SPROJ Report

Owais Ahsan 24100290
Nauman Ijaz 24100311
Khizar Nawab 24100225
Muhammad Abdullah Sarfraz 24100029
Muneeb Akmal 24100218

Advisor: Dr. Waqar Ahmad
School of Science and Engineering
Lahore University of Management Sciences
28th April 2024

Acknowledgement and Dedication

We would like to express my sincere gratitude to the following individuals and groups:

The LUMS Student Community and Alumni: Your vibrant spirit and shared experiences have been a constant source of inspiration throughout this project.

Professor Waqar: Your invaluable guidance, insights, and unwavering support have been instrumental in shaping this work.

Dedication

This project is dedicated to the future of LUMS. May it serve as a tool to enhance the experiences of students for generations to come.

Certificate

I certify that the senior project titled "**Lumin'**" was completed under my supervision by the following students:

_____

_____

_____

and the project deliverables meet the requirements of the program.

---------- Date:

Advisor (Signature)

---------- Date:

Co-advisor (if any)

Table of Contents

List of Figures

The figures used in this document include:

1. Introduction

a. Introduction

Lumin' is a comprehensive and innovative platform designed exclusively for LUMS students and alumni. With a wide range of powerful features, this app offers a seamless experience to navigate all aspects of campus life. The Social Platform enables users to connect with the entire LUMS community, fostering a strong sense of belonging and networking opportunities. Graduation plans are made stress-free through the Course Scheduler, ensuring students stay on track with their academic journey. The Information Cluster provides easy access to vital campus details and services, keeping users informed and organized. Real-time event information is readily available through the Events Manager, ensuring users never miss out on campus activities. The application also promotes student welfare causes, allowing users to support them with ease. Centralized instructor ratings and reviews aid in making informed decisions about courses and instructors. Academic success is further facilitated with the GPA Predictor, helping students set and achieve their academic goals. In summary, the application is an indispensable tool for LUMS students and alumni, enhancing their campus experience, fostering community engagement, and supporting academic success.

1. Domain of our application: The application is designed for the Lahore University of Management Sciences (LUMS) community, encompassing a wide range of functionalities aimed at enhancing campus life and academic success for students and alumni .
2. Target users of this application: The primary target users of the application are LUMS students and alumni. The app offers features tailored to improve their academic and campus life experience by providing course scheduling, event management, community interaction, and more .
3. The product is a comprehensive **mobile application** that integrates various tools and services to support the academic and social life of students and alumni at LUMS.

b. Objective and Scope

The objective of the application is to create a comprehensive and innovative platform tailored specifically for LUMS students and alumni, enhancing their campus experience. This mobile application aims to streamline campus life by integrating various functionalities such as a social platform, course scheduler, event manager, GPA predictor, and a comprehensive instructor review database.

The decision to develop this application stems from the need to foster a strong sense of community and improve the academic journey of students and alumni. By providing tools for better schedule management, social engagement, and academic resources, the application significantly impacts and enhances business operations within the educational domain by promoting efficiency and accessibility in academic and extracurricular activities.

This platform not only caters to the immediate needs of managing academic schedules and engagements but also serves as a long-term engagement tool for alumni, keeping them connected with the campus community and current events, thus sustaining a lifelong bond with the institution.

c. Development Methodology

In developing our application, we employed a hybrid development methodology, blending elements of both the Waterfall and Agile (SCRUM) frameworks to leverage the strengths of each while mitigating their weaknesses. This approach was selected to ensure a well-structured yet flexible development process, essential for meeting the dynamic needs of our target users at LUMS.

Waterfall Methodology:

We started with the Waterfall methodology for its sequential design process, which is advantageous for its clarity and order. This approach allowed our team to define all project requirements upfront, ensuring that each phase of the project, from conception to deployment, was well documented and followed a predetermined path. This method was particularly useful in the early stages of development, where clear, static requirements were essential for laying a solid foundation for the application.

Agile (SCRUM) Methodology:

To complement the structured planning of Waterfall, we integrated Agile (SCRUM) practices as the project progressed. This methodology supported a more iterative and incremental development approach, allowing our team to remain adaptable to changing requirements and user feedback. Regular sprint meetings and scrum sessions facilitated a responsive development environment where new features and adjustments could be swiftly implemented based on real-time insights from our ongoing testing and user interactions.

The combination of these methodologies enabled our development team to maintain rigorous standards of documentation and planning, while also embracing the flexibility needed to innovate and adjust to the evolving requirements of the LUMS community. This hybrid approach ensured the delivery of a robust and user-centered application, aligning with both the technical goals and the community-oriented objectives of the project.

d. Contributions

Lumin revolutionizes the student experience with several key features. Our innovative instructor review portal empowers informed decision-making by combining real-time feedback with historical LUMS performance data. The intuitive course scheduler eliminates planning stress, allowing students to visualize their academic path, ensure timely graduation, and effortlessly adjust based on course availability. We foster a thriving LUMS network with a dedicated social media platform, encouraging connections, knowledge sharing, and a stronger sense of belonging. Lumin's real-time events management system keeps students in the loop, maximizing engagement with campus life. Additionally, Lumin provides a central information cluster, consolidating all essential student resources in one easily accessible location. Finally, recognizing student wellbeing as paramount, Lumin offers a secure student welfare case module to confidentially report concerns, ensuring timely support from the university.

While apps like U.n.i offer social networking features for university communities, Lumin stands alone as a comprehensive solution designed exclusively for LUMS students. No other app combines social connection, academic planning tools, centralized LUMS information, real-time

event updates, and a dedicated student welfare module in one seamless experience. Lumin addresses the multifaceted needs of LUMS students, streamlining their campus life and empowering them to thrive both inside and outside the classroom.

2. System Requirements

This part will discuss the different actors, their features, and their limitations. These will be categorized into functional and non-functional requirements with their respective objectives.

a. System Actors

Actor Name	Description
Students	Primary users and beneficiaries of the app.
Alumni	Primary users and beneficiaries of the app.
Society	Access to share and manage society-related information, promoting community engagement.
Student Council	Access to share and manage Student Council related information.
Admin	Moderators and updaters responsible for maintaining and adding new information to the app.

b. Functional Requirements

Actor Name	Description
Students	Predict GPA Login Sign Up Add LDF post View event on the map View events list Add transcript Edit profile details Like/Dislike posts Add instructor review Add another account Reply comment Bookmark post Change settings Get notifications

	View Profile
Student Council / Society/ Admin	Add LDF post Add donation Add event Edit donations

c. Non-functional Requirements

List down non-functional requirements.

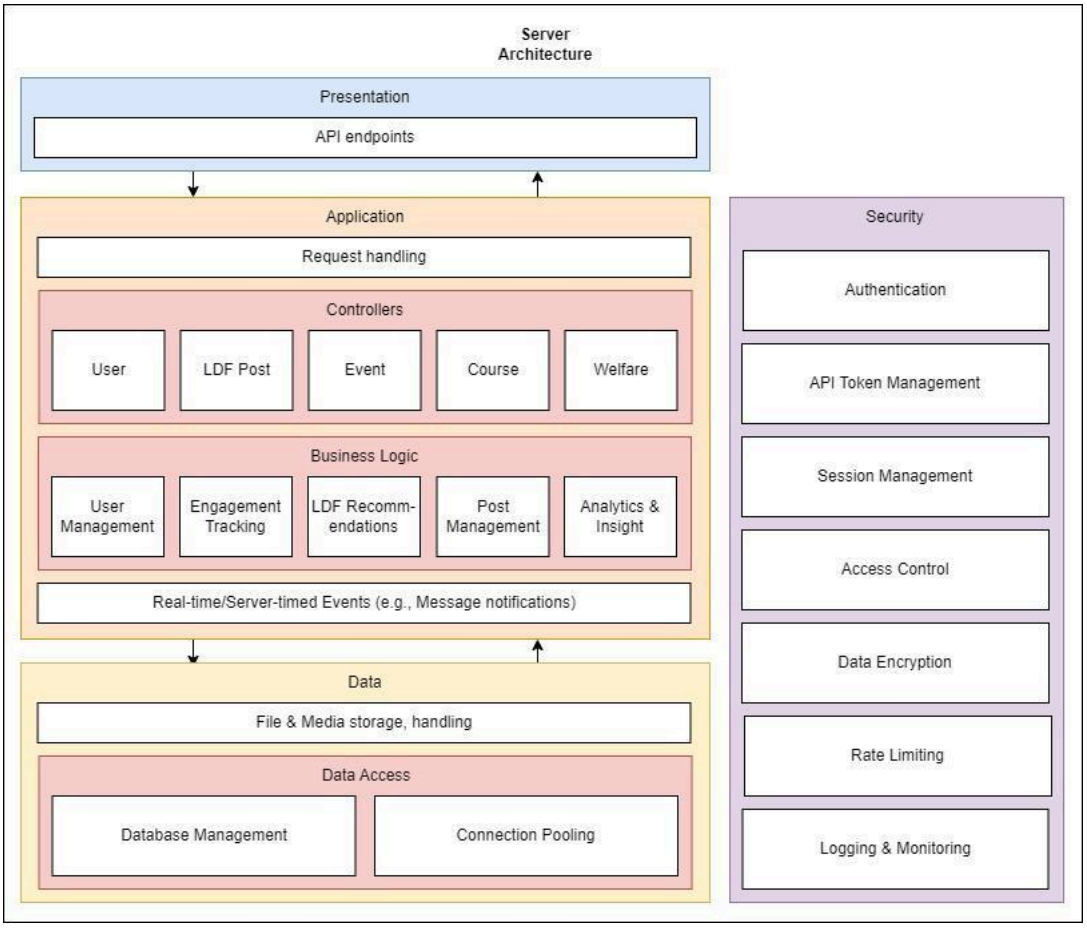
Sr #	Requirements
1	The app should not utilize more than 50 MBs of memory at any time during its execution.
2	Each page of the social media feed in the app should not have a loading time of higher than 5 seconds
3	The app should possess the capability to accommodate evolving requirements through updates.
4	The user should still be able to use offline features even in the unavailability of an internet connection.
5	The server should have an availability of higher than 98%.
6	The app must ensure that campus information, including course details, is consistently updated, with data refreshed no later than every 7 days to maintain accuracy.
7	All sensitive user data must be stored and transmitted using end-to-end encryption to ensure maximum privacy.
8	The application must be capable of accommodating a minimum of 500 users at its initial deployment and should be designed to scale to support 10,000 users and beyond as the user base grows.
9	The application must implement a multi-factor authentication system that includes password hashing, secure token generation, and account lockout mechanisms to ensure robust user authentication.

3. System Architecture

The frontend application itself follows a standard Model-View-Controller architecture, while the server architecture consists of four main layers:

1. Presentation Layer: Exposes API endpoints.
2. Application Layer: Handles request processing, including routers and controllers that execute business logic.
3. Data Layer: Concerned with file and media storage, as well as database management
4. Cross-Cut: Ensures security throughout the application via access control, token management, authentication, and encryption. Logging is also done here.

a. Architecture Diagram



b. Architecture Description

The **frontend application** itself follows a standard Model-View-Controller architecture, where:

1. The model is responsible for managing the data of the application. It receives user input from the controller. This part is mostly managed by our server on the backend, but some concerns are within the local application as well.
2. The view represents the UI/UX of our application, the part the user sees.
3. The controller responds to the user input and performs interactions on the data model.

While the **server architecture** consists of four main layers:

1. Presentation Layer: Exposes API endpoints.
2. Application Layer: Handles request processing, including routers and controllers that execute business logic.
3. Data Layer: Concerned with file and media storage, as well as database management
4. Cross-Cut: Ensures security throughout the application via access control, token management, authentication, and encryption. Logging is also done here.

c. Justification of the Architecture

Pros:

1. Separation of Concerns: The architecture provides a clear separation between different layers. This allows for updates and improvements in specific areas without affecting the whole system.
2. Security: The implementation of access control, token management, and authentication helps meet the non-functional requirement of robust user authentication, including multi-factor authentication (OTP) and password hashing.
3. Logging: Provides insights into system performance and can help identify and troubleshoot issues, improving overall server reliability.
4. Scalability: The architecture is designed to support initial deployment for 500 users and can scale as the user base grows, supporting more than 10,000 users.

Cons:

1. Complexity: The architecture is complex due to multiple layers, requiring more effort in development and maintenance.
2. Security Overhead: Implementing security measures such as encryption and authentication adds some processing overhead, impacting system performance and response times in particular.

