X POST http://localhost:3003/user-courses -d '{"email": "invalid@gmail.com"}' -H "Content-Type: application/json"	invalid data	invalid data	Pass
#TC9_B	Get posts	Valid Course ID	curl -X GET http://localhost:3003/posts/444	Gives you the Post	Gives you the Post	Pass
#TC10_B	Get posts	Inavlid Course ID	curl -X GET http://localhost:3003/posts/abc	invalid data	invalid data	Pass
#TC11_B	Get posts	Valid Course ID with no posts	curl -X GET http://localhost:3003/posts/401	No post found under this couse ID	No post found under this couse ID	Pass
#TC12_B	Delete posts	Valid Post and User ID	curl -X DELETE http://localhost:3003/delete-post/2/2	Deleted Post	Deleted Post	Pass
#TC13_B	Delete posts	Invalid Post ID	curl -X DELETE http://localhost:3003/delete-post/2/3	invalid data	invalid Data	Pass
#TC14_B	Delete posts	Invalid User ID	curl -X DELETE http://localhost:3003/delete-post/3/2	invalid data	Invalid DAta	Pass
#TC15_B	Delete posts	Missing one of them	curl -X DELETE http://localhost:3003/delete-post/2/	Data missing	Data missing	Pass
#TC16_B	Edit posts	All Valid Inputs				Pass
#TC17_B	Edit posts	Invalid post ID or user ID				Pass
#TC18_B	Edit posts	Title/Content too long				Pass
#TC19_B	Edit posts	Empty fields				Pass
#TC20_B	Create Post	Valid Data	curl -X POST http://localhost:3003/create-post -H "Content-Type: application/json" -d '{"userId": "123", "title": "New Post1", "content": "This is a new post", "courseId": "444", "program": "Test Program1"}'	Post is Created	Post is Created	Pass
#TC21_B	Create Post	Missing Data	curl -X POST http://localhost:3000/create-post -H "Content-Type: application/json" -d '{"userId": "123", "title": "New Post", "content": "This is a new post", "program": "Test Program"}'	Data is missing	Data is missing	Pass
#TC22_B	Create Post	Invalid Data	curl -X POST http://localhost:3000/create-post -H "Content-Type: application/json" -d '{"userId": "ABC", "title": "New Post", "content": "This is a new post", "courseId": "444", "program": "Test Program"}'	Invalid data	Invalid Data	Pass
#TC23_B	Get Post Id	Valid Data	curl -X POST http://localhost:3003/postId -H "Content-Type: application/json" -d '{"userId": "456", "title": "New Post2", "content": "This is sec post", "courseId": "404"}'	Gives post id eg: 2	gives post id eg: 2	Pass
#TC24_B	Get Post Id	Invalid Data	curl -X POST http://localhost:3003/postId -H "Content-Type: application/json" -d '{"userId": "456", "title": "New Post2", "content": "This is third post", "courseId": "404"}'	Post not found	Post not found	Pass
#TC25_B	Get Post Id	Invalid Course ID	curl -X POST http://localhost:3003/postId -H "Content-Type: application/json" -d '{"userId": "456", "title": "New Post2", "content": "This is sec post", "courseId": "403"}'	Post not found	Post not found	Pass
#TC26_B	Get Post By Id	Valid ID	curl -X POST http://localhost:3003/postId -H "Content-Type: application/json" -d "{\"userId\":456,\"title\":\"New Post2\",\"content\":\"This is sec post\",\"courseId\":404}"	Gives Post	Gives Post	Pass
#TC27_B	Get Post By Id	non existent ID	curl -X POST http://localhost:3003/postId -H "Content-Type: application/json" -d "{\"userId\":143,\"title\":\"New Post2\",\"content\":\"This is sec post\",\"courseId\":404}"	Post not found	Post not found	Pass
#TC28_B	Get Post By Id	missing ID	curl -X POST http://localhost:3003/postId -H "Content-Type: application/json" -d "{\"userId\":143,\"content\":\"This is sec post\",\"courseId\":404}"	Invalid data	Invalid Data	Pass
#TC29_B	Get Post By Id	Empty ID	curl -X POST http://localhost:3003/postId -H "Content-Type: application/json" -d "{\"userId\":,\"content\":\"This is sec post\",\"courseId\":404}"	Invalid data	Invalid Data	Pass
#TC30_B	Check authorization	User Authorized	curl -X GET http://localhost:3003/check-authorization/1/1	Authorized User	Authorized	Pass
#TC31_B	Check authorization	User Not Authorized	curl -X GET http://localhost:3003/check-authorization/2/1	Unauthorized	Unauthorized	Pass
#TC32_B	Check authorization	User is Missing	curl -X GET http://localhost:3003/check-authorization//1	Invalid data	Invalid Data	Pass
#TC33_B	Check authorization	Post ID is invalid	curl -X GET http://localhost:3003/check-authorization/1/xyz	Invalid data	Invalid Data	Pass