Justification and Non-Functional Requirements:

1. Memory Usage: The separation of concerns allows for optimized memory usage, helping keep memory utilization within the memory limit.
2. Offline Features: The architecture allows for caching and storage management, supporting offline features for users when there is no internet connection.
3. High Availability: The system's design, with multiple layers and redundancy measures, helps achieve an availability of higher than 98%.
4. Data Consistency: The data layer manages database updates and file storage, ensuring that campus information is consistently updated.
5. Scalability: The architecture's design supports scaling to accommodate a growing user base, from 500 to over 10,000 users.

d. Tools and Technologies

App:

1. Expo: ^49.0.21
2. React-native: 0.72.10
3. React: 18.2.0
4. Eas-cli: ^5.9.3

Server:

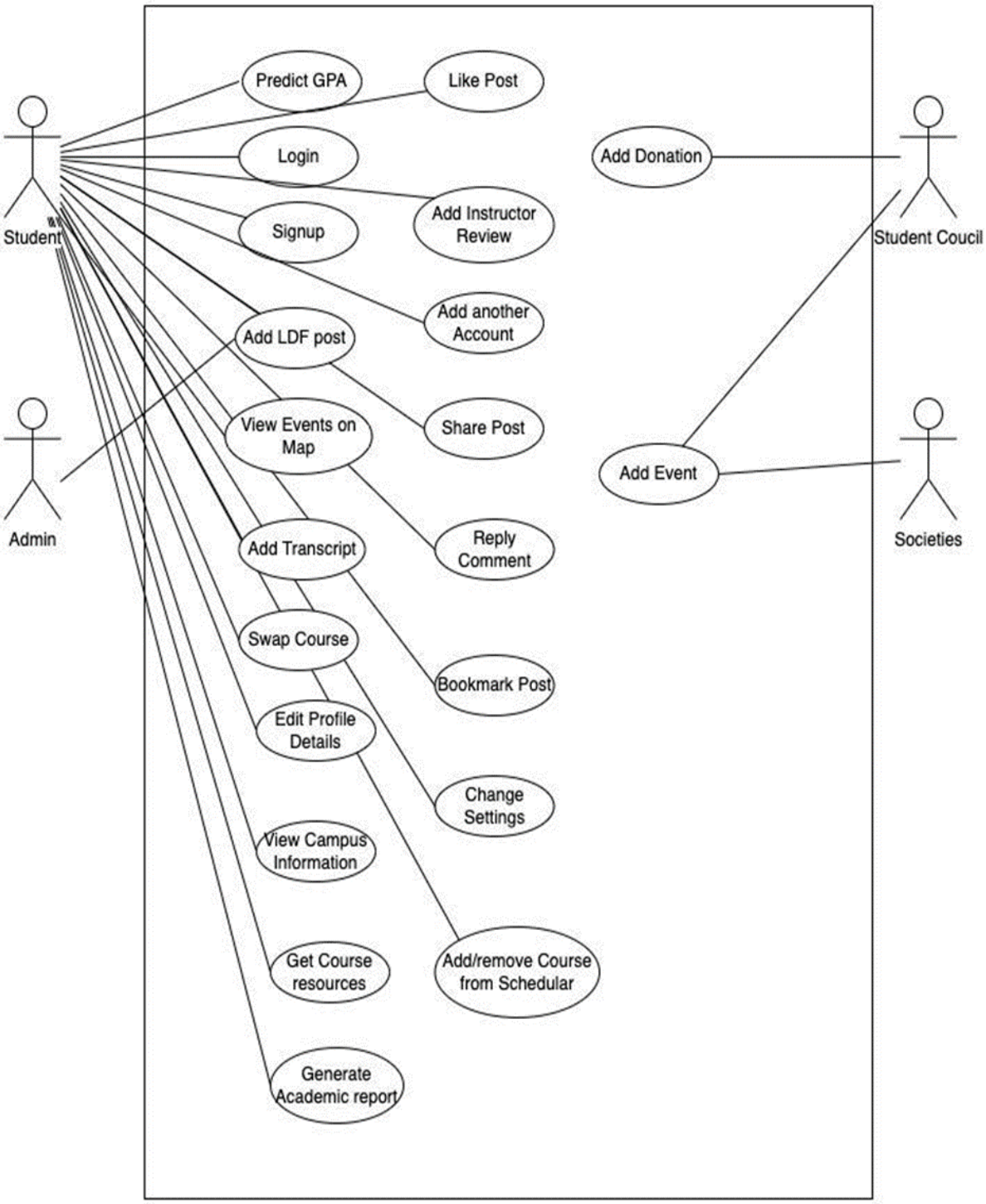
1. Express: ^4.18.2
2. Mongoose: ^8.0.1 (MongoDB Atlas)

3. Firebase: ^10.11.0

Tools to aid development:

1. Github
2. VSCode
3. Figma
4. Trello
5. Miro

4. Requirements Specifications



Use Case Diagram

a. Use Cases

Add Transcript

Identifier	UC-001
Purpose	The student uploads the transcript on the app
Pre-conditions	The student is logged in.
Post-conditions	The transcript is added to the app and is accessible for later use.
Step #	**Typical Course of Action**
1.	The user selects Academics option from homescreen
2.	The user then clicks on the upload transcript button
3.	The user is asked to select the pdf file from the local file explorer
4.	The user clicks the upload button
5.	The user case ends
6.	
Step #	**Alternate Courses of Action**
	N/A
Step #	**Exception Paths**
1.	In step 4, if the upload fails, then an error message is thrown, and the user is asked to retry from step 2.

Add Event

Identifier	UC-002
Purpose	The user can add an upcoming event to notify the app users.
Pre-conditions	The user must be logged in as student-council member or as a society.
Post-conditions	The event is displayed on the events screen.

Step #	**Typical Course of Action**
1.	The user clicks on events tab on homescreen
2.	The user clicks on the add event icon
3.	The user then enters the relevant details on the form
4.	The user clicks on the Done button
5.	The user case ends

Step #	**Alternate Courses of Action**
	N/A

Step #	**Exception Paths**
1.	In step 4, if the event details are not according to specified format, then an error message is displayed with the retry option. (step 3)

Show Event on Map

Identifier	UC-003
Purpose	The user can see the event's location on the map
Pre-conditions	The user is logged in.
Post-conditions	Event on map is visible to the user.

Step #	**Typical Course of Action**
1.	The user clicks on the map button.
2.	The user clicks on the marker over a location.
3.	The events on the location are displayed.
4.	The user clicks on an event.
5.	The event details are displayed
6.	The user case ends

Step #	**Alternate Courses of Action**
	N/A

Step #	**Exception Paths**
	N/A

Add Donation

Identifier	UC-004
Purpose	The user can add a donation case to notify the app users.
Pre-conditions	The user must be logged in as student-council member or as a society.
Post-conditions	The donation case is displayed on the donations screen.

Step #	**Typical Course of Action**
1.	The user clicks on donations tab on homescreen
2.	The user clicks on the add donation icon
3.	The user then enters the relevant details on the form
4.	The user clicks on the Done button
5.	The user case ends

Step #	**Alternate Courses of Action**
	N/A

Step #	**Exception Paths**

1.	In step 4, if the donation details are not according to the specified format, then an error message is displayed with the retry option. (starts from step 3 again)

View Campus Information

Identifier	UC-005
Purpose	The students or alumni can view the information for various facilities of the campus.
Pre-conditions	Users must be logged in. Course Resources are uploaded on the database already.
Post-conditions	Users are able to access the full database of campus information.
Step #	**Typical Course of Action**
1.	The user clicks on the campus information button on the home screen.
2.	The user selects the category of information they want to view.
3.	The user selects the individual information they want to view.
4,	The user is then able to bookmark the specific information.
Step #	**Alternate Courses of Action**
1.	
Step #	**Exception Paths**
1.	In step 1, if the user is not connected to the internet, an error message is displayed.

Build a schedule for courses

Identifier	UC-007
Purpose	The students or alumni can build a weekly courses schedule using a centralized database of courses being offered in the current semester.
Pre-conditions	User must be logged in.
Post-conditions	User is able to build a weekly schedule of courses.
Step #	**Typical Course of Action**
1.	The user clicks on the Scheduler button on the home screen.
2.	The user adds a course by searching for it in the database.
3.	The user finalizes the selection of courses and saves the schedule.
Step #	**Alternate Courses of Action**
1.	When adding courses to the Scheduler, the user can also remove courses by clicking on the selected course and choosing the option to remove the course.
Step #	**Exception Paths**
1.	In step 1, if the user is not connected to the internet, an error message is displayed.

Add another account

Identifier	UC-009
Purpose	User can switch between SC/Society Account and Personal Account.
Pre-conditions	The user has *Logged In* successfully.
Post-conditions	The user can switch between SC/Society Account and Personal Account without having to log out and sign in to another account.

Step #	**Typical Course of Action**
1.	The user goes to his profile page.
2.	The user presses the dropdown menu of accounts.
3.	The user is given the choice to add another account.
4.	The user selects the option and logs in to his society/student council account.
5.	Now the user has the option to switch between accounts.
6.	The use case ends.

Step #	**Alternate Courses of Action**
	N/A

Step #	**Exception Paths**
1.	In Step 3, the user tries to add a 5th Account. System returns an error message: User cannot add more than 4 accounts at one time.
2.	In Step 4, the user gives the wrong credentials. System returns an error message: Incorrect Credentials.

Edit Profile Details

Identifier	UC-010
Purpose	User can edit his/her profile.
Pre-conditions	The user has *Logged In* successfully.
Post-conditions	The user has the profile information updated.
Step #	**Typical Course of Action**
1.	The user goes to his profile page.
2.	The user presses the edit profile button.
3.	The user is taken to the page where he can update his profile info.
4.	The user makes the required changes (such as changing his profile picture or bio etc).
5.	The profile info gets updated.
6.	The use case ends.
Step #	**Alternate Courses of Action**
	N/A
Step #	**Exception Paths**

1.	In Step 4, User makes syntactic error while making changes to the profile info; Error is returned: Syntax Error

Add instructor review

Identifier	UC-011
Purpose	Give a review to an instructor.
Pre-conditions	The user has *Logged in* successfully and The user has *taken the course* previously.
Post-conditions	The user successfully posts a review of the instructor (with a rating out of 5).
Step #	**Typical Course of Action**
1.	The user selects the academic tab from the homepage.
2.	The user selects the instructor tab from the academics page.
3.	The user is taken to the page where he can see a list of all instructors.
4.	The user selects an instructor.
5.	The user is taken to the instructor specific page, where all the info and reviews of the instructor are present.
6.	The user rates the instructor out of five and gives a review.
7.	The use case ends.
Step #	**Alternate Courses of Action**

	N/A
Step #	**Exception Paths**
1.	The user has not taken the course but still tries to add a review: The user is shown a Message: Do you want to upload Updated Transcript? (and upload transcript use case starts then).

Change Settings

Identifier	UC-012
Purpose	Change Settings
Pre-conditions	The user has *Logged in* successfully.
Post-conditions	The user successfully changes app settings according to his needs.
Step #	**Typical Course of Action**
1.	The user goes to his profile page.
2.	The user presses the change settings button.
3.	The user is presented with a list of settings.
4.	The user changes settings according to his/her preferences successfully.
Step #	**Alternate Courses of Action**

	N/A
Step #	**Exception Paths**
	N/A

Post to LDF

Identifier	UC-013
Purpose	Publish a post to LDF.
Pre-conditions	The user has *Logged in* successfully.
Post-conditions	The user successfully publishes a post on the forum.
Step #	**Typical Course of Action**
1.	The user goes to the LDF tab.
2.	The user presses the Create Post button..
3.	The user is presented with an input dialog.
4.	The user inputs and edits the post body, including attaching any media (images, videos, etc).
5.	The user presses the Submit button.
6.	The use case ends.

Step #	Alternate Courses of Action
	Following step 4, the user may choose to discard the post by pressing the Close button.
Step #	**Exception Paths**
	In step 5, if the post contents (i.e text & multimedia) are not in the specified format then an exception is thrown. User is asked to retry and the use-case starts from step 4 again.

Comment on LDF Post

Identifier	UC-014
Purpose	Publish a comment on an existing LDF post.
Pre-conditions	The user has *Logged in* successfully.
Post-conditions	The user successfully publishes a comment under the post.
Step #	**Typical Course of Action**
1.	The user goes to the LDF tab.
2.	The user presses the Comment button under a post
3.	The user is presented with an input dialog.
4.	The user inputs and edits the comment body, including attaching any media (images, videos, etc).

5.	The user presses the Submit button.
6.	The use case ends.
Step #	**Alternate Courses of Action**
	Following step 4, the user may choose to discard the comment by pressing the Close button.
Step #	**Exception Paths**
	If the comment body is empty then the comment is ignored and not posted. Use Case Starts from Step 4.

Bookmark LDF Post

Identifier	UC-015
Purpose	Bookmark your desired LDF Post
Pre-conditions	The user has *Logged in* successfully.
Post-conditions	The user successfully changes and adds the post to bookmarks.
Step #	**Typical Course of Action**
1.	The user goes to the LDF page.
2.	The user located their selected post.

3.	The user clicks on the bookmark icon.
4.	The icon changes from inactive to active configuration.
Step #	**Alternate Courses of Action**
1.	Do the same step 1 and 2
2.	The user clicks on the post to expand it.
3.	The user then clicks on the bookmark icon.
4.	The icon changes from inactive to active configuration.
Step #	**Exception Paths**
1.	In a situation where the post currently viewed by the user is deleted, the application will refresh rather than working as the typical course of action.

Login

Identifier	UC-016
Purpose	User to login to the application
Pre-conditions	The user has downloaded the application and has a working internet connection.
Post-conditions	The user successfully changes logins in.
Step #	**Typical Course of Action**

1.	The user has opens the application.
2.	The user has a working internet connection.
3.	The user enters their login details (email and password).
4.	The user is logged into the application and redirected to the homepage.
Step #	**Alternate Courses of Action**
1.	Do the same step 1 and 2
2.	The user clicks on the post to expand it.
3.	The user logins using their outlook accounts.
4.	The user is logged into the application and redirected to the homepage.
Step #	**Exception Paths**
1.	The user has entered the wrong matching credentials.
2.	The user is denied entry and asked to try again.

Predict GPA

Identifier	UC-019
Purpose	Predict final CGPA
Pre-conditions	The user has *Logged in* successfully. The user has uploaded their transcript or input courses manually.

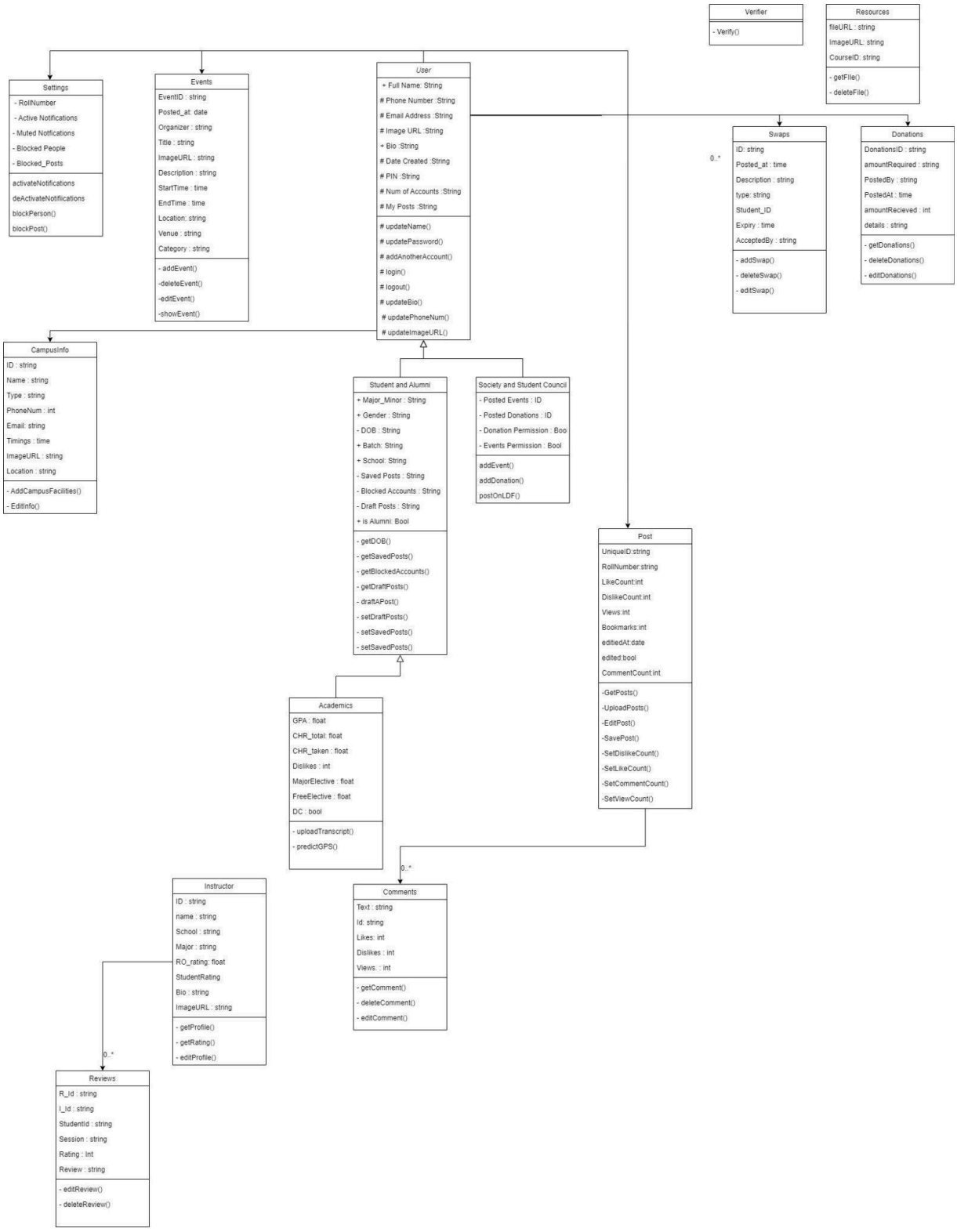
Post-conditions	The user successfully predicts their final CGPA.
Step #	**Typical Course of Action**
1.	The user selects Academics option from homescreen
2.	The user clicks on the GPA predictor button
3.	The user slides the expected GPA and Credit Hours sliders to their desired value.
4.	The user is shown their projected final CGPA based on the input.
5.	The use case ends.
Step #	**Alternate Courses of Action**
1.	N/A
Step #	**Exception Paths**
1.	N/A