Frontend:

Test Case #	Use Case	Function Being Tested	Initial Systems State	Input	Expected Output	Actual Output	Result
#TC1_F	User signs up	Sign Up	Application launched	Username : "test@test.com" Password; "test"	Successful signup and redirect to login page	Successful signup and redirect to login page	Pass
#TC2_F	User logs in	Login	Sign-up completed	Username : "test@test.com" Password; "test"	Redirects to home page	Redirects to homepage	Pass
#TC3_F	User adds a new post to a s	Add New Post	User logged in , sidebar course clicked	Post title: "test" , Post body : "test"	Post successfully added to the selected course	Post successfully added to the selected course	Pass
#TC4_F	User deletes a post	Delete Post	Post displayed in user interface	Click delete button	Confirmation prompt for deletion or post removal	Confirmation prompt for deletion or post removal	Pass
#TC5_F	User edits a post	Edit Post	Post displayed in user interface	Click edit button , make changes,Click submit	Post updated with the edited content	Post updated with the edited content	Pass
#TC6_F	word limit on post title	Title Word Limit	Post creation form displayed	Title: " iiiiiiiiiiiiiiiiiiiiiii....."	Post does not get submitted due to input being beyond word limit.	Post does not get submitted due to input being beyond word limit	Pass
#TC7_F	word limit on post body	Body Word Limit	Post creation form displayed	Body: " Hiiiiiiiiiiiiiiiiiiiiiiiiiiiii....."	Post does not get submitted due to input being beyond word limit.	Post does not get submitted due to input being beyond word limit	Pass
#TC8_F	User inputs non-existent en	Login Email text box	login page displayed	email: "testing@test.com"	Does not redirect to homepage as email doesn't exist in Database	Does not redirect to homepage as email doesn't exist in Database	Pass
#TC9_F	User inputs wrong email	Login Email text box	Login page displayed	email:"testing"	Error: please put an @ in user email, email format incorrect	Error: please put an @ in user email, email format incorrect	Pass
#TC10_F	Leaves signup page incompl	Sign up	Sign up page displayed	Email left blank	Error: sign up incomplete	Error: sign up incomplete	Pass
#TC11_F	Dark mode functionality	Dark mode button	Home page in light mode	User clicks on dark mode button	Page turns to dark mode	Page turns to dark mode	Pass
#TC12_F	Posting an empty post	Add New Post	Home page	User attempts to post an empty post	Post does not get published as its empty	Post does not get published as its empty	Pass
#TC13_F	Swapping between differen	Course Buttons	Home page	User attempts to swap between different courses in home page	Successfully swaps between different courses	Successfully swaps between different courses	Pass

RTM:

	A	B	C	D	E
1	**Requirement ID**	**Functional/Non-Functional**	**Test Case ID(s)**	**Functionality Tested**	**Requirement**
2	#RID_1	Functional	#TC1_F, #TC1_B	User signs up	Users must be able to sign up successfully.
3	#RID_2	Functional	#TC2_F, #TC2_B	User logs in	Users must be able to log in with valid credentials.
4	#RID_3	Functional	#TC3_F, #TC20_B	Create Post	Users must be able to add a new post to a specific course.
5	#RID_4	Functional	#TC4_F, #TC12_B	Delete Post	Users must be able to delete their posts.
6	#RID_5	Functional	#TC5_F, #TC16_B	Edit Post	Users must be able to edit their posts.
7	#RID_6	Functional	#TC6_F, #TC22_B	Word limit on post title	Posts must adhere to the specified word limit for the title.
8	#RID_7	Functional	#TC7_F, #TC22_B	Word limit on post body	Posts must adhere to the specified word limit for the body.
9	#RID_16	Functional	#TC23_B	Create Post	Users must not be able to create a post if required data is missing.
10	#RID_17	Functional	#TC23_B, #TC24_B, #TC25_B	Get Post ID	Users must receive valid post ID for valid data and notifications if post is not found or if course ID is invalid.
11	#RID_18	Functional	#TC26_B, #TC27_B, #TC28_B, #TC29_B	Get Post By ID	Users must receive valid post data for a valid post ID and notifications if post is not found or if ID is missing or empty.
12	#RID_19	Functional	#TC30_B, #TC31_B, #TC32_B, #TC33_B	Check authorization	Users must be authorized to perform certain actions and receive notifications for unauthorized attempts or missing/invalid data.
13	#RID_8	Non-Functional	#TC8_F, #TC3_B	User inputs non-existent email	Users must not be able to log in with a non-existent email.
14	#RID_9	Non-Functional	#TC9_F, #TC3_B	User inputs wrong email	Users must provide a valid email format for login.
15	#RID_10	Non-Functional	#TC10_F, #TC6_B	Leaves signup page incomplete	Users must fill in all required fields for sign up.
16	#RID_11	Non-Functional	#TC11_F	Dark mode functionality	Users must be able to switch between dark and light modes.
17	#RID_12	Non-Functional	#TC13_F	Swapping between different courses	Users must be able to switch between different courses.
18	#RID_14	Non-Functional	#TC9_B, #TC10_B, #TC11_B	Get posts	Users must receive valid posts for a valid course ID and a notification if no posts are found.
19	#RID_15	Non-Functional	#TC12_B, #TC13_B, #TC14_B, #TC15_B	Delete posts	Users must be able to delete their posts and receive notifications for invalid data or missing IDs.