Delete Transcript

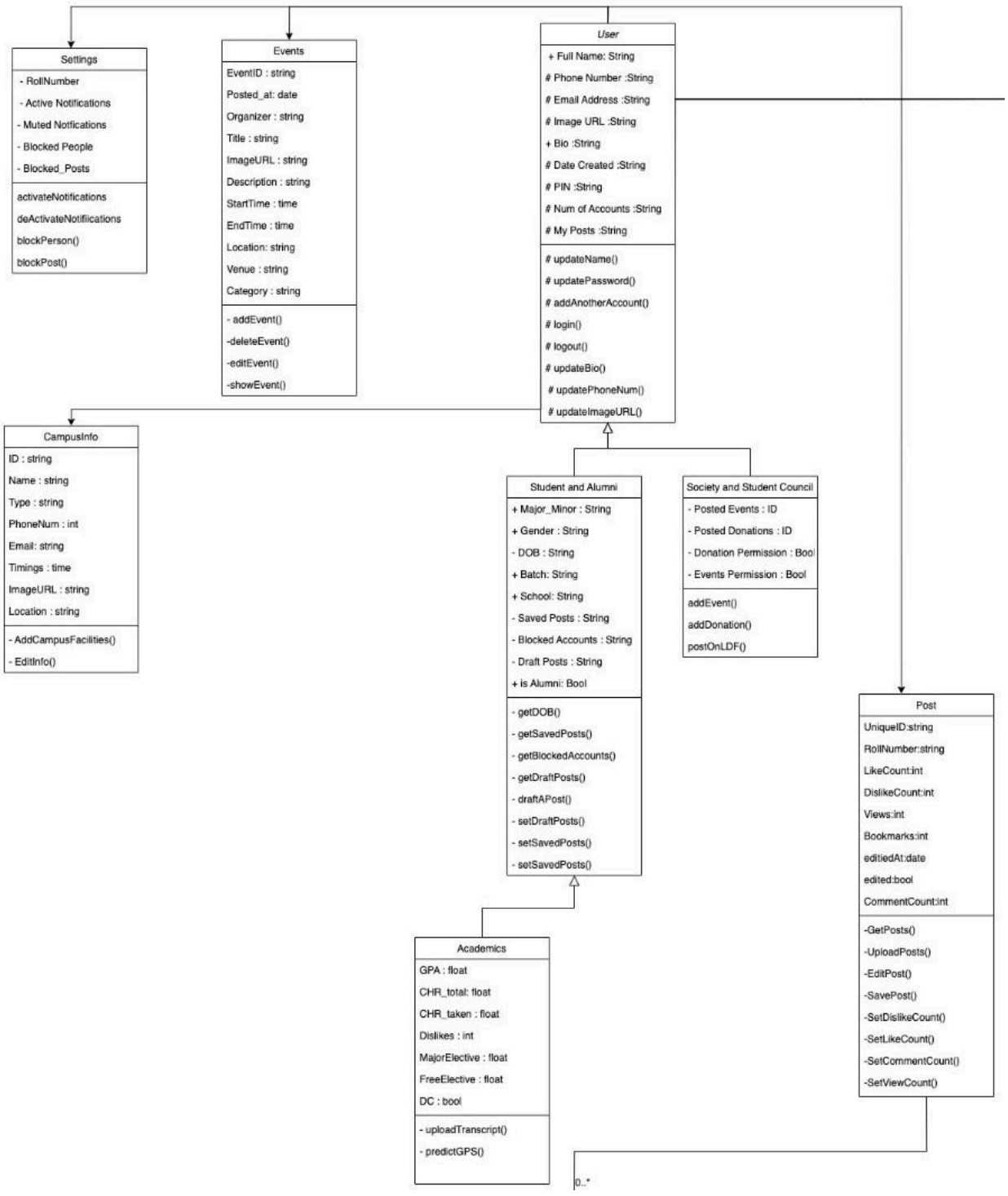
Identifier	UC-020
Purpose	Delete uploaded transcript
Pre-conditions	- The user has *Logged in* successfully. - The user has uploaded their transcript..

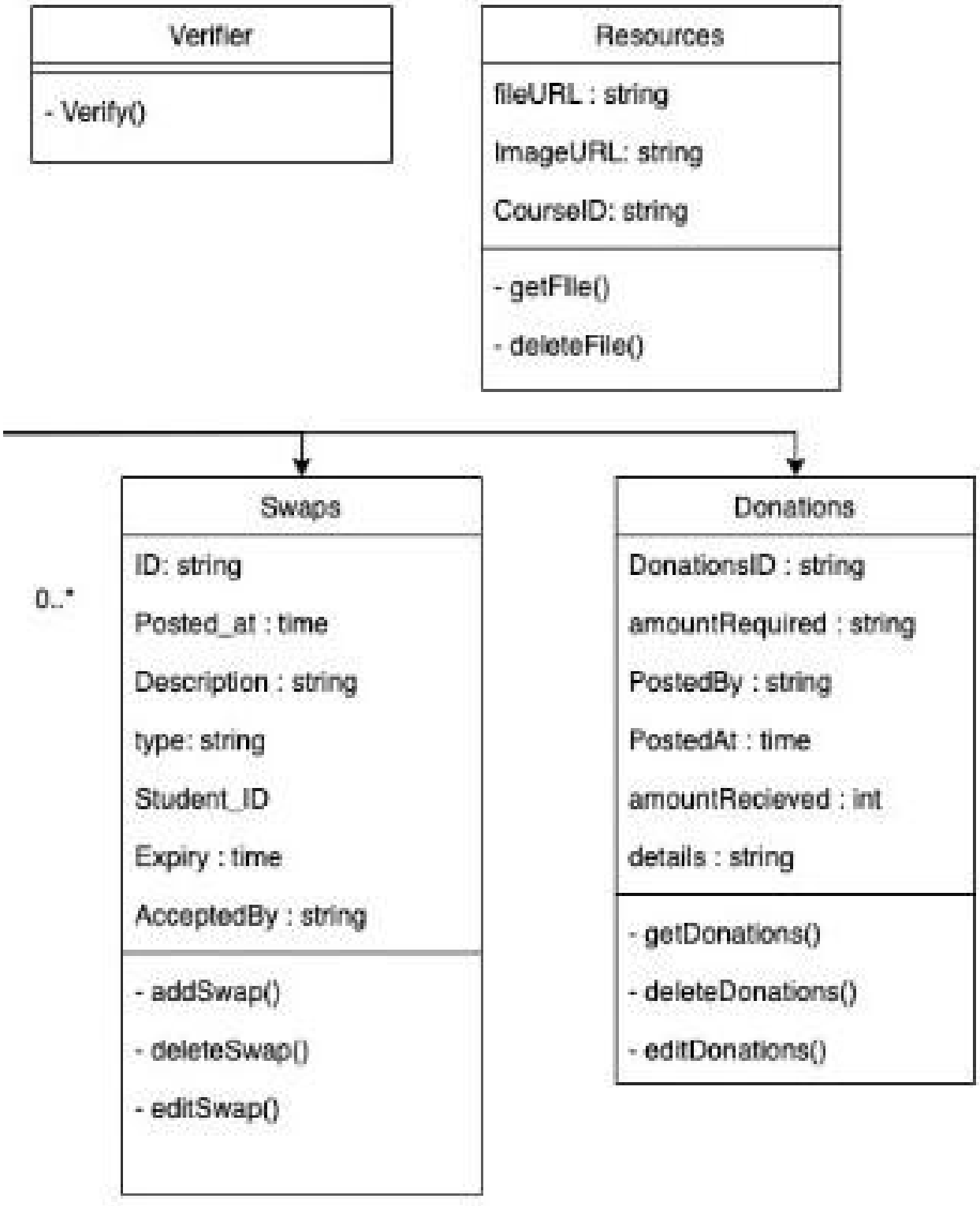
Post-conditions	The user successfully deletes their transcript from the system.
Step #	**Typical Course of Action**
1.	The user selects the Academics option from the homescreen.
2.	The user clicks on the GPA predictor button.
3.	The user then clicks on the Delete Transcript button.
5.	The use case ends.
Step #	**Alternate Courses of Action**
1.	N/A
Step #	**Exception Paths**
1.	N/A

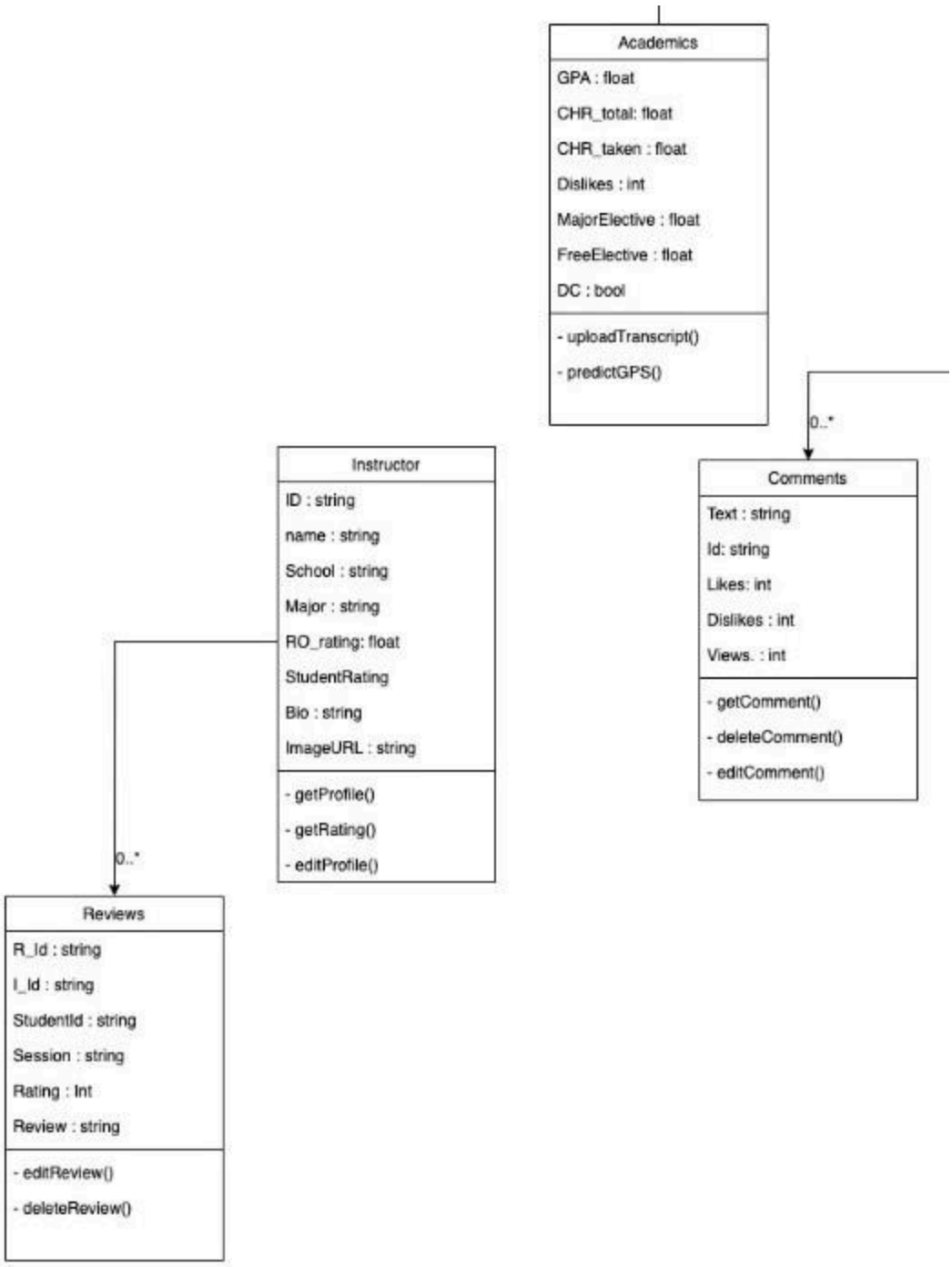
b. Class Diagram



Zooming in


Settings
- RollNumber
- Active Notifications
- Muted Notfications
- Blocked People
- Blocked_Posts
activateNotifications
deActivateNotifications
blockPerson()
blockPost()
Events
EventID : string
Posted_at: date
Organizer : string
Title : string
ImageURL : string
Description : string
StartTime : time
EndTime : time
Location: string
Venue : string
Category : string
- addEvent()
-deleteEvent()
-editEvent()
-showEvent()
User
+ Full Name: String
# Phone Number :String
# Email Address :String
# Image URL :String
+ Bio :String
# Date Created :String
# PIN :String
# Num of Accounts :String
# My Posts :String
# updateName()
# updatePassword()
# addAnotherAccount()
# login()
# logout()
# updateBio()
# updatePhoneNum()
# updateImageURL()
CampusInfo
ID : string
Name : string
Type : string
PhoneNum : int
Email: string
Timings : time
ImageURL : string
Location : string
- AddCampusFacilities()
- EditInfo()
Student and Alumni
+ Major_Minor : String
+ Gender : String
- DOB : String
+ Batch: String
+ School: String
- Saved Posts : String
- Blocked Accounts : String
- Draft Posts : String
+ is Alumni: Bool
- getDOB()
- getSavedPosts()
- getBlockedAccounts()
- getDraftPosts()
- draftAPost()
- setDraftPosts()
- setSavedPosts()
- setSavedPosts()
Society and Student Council
- Posted Events : ID
- Posted Donations : ID
- Donation Permission : Bool
- Events Permission : Bool
addEvent()
addDonation()
postOnLDF()
Academics
GPA : float
CHR_total: float
CHR_taken : float
Dislikes : int
MajorElective : float
FreeElective : float
DC : bool
- uploadTranscript()
- predictGPS()
Post
UniqueID:string
RollNumber:string
LikeCount:int
DislikeCount:int
Views:int
Bookmarks:int
editiedAt:date
edited:bool
CommentCount:int
-GetPosts()
-UploadPosts()
-EditPost()
-SavePost()
-SetDislikeCount()
-SetLikeCount()
-SetCommentCount()
-SetViewCount()
0..*


Verifier
- Verify()
Resources
fileURL : string
ImageURL: string
CourseID: string
- getFile()
- deleteFile()
0..*
Swaps
ID: string
Posted_at : time
Description : string
type: string
Student_ID
Expiry : time
AcceptedBy : string
- addSwap()
- deleteSwap()
- editSwap()
Donations
DonationsID : string
amountRequired : string
PostedBy : string
PostedAt : time
amountRecieved : int
details : string
- getDonations()
- deleteDonations()
- editDonations()


Academics
GPA : float
CHR_total: float
CHR_taken : float
Dislikes : int
MajorElective : float
FreeElective : float
DC : bool
- uploadTranscript()
- predictGPS()
0..*
Instructor
ID : string
name : string
School : string
Major : string
RO_rating: float
StudentRating
Bio : string
ImageURL : string
- getProfile()
- getRating()
- editProfile()
Comments
Text : string
Id: string
Likes: int
Dislikes : int
Views. : int
- getComment()
- deleteComment()
- editComment()
0..*
Reviews
R_Id : string
I_Id : string
StudentId : string
Session : string
Rating : Int
Review : string
- editReview()
- deleteReview()

Description

1. **Eateries**:

   The "eateries" table serves as the main repository for information about various eateries. It includes essential details such as a unique identifier (id), eatery name (name), contact phone number (phone number), associated email address (email), operating timings (timings), an image URL (imageurl) for visual representation, and geographic location details (location). This table centralizes comprehensive data about different eateries, facilitating easy access and management of key information for users and administrators.

2. **User**:

   The User class represents an individual with various attributes including personal information like Type, Password, Full Name, and contact details such as Email and Phone Number. It also includes educational details like Roll Number, Batch, School, Major/Minor, and relevant dates like Date of Birth and Date Created. Additional features like a profile image URL, bio, gender, and a security PIN are also included. The class tracks the number of associated accounts with Num_of_Accounts attribute. It serves as a central entity linked directly to posts, events, donations, courses, and major/minor information etc.

3. **Campus Facilities**:

   The "campus_facilities" table stores information about various facilities available on a campus. Each record in this table includes a unique identifier (id), the name of the facility (name), a contact phone number (phone), an associated email address (email), operating timings (timings), an image URL (imageurl) for visual representation, and geographic location details (location). This table serves as a central repository for managing and accessing details about campus facilities, assisting in efficient organization and utilization of resources within the campus.

4. **Resources**:

   The "resources" table is a central repository for managing various resources, containing attributes like fileurl for accessing associated files and imageurl for visual representation. The fileurl provides a link to access files related to a resource, while the imageurl offers a link for visual representation of the resource. The table includes methods such as getfile(resource_id) to retrieve the file linked to a specific resource based on its ID, and deletefile(resource_id) and deleteimage(resource_id) to respectively remove the file and image associated with the specified resource ID from the storage, enhancing efficient management and utilization of diverse resources.

5. **Events:**

   The Events class encapsulates information about organized events. It includes attributes such as EventID, Posted_at (date of posting), organizer, start_date, end_date, Title, ImageURL, Description, StartTime, EndTime, Location, venue, and category.

   Additionally, it provides several methods to manage events:

   - addEvent(): This method allows for the addition of a new event to the system.
   - deleteEvent(): It facilitates the removal of an existing event from the database.
   - editEvent(): This method enables the modification of event details.
   - getEvent(): It retrieves information about a specific event.
   - showEvent(): This method displays the details of an event.

   Together, these attributes and methods make the Events class a comprehensive tool for managing and presenting event-related information in the system.

6. **Likes**:

   The "likes" table serves as a repository for storing information about user likes for various items. Each record includes a unique identifier (p_id) for the item being liked, the user's identifier (user_id), and the timestamp when the item was liked (liked_at). This table is crucial for tracking user preferences and interactions with items, providing valuable insights into user engagement and popular content.

7. **Dislikes**:

   The "dislikes" table stores information about user dislikes for various items. Each record includes a unique identifier (p_id) for the item being disliked, the user's identifier (user_id), and the timestamp when the item was disliked (disliked_at). This table helps track user preferences and interactions with items, providing valuable insights into user engagement and areas of improvement.

8. **Settings**:

   The Settings class is responsible for managing user-specific preferences and configurations. It includes attributes like Active_Notifications (for enabled notifications), Muted_Notifications (for silenced notifications), Blocked_People (for individuals the user has blocked), and Blocked_Posts (for posts that are filtered or blocked from view). These settings allow users to customize their experience and control their interactions within the system.The Settings class provides essential methods for user customization and profile management:

   - AddAccount(): This method allows users to add additional accounts or profiles to their settings.

   - ToggleNotifications(): It enables users to switch notifications on or off, providing them with control over their notification preferences.

   - EditProfile(): This method permits users to make changes to their profile details, ensuring they can update and maintain their information as needed.

These methods complement the attributes of the Settings class, enhancing the user's ability to configure their preferences and manage their profile effectively.

9. **Tools and Subscriptions**:

   The "tools_and_subscriptions" table serves as a central repository for storing information about various tools and subscriptions available to users. Each record includes a unique identifier (id), the name of the tool or subscription (name), a brief description of its features or services (description), a link to access or learn more about the tool or subscription (link), and a URL to the icon representing the tool or subscription (iconurl). This table is essential for managing and providing information about different tools and subscriptions to users.

10. **Saved Posts**:

    The "saved posts" table stores information about posts saved by users. Each record includes the unique identifier of the user (userid), the unique identifier of the saved post (postid), and the timestamp when the post was saved (saved_at). This table helps users keep track of the posts they've saved for later viewing or reference.

11. **Swaps**:

    The "swaps" table stores information about various swap requests. Each record includes a unique identifier (id), the timestamp when the swap was posted (posted_at), a brief description of the swap request (description), the type of swap being requested (type), the unique identifier of the student posting the swap request (student_id), the expiry timestamp for the request (expiry), the timestamp when the swap was accepted (accepted_at), and the unique identifier of the student who accepted the swap (accepted_by). This table facilitates the organization and management of swap requests between students.

12. **Validator**:

    The "validator" table stores syntax patterns for validating different types of data, such as phone numbers (phone_syntax), email addresses (email_syntax), event data

(event_syntax), and donation-related data (donation_syntax). These syntax patterns can be used to validate input and ensure it adheres to the expected format for various purposes.

13. **Verifications**:

The "verification" table stores information related to user verification during login processes. Specifically, it includes an attribute for tracking the verification status of the login process (verify_login). This table is useful for managing and tracking the verification status of users attempting to log in.

14. **Comments**:

The Comments class manages individual comments within the system. It encompasses attributes such as text (the content of the comment), ID (unique identifier), likes (number of likes received), dislikes (number of dislikes received), post_id (associated post identifier), user_id (associated user identifier), and views (number of times the comment has been viewed).

The class provides the following methods:

getComment(): This method retrieves information about a specific comment.

deleteComments(): It allows for the removal of one or more comments from the system.

These methods facilitate the management and retrieval of comment-related information, contributing to an interactive and engaging user experience.

15. **Academics:**

The Academics class serves as a comprehensive repository for a user's educational records, encompassing attributes like GPA, total and taken credit hours, specific credits in electives and core courses, as well as a flag indicating degree completion. Additionally, it offers a method, uploadTranscript(), enabling users to seamlessly update their academic records by uploading their official transcripts. This class plays a pivotal role in allowing users to track and manage their academic progress within the system.

16. **Donations**:

The "donations" table stores information related to donation requests. Each record includes a unique identifier for the donation request (donationid), the amount of donation required (amountrequired), the user or entity posting the request (posted_by), the timestamp when the request was posted (posted_at), the amount of donations received (amount_received), additional details about the donation request (details), the name of the bank associated with the account for donations (bank_name), the account holder's name (account_name), the account number for donations (account_number), and the International Bank Account Number (iban). This table helps in managing and organizing donation requests and their associated details.

17. **Reviews:**

The Reviews class manages user-generated reviews within the system. It includes attributes like R_ID (Review ID), I_ID (Instructor ID), Student_ID (Student's ID), Session (Academic session for the review), CourseID (Course identifier), and Rating (Numeric rating provided by the student). This class facilitates the collection and organization of feedback on instructors and courses, enabling users to make informed decisions based on peer reviews.

18. **Courses**:

The "courses" table stores information related to academic courses. Each record includes details such as the title of the course (title), the course code (coursecode), whether it's a major or minor course (major/minor), the instructor's name (instructor), the course section (section), the semester in which the course is offered (semester), the time schedule for the course (time), prerequisites for the course (pre_req), antirequisites for the course (anti_req), academic level of the course (academicLevel), core session details (coreSessions), whether the course is a free elective (freeElective), outgroup information (outgroup), university distribution (uniDistribution), room details (room), and the credit hours for the course (credithour).

19. **Major_Minor:**

    The Major_Minor class manages information related to academic majors and minors. It includes attributes such as ID (Identifier for the major/minor), Name (Name of the major/minor), Category (Category to which it belongs), and schoolID (Identifier for the associated school or institution). This class serves as a fundamental component for organizing and cataloging academic programs within the system.

20. **Location**:

    The "location" table stores information related to different locations. Each record includes a unique identifier (l_id), the name of the location (name), and the coordinates of the location (coordinates). This table is useful for managing and organizing location-related data.

21. **Posts**:

    The Posts class manages user-generated content in the system. It includes attributes such as UniqueID (Unique identifier for the post), Roll_number (User's roll number), CreatedAt (Date of creation), Like_count (Number of likes), Dislike_count (Number of dislikes), Views (Number of views), Bookmarks (Number of times bookmarked), edited_at (Date of last edit), edited (Flag indicating if the post has been edited), and comment_count (Number of comments).The class provides the following methods:

    - GetPosts(): This method retrieves posts from the system.
    - UploadPost(): It allows users to upload a new post.
    - savePost(): This method saves changes made to an existing post.

    These methods enable users to interact with and manage posts effectively, contributing to a dynamic and engaging user experience within the system.

22. **Room**:

The "room" table is a fundamental repository storing essential details about rooms within buildings. It encompasses attributes such as the room name (name), a unique identifier for the room (roomID), and the identifier of the building to which the room belongs (buildingID). The getRoom method complements this table, facilitating the retrieval of room information based on the provided room identifier, aiding in efficient access and management of details about specific rooms within various buildings. Together, these components form a structured system to organize and retrieve room-related data, contributing to effective facility management.

23. **Admin Offices**:

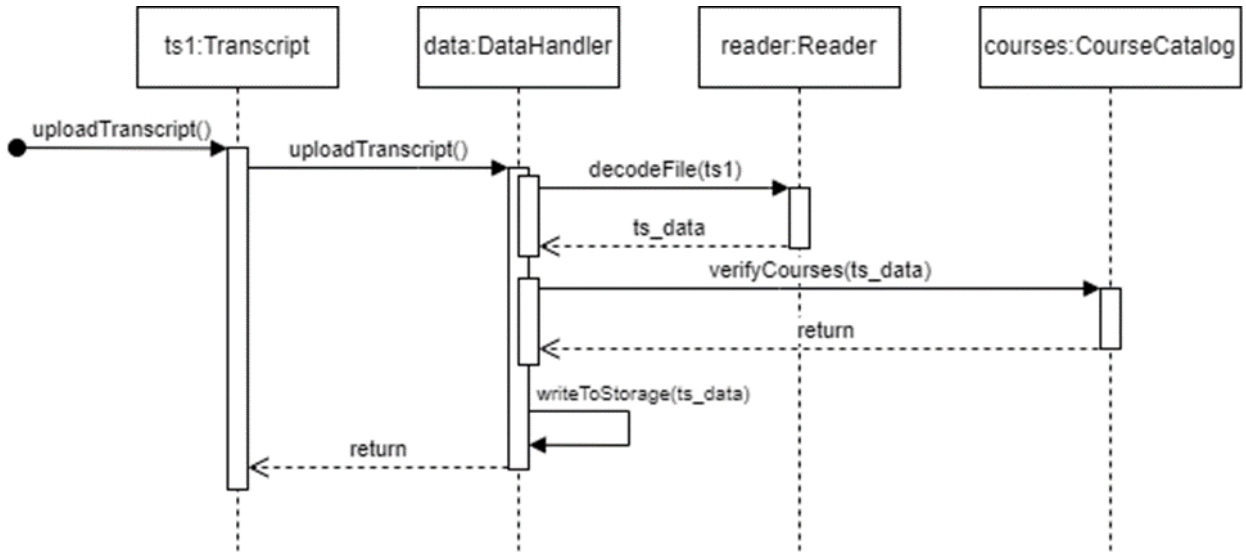
The Admin Offices class handles information related to administrative offices within the system. It encompasses attributes like ID (Identifier for the office), Name (Name of the office), PhoneNum (Contact phone number), Email (Contact email address), Timings (Operating hours), ImageURL (URL for office image), and Location (Physical address). This class serves as a centralized hub for organizing and accessing administrative office details, providing users with essential contact information and resources.