Project Planning:

Team Organization:

We have divided into two teams: Front-end and Back-end Development

Front-end Development	Backend Development
Adhil Ashraf Al Farhana Siddique Mehvish Shakeel	Anfaal Mahbub Arvind Krishnaa Varshiny Gogulanathan

We split evenly between front-end and back-end tasks. Front-end developers focus on client-side functionality, while back-end developers handle server-side logic and database management. We held weekly meetings to discuss project progress, address any challenges, and coordinate upcoming tasks. Additionally, we use an organized Discord server with channels for tasks, deadlines, and specific project areas like front-end and back-end. We also used a WhatsApp groupchat for quick reminders and conversations.

Schedule:

Week 1-2 (Feb 6 - Feb 19): Detailed Design and Setup

Front-end Team:

- Design UI mockups and wireframes (Feb 6 - Feb 10)
- Task 2: Set up React.js environment (Feb 10 - Feb 12)

Back-end Team:

- Design system architecture and database schema (Feb 6 - Feb 10)
- Set up Node.js, Express.js, and MySQL environment (Feb 10 - Feb 12)

Both Teams:

- Set up version control system (e.g., Git/GitHub) (Feb 14 - Feb 19)

Week 3-4 (Feb 20 - Mar 4): Front-end and Back-end Development

Front-end Team:

- Develop UI components for exam date scheduler and study group finder (Feb 20 - Feb 26)
- Integrate authentication using Passport.js (Feb 27 - Mar 2)

Back-end Team:

- Implement RESTful API endpoints for backend to frontend integration/ information transfer (Feb 20 – Feb 26)
- Set up MySQL database and implement data storage logic (Feb 27 - Mar 2)

Week 5-6 (Mar 5 - Mar 18): Feature Implementation and Testing

Front-end Team:

- Develop UI components for mental health tips and course finder (Mar 5 - Mar 11)
- Conduct UI/UX testing and refinement (Mar 12 - Mar 14)

Back-end Team:

- Implement RESTful API endpoints for mental health tips and course finder/feedback (Mar 5 - Mar 11)

Both Teams:

- Integrate all features for comprehensive testing (Mar 15 - Mar 18)

Week 7-8 (Mar 19 – Mar 27): Testing and Deployment Preparation

- Develop and execute unit tests, integration tests, and end-to-end tests (Mar 19 - Mar 25)
- Document deployment process and rollback mechanisms (Mar 25 – Mar 26)
- Conduct final system testing and bug fixing (Mar 26 -27)

Milestones/Accomplishments:

- System Design and Architecture Planning: Create the system's architecture and design.
- Front-end Development : Develop front-end components for exam scheduling and study group finding.
- Back-end Development (User authentication, Database setup): Implement user authentication and set up the database.
- Front-end Development (Mental health tips, Course finder/feedback): Develop front-end components for mental health tips and course-related features.
- Back-end Development (RESTful API): Implement the back-end components including RESTful API.
- Testing and Quality Assurance (Unit, integration) Conduct comprehensive testing to ensure system functionality and quality.
- Deployment and Scaling - Deploy the system on local machine.

Appendix:

- Initial Rough Sketch of Databases

DataBase: UserDB

Table: Login

userID	Email	Password

Table: Signup

userID	F_name	L_name	Email	Program

DataBase: Courses

Table: Fall 2024

userID	Course_1	Course_2	Course_3	Course_4	Course_5

- Similar tables will exist for other courses

DataBase: Posts

Table: Content

userID	Course	Content