24. **Instructor**:

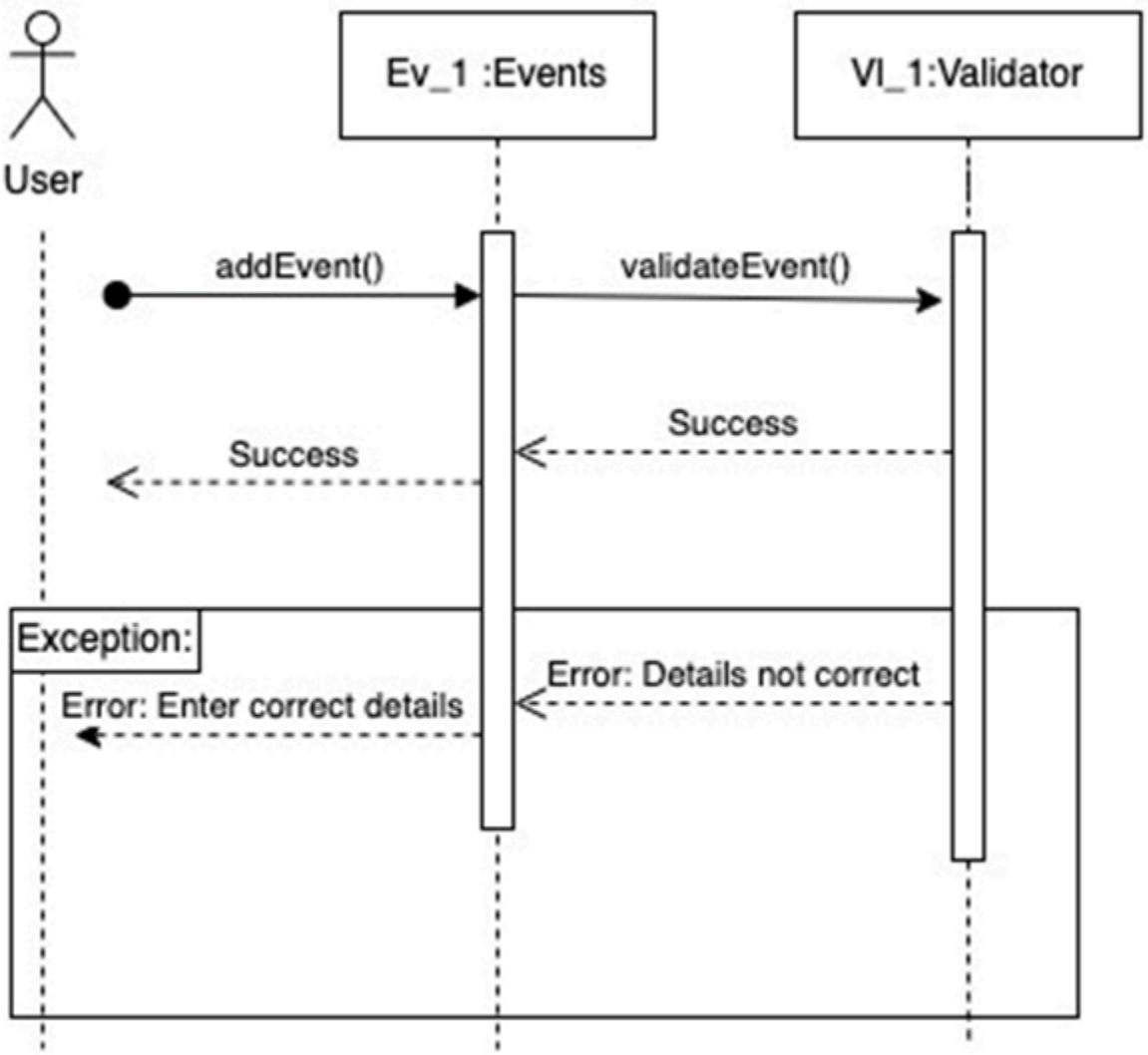
The "instructor" table serves as a comprehensive repository for vital information about instructors, containing attributes such as a unique identifier (id), the instructor's name (name), affiliated school (school), major or minor status (major/minor), ratings from both fellow instructors (ro_rating) and students (student_rating), a brief biography (bio), and an image URL (imageurl). The getProfile method allows for the retrieval of an instructor's profile based on their unique ID, presenting details like name, school, status, ratings, biography, and image URL. On the other hand, the getRating method provides the ratings given by fellow instructors and students for a specific instructor, aiding in the assessment of their performance and effectiveness in an educational context. This table and its associated methods streamline the organization and access of crucial information about instructors, promoting efficient insights into their profiles and ratings.

c. Sequence Diagrams

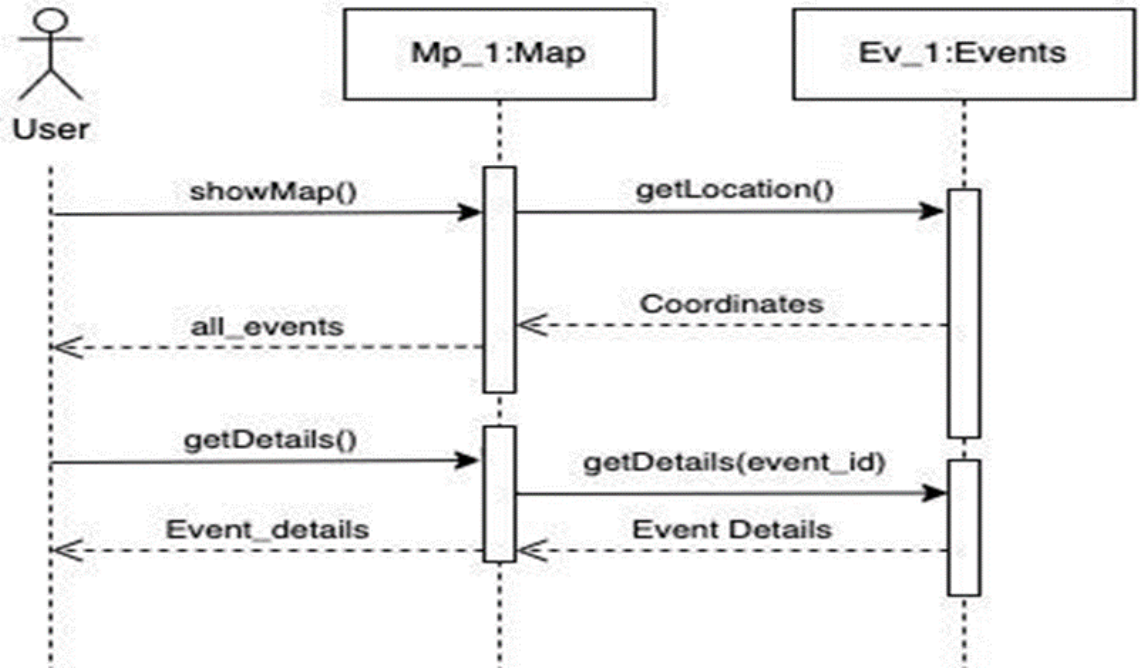
Add Transcript



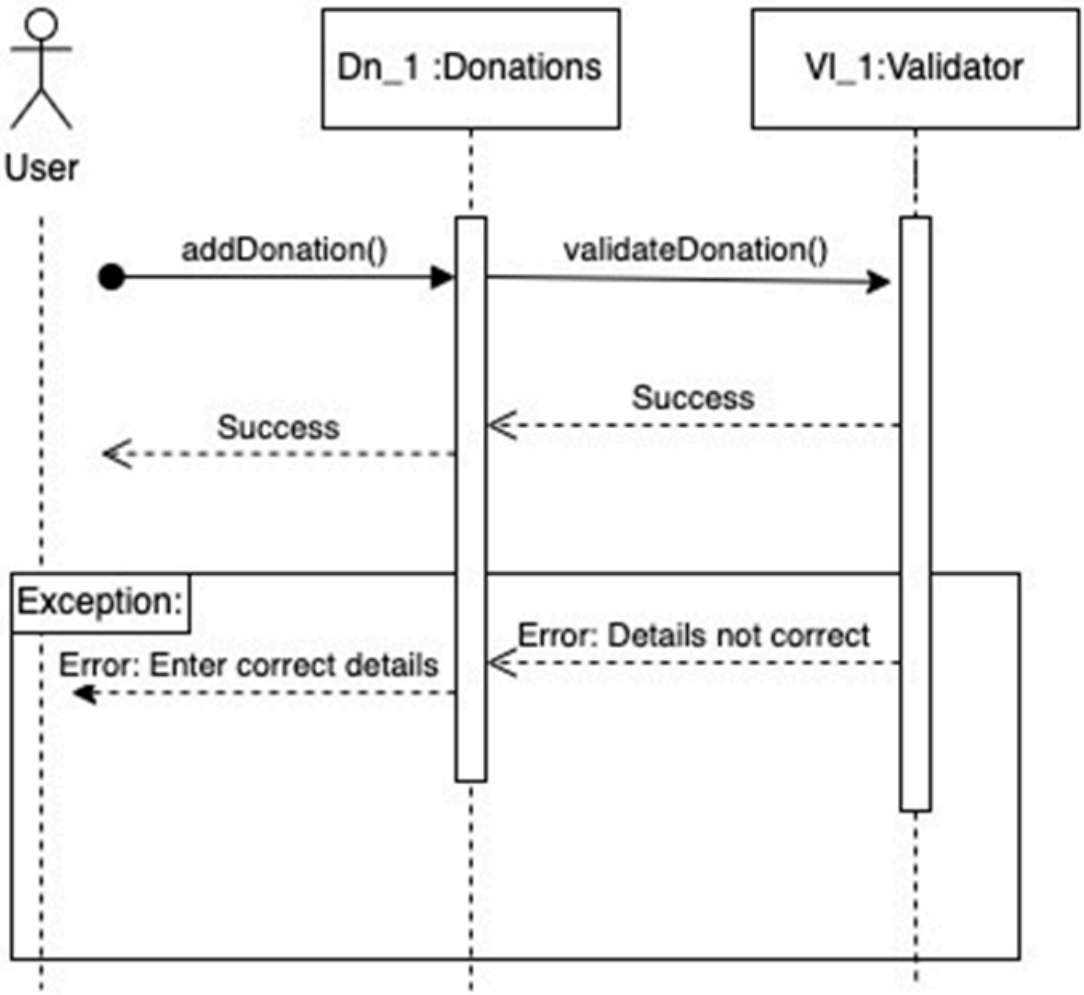
Add Event


User
Ev_1 :Events
Vl_1:Validator
addEvent()
validateEvent()
Success
Success
Exception:
Error: Details not correct
Error: Enter correct details

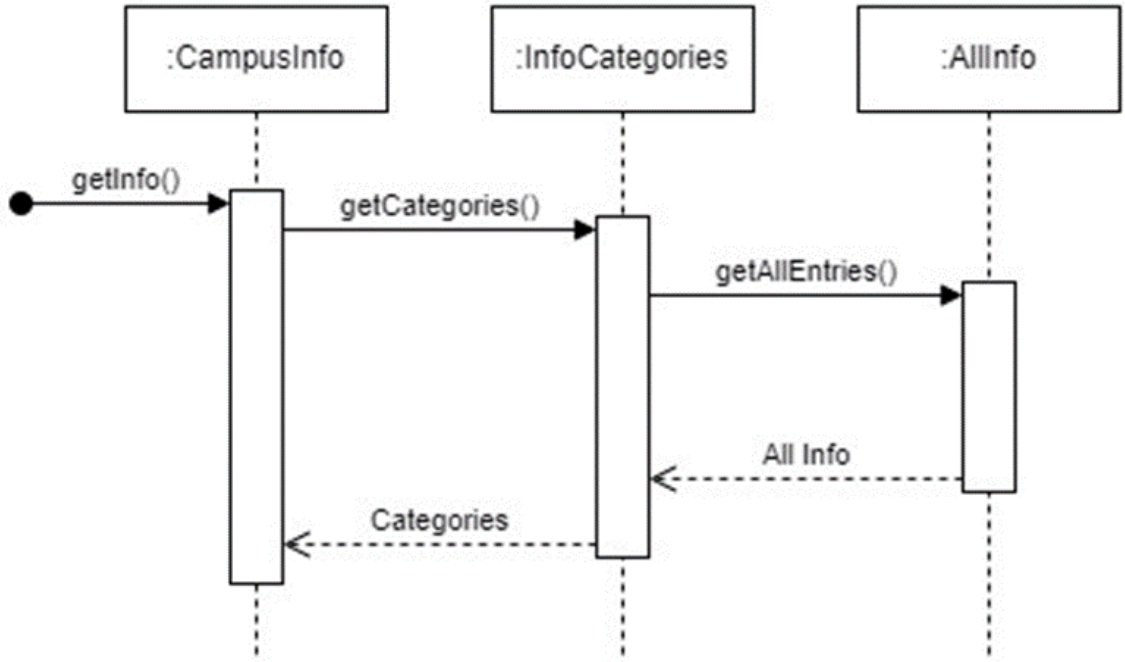
Show Event on Map



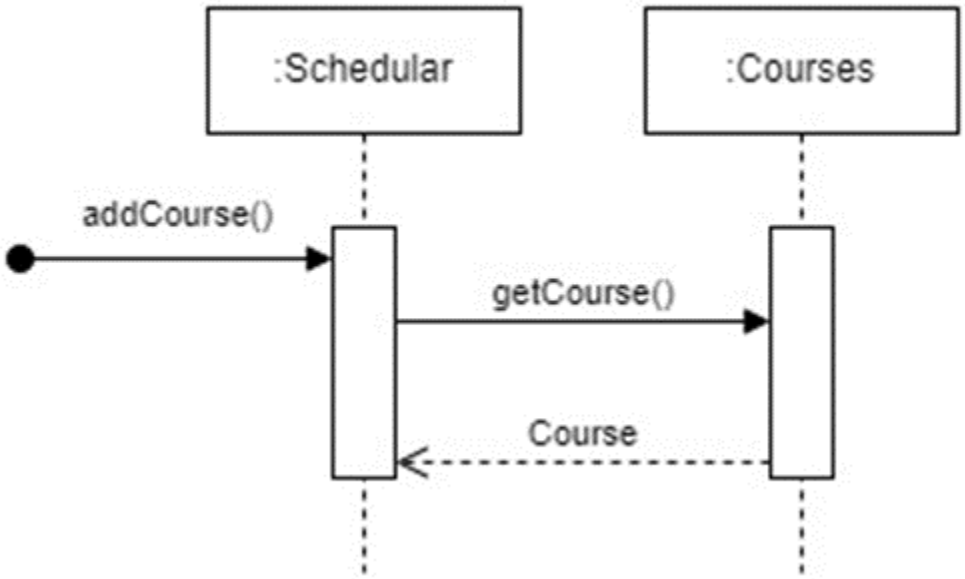
Add Donation



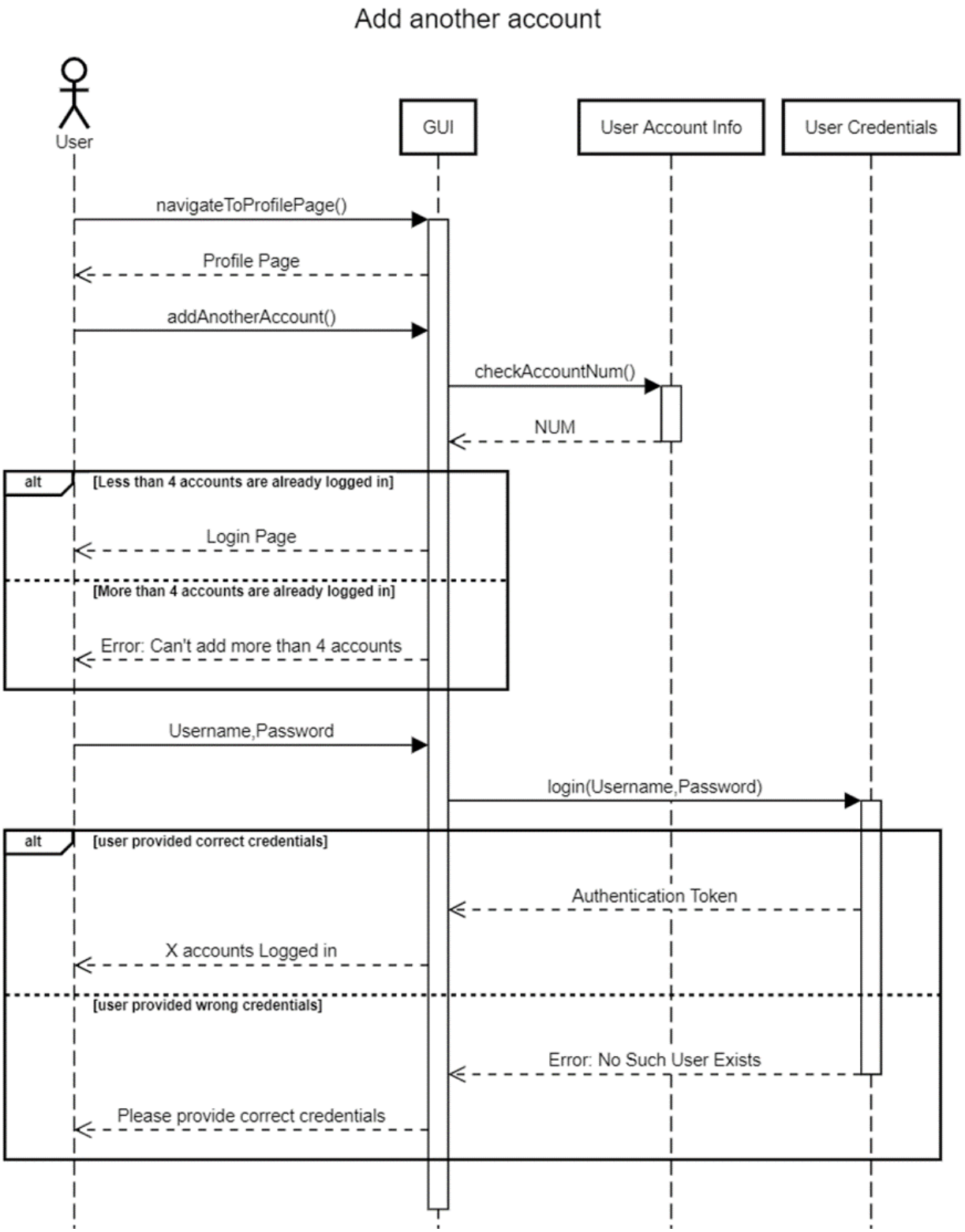
View Campus Info



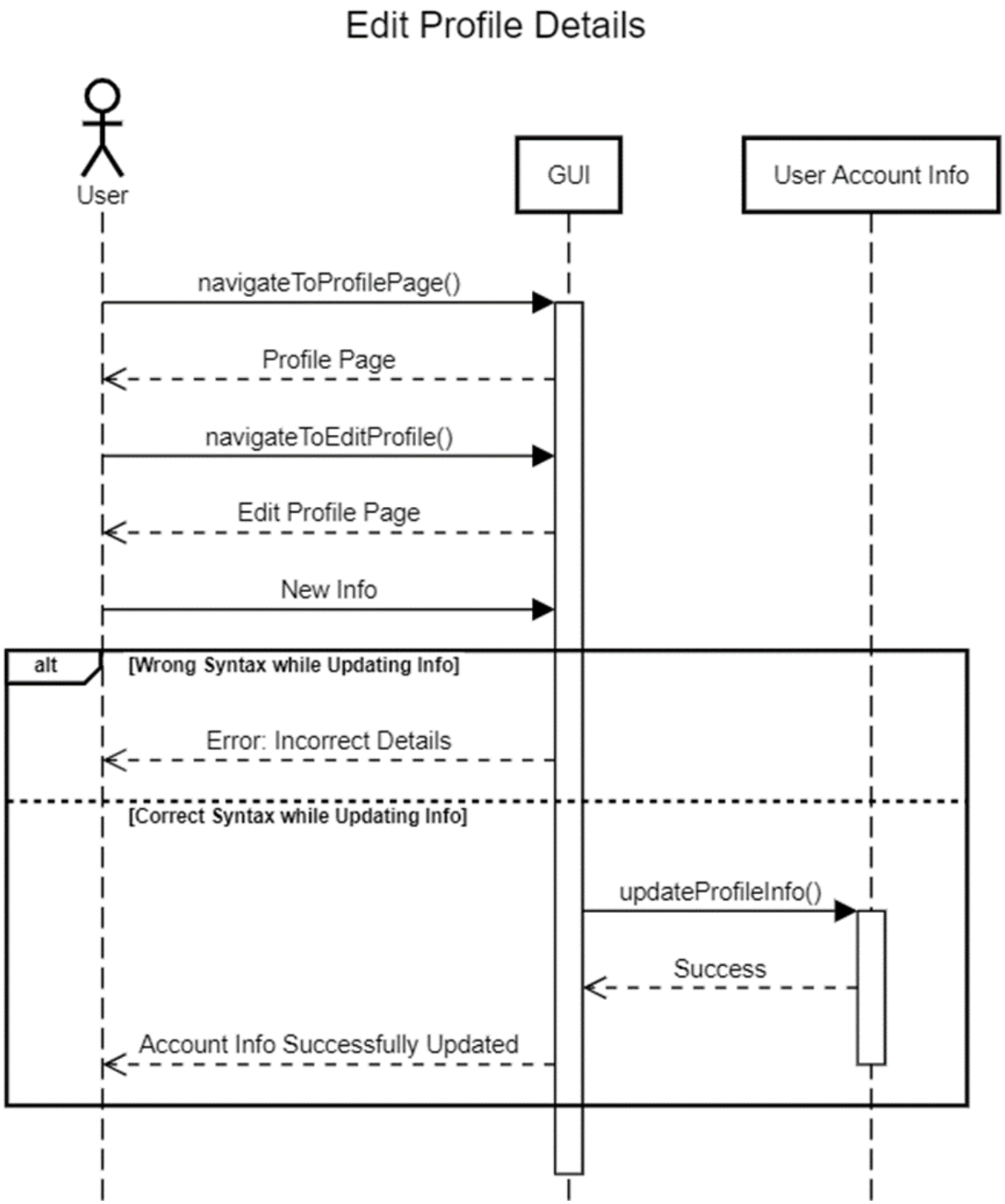
Build a schedule



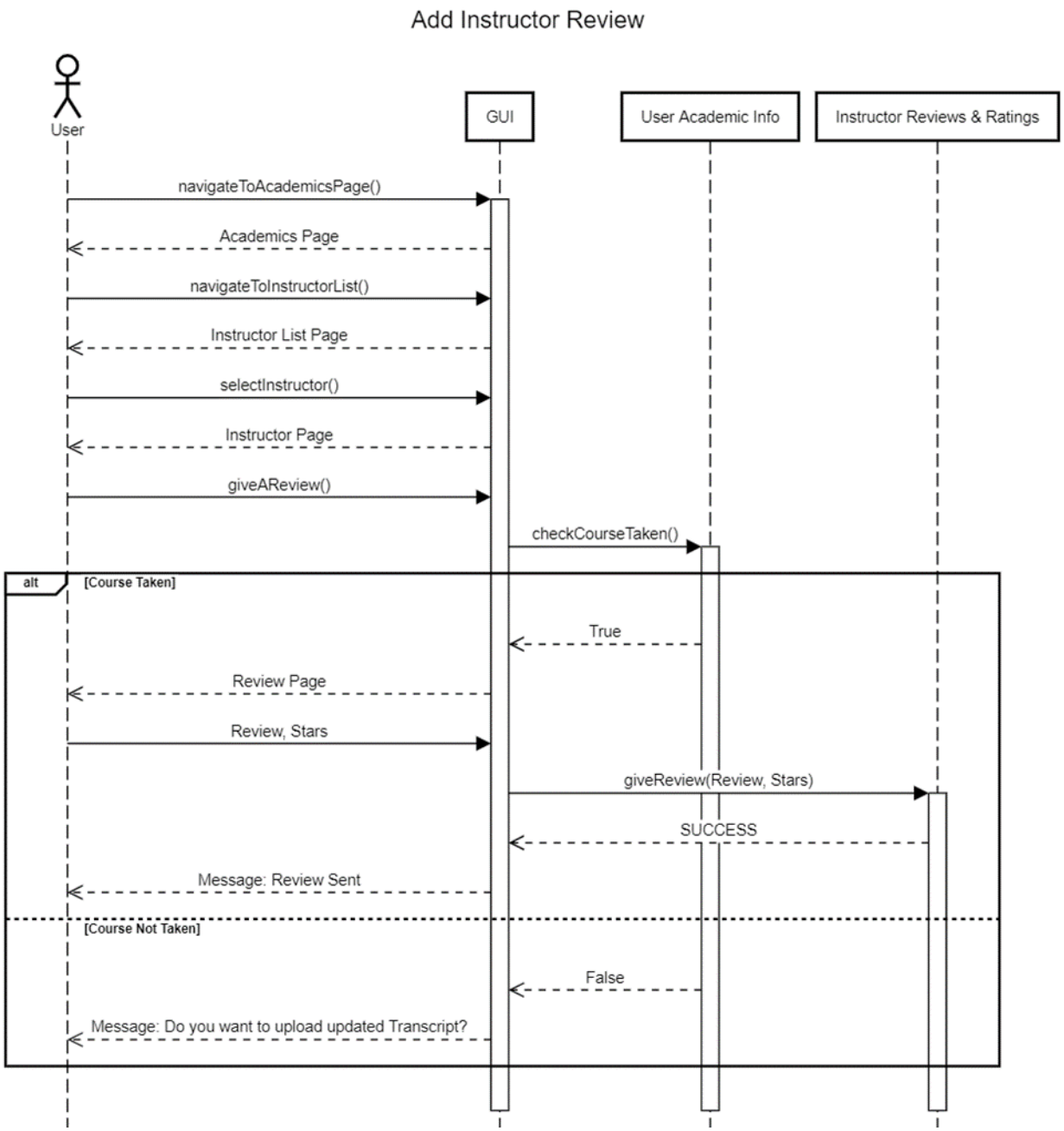
Add Another Account



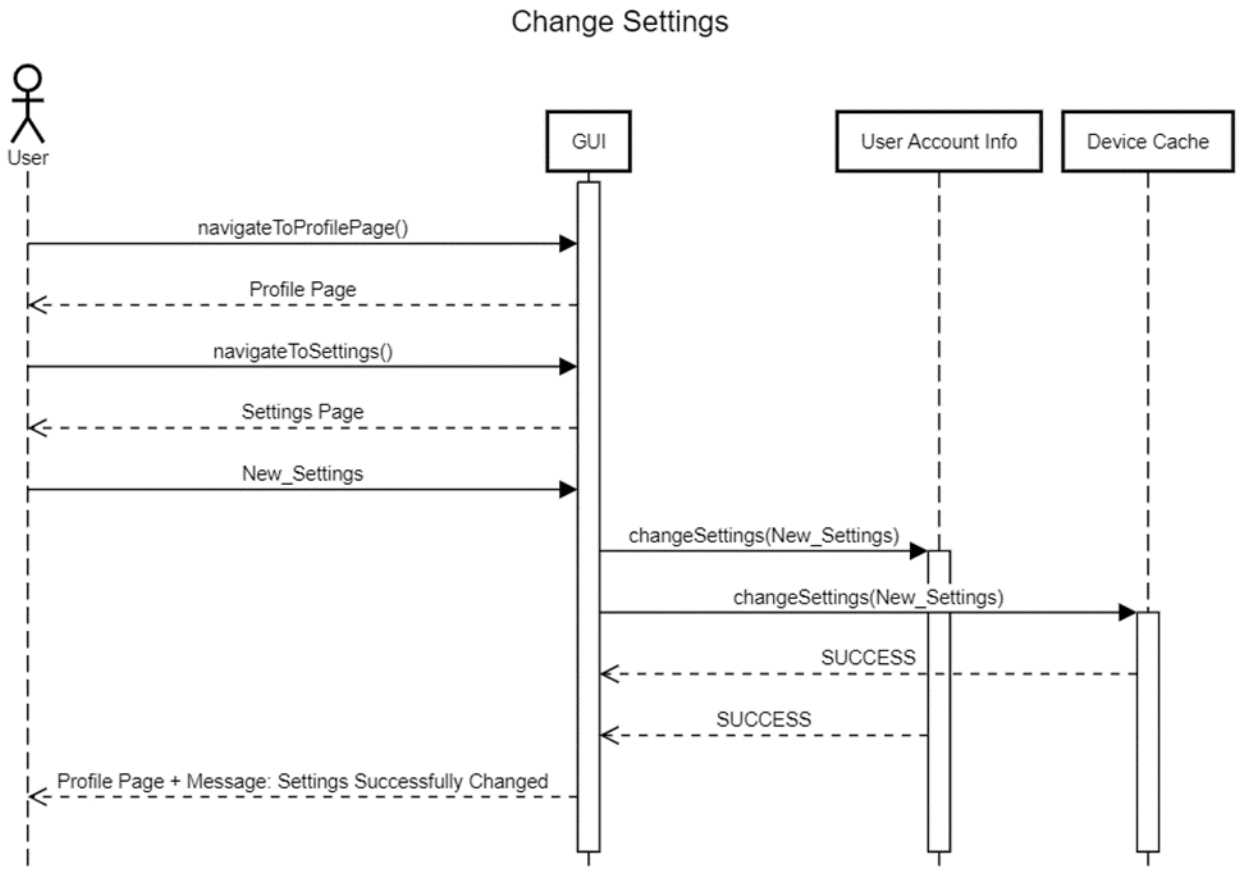
Edit Profile



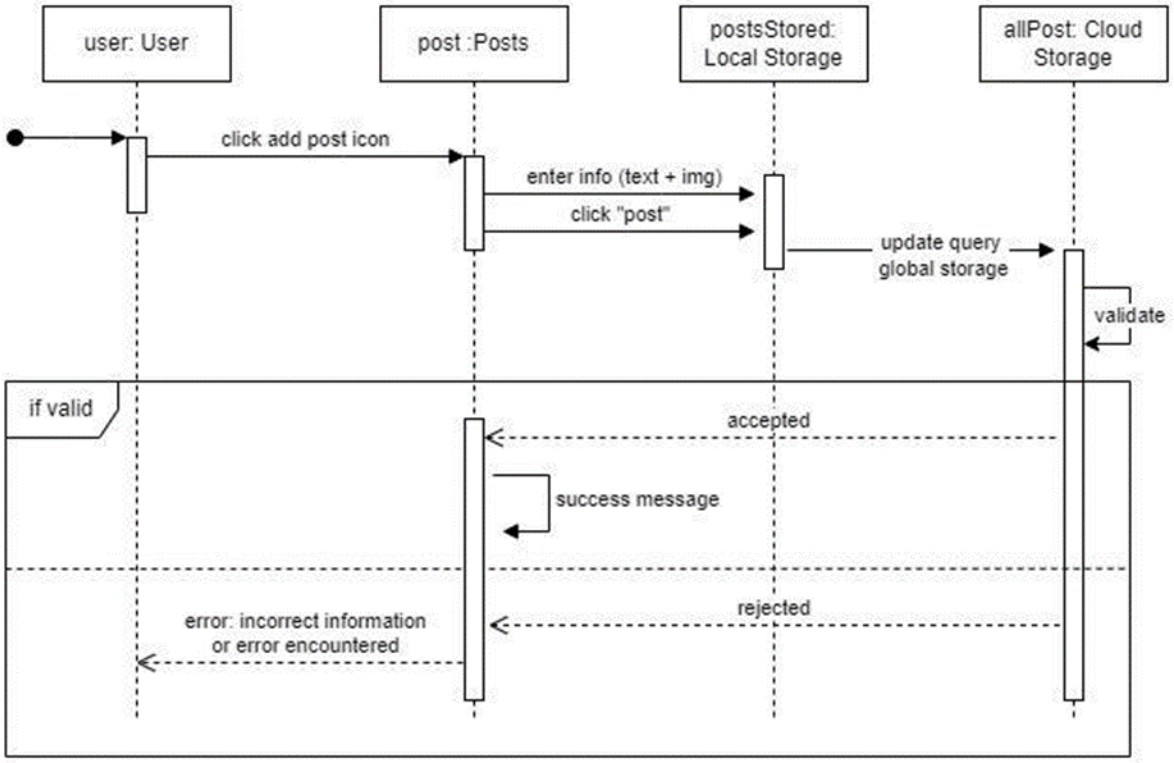
Add Instructor Review



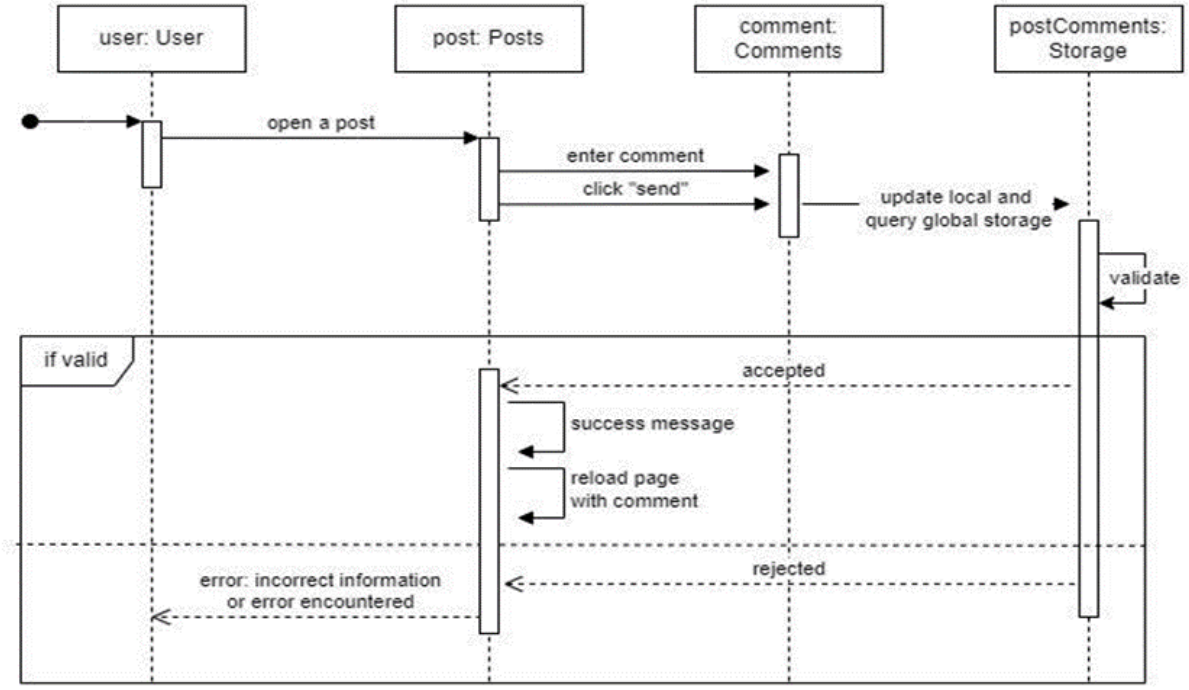
Change Settings



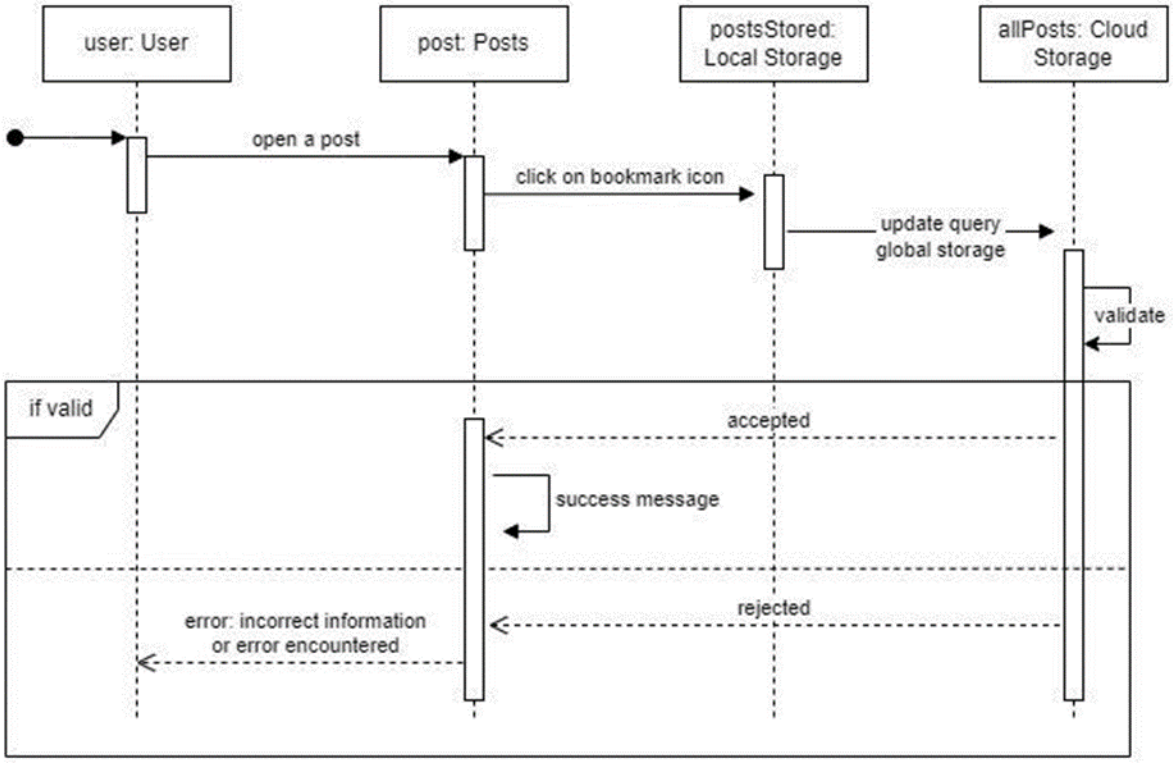
Add Post to LDF



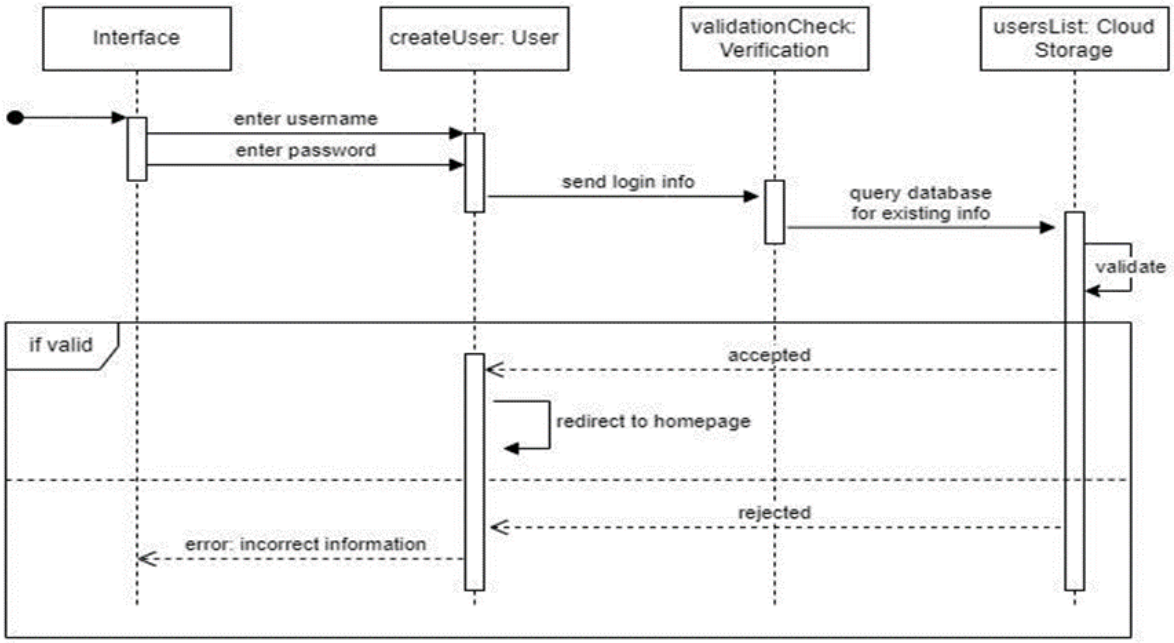
Add Comment to Post



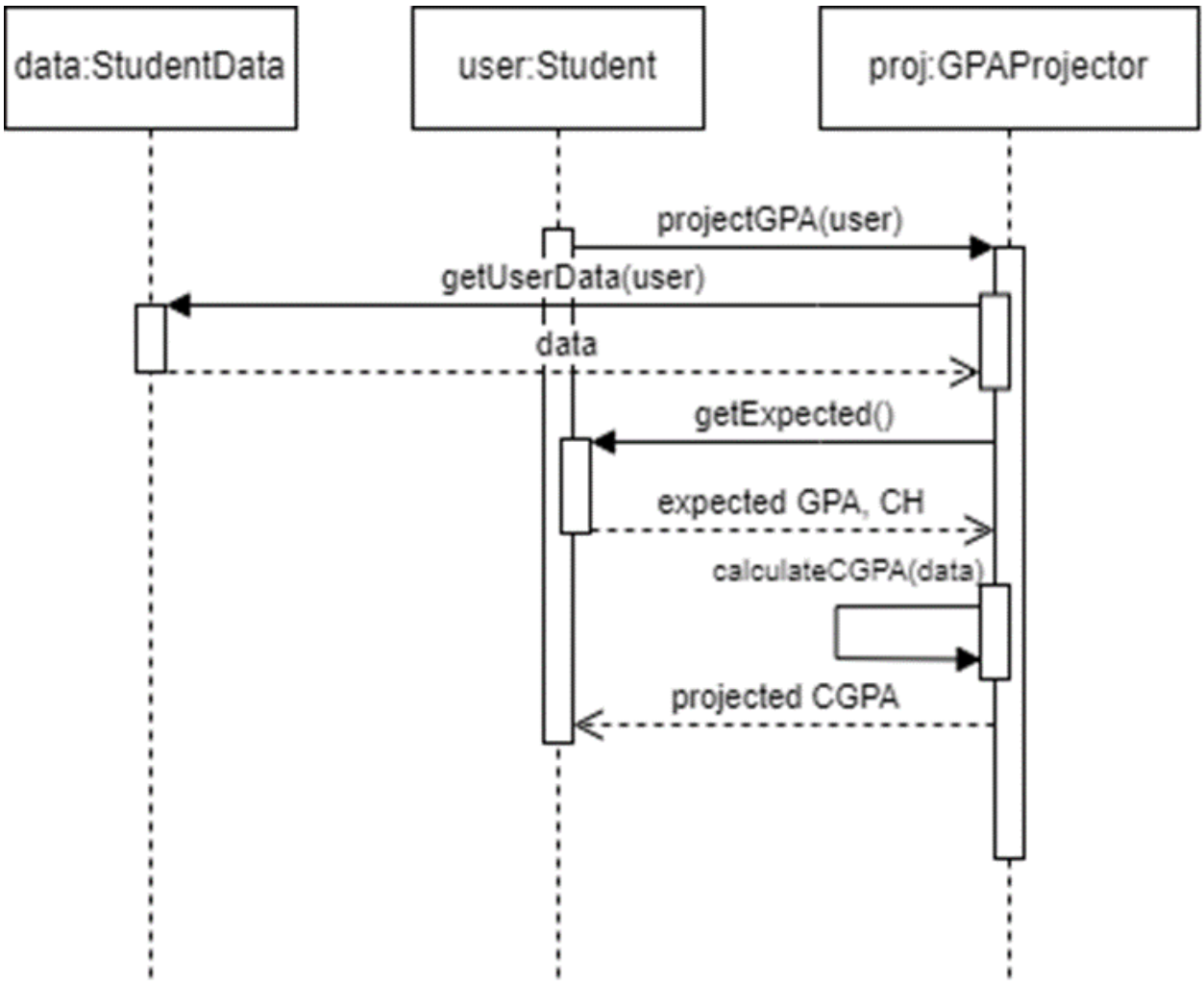
Bookmark post on LDF



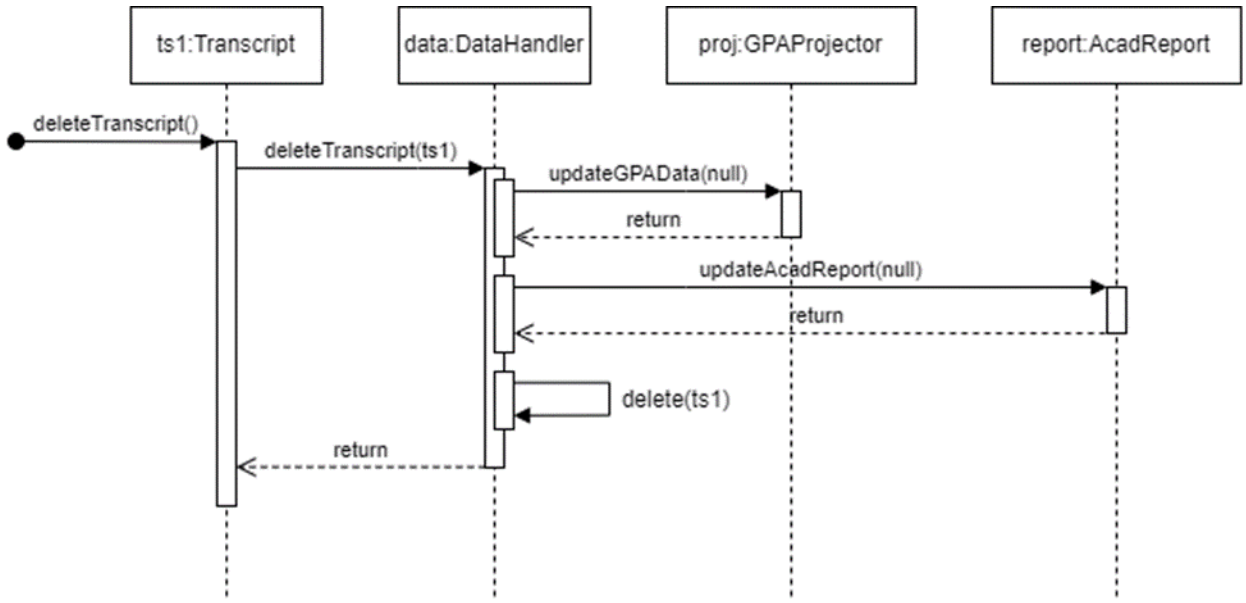
Login



Predict GPA


data:StudentData
user:Student
proj:GPAProjector
projectGPA(user)
getUserData(user)
data
getExpected()
expected GPA, CH
calculateCGPA(data)
projected CGPA

Delete Transcript



5. Software Development Methodology and Plan

Waterfall Method:

Pros:

1. The structured approach allows for proper time allocation to each process. This means that all ideas are flushed out and therefore, when moving forward, there is little need to revisit previous steps.
2. Structured approach promotes a step by step approach with proper guidelines and flow.
3. It allows proper understanding of the project scope by all parties which prevents future misunderstandings.
4. Waterfall process places great emphasis on documentation at every step, this allows for new developers / users to easily understand the workflow and be caught up with its requirements without having to debug using the trial and error method.

5. Due to its step by step approach, milestones set and achieved provide a clear marker to how much progress is done.

Cons:

1. Too much documentation can cause unnecessary delay in the actual production of the system.
2. It prevents clients from giving their feedback as the process progresses which can create a system that does not fit the client's vision for it.
3. The separation of the process hinders changes in requirements as a new change forces all steps to be halted and the process to be started from the start.
4. It requires a precise and clear concept of all system requirements and as such, it is mostly useful for large scale organizations which have a total understanding of their requirements.
5. A working system becomes available only at the end of the process.

Agile (SCRUM) Method:

Pros:

1. The incremental process allows for responsive requirement changes which helps clients keep in touch with their system and mold it to their vision of the system.
2. In this, each team member can choose their own responsibilities while still working in the team. This freedom allows those with better understanding of specific topics to work on their specialities while also allowing them to help others where they are stuck.
3. Due to the continuous development, it allows for bugs and issues to be identified and then corrected which can prevent bigger issues from occurring later on.
4. It also is a faster process for creating systems as it does not waste too much time on documentation as it focuses more on implementation of the system.

Cons:

1. It puts pressure on the scrum master to handle all processes i.e., who does what and if the work being done is being done at an acceptable pace.
2. Due to the client's constant connection, it becomes challenging to keep up with client demands (especially when multiple stakeholders are involved) while also developing the system.
3. As a result of its lack of documentation, new developers / users can find it challenging to change or understand the pre existing setup.
4. Scrum methodology requires all resources at once, as all actors are working on the system on different requirements. This forces all resources to be engaged.

a. Software Process Selection

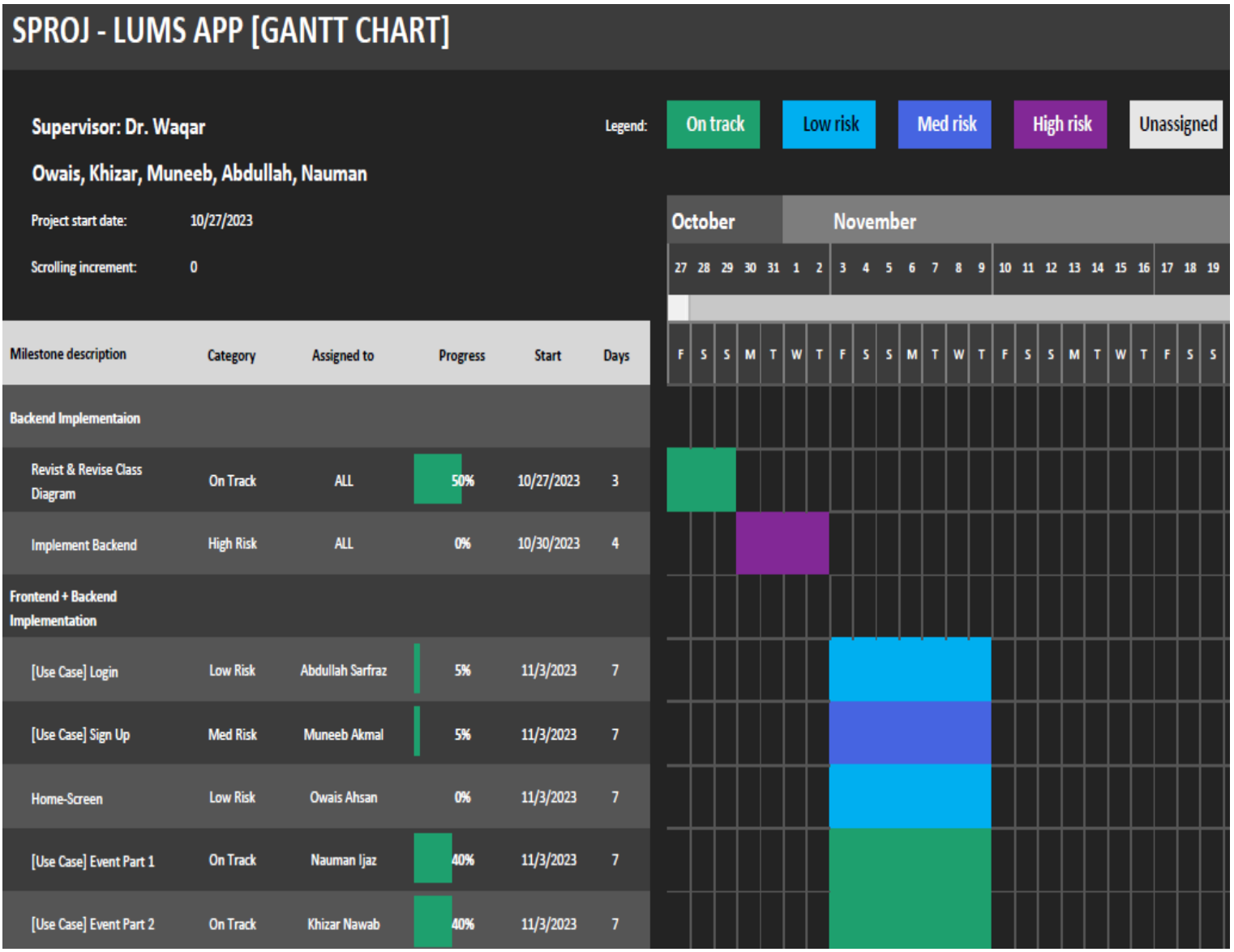
We will use the “Agile (SCRUM) Method”. This is due to our requirements (given below in the Project Context Analysis diagram) and its compatibility with maximizing the pros of selected methods and minimizing the cons.

Project Context Analysis			
	Low	Medium	High
Potential loss due to defects/bugs		It can prevent users from accessing the system but it is not a significant failure	
Developers’ experience/skills		The team has experience building website and mobile applications	
Rate of requirements change		Requirements may evolve as new requirements come in but the standard model will remain the same	
Team Size (5, 10, 25, 50, 100+)	Low number of members (5) working on development		
Organization culture (adaptive to change)			As we learn about new tools, we are adaptive to change and improve our systems
Pressure to develop early releases	There is no pressure to develop early. Quality material is needed within a		

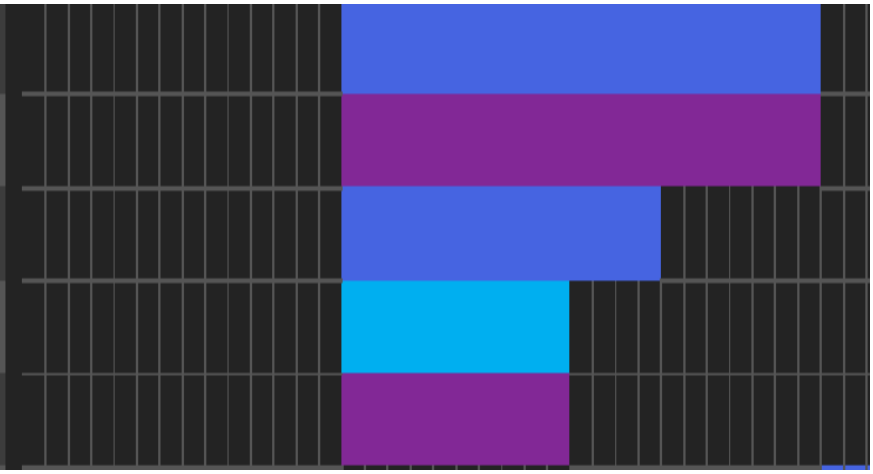
	deadline		
Business staff's commitment to work extensively with development team			The business staff is fully committed to working with the team
Developer's experience with similar systems		Some developers have experiences similar to our new system	
Availability of reusable components		Reusable components are available such as forms, authentications systems and so on.	

b. Gantt Chart

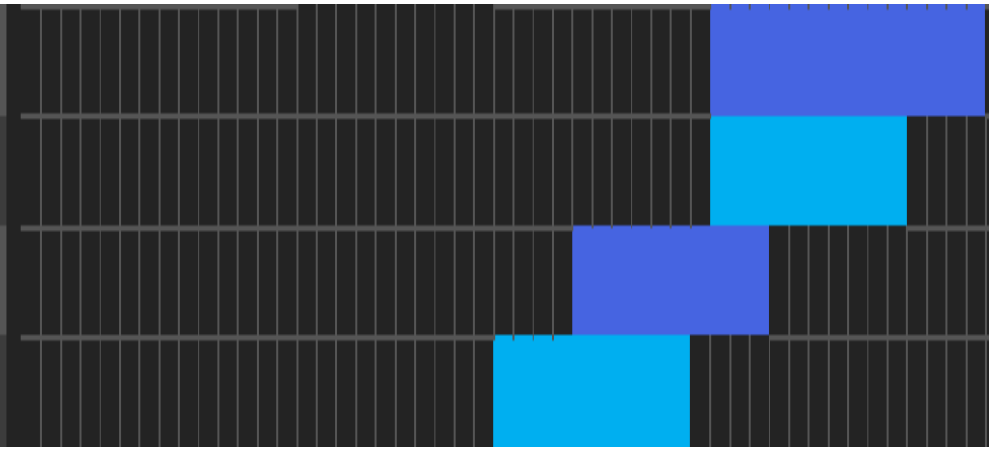
Zooming In:



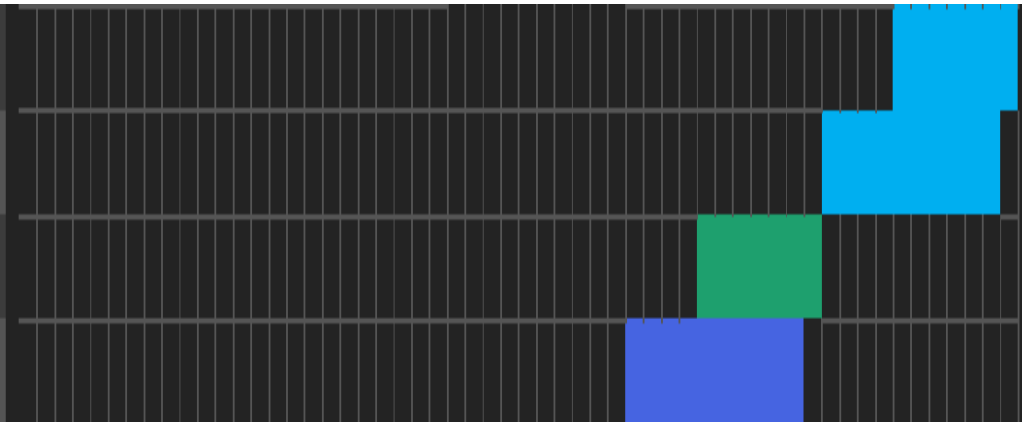
[Use Case] LDF Part 1 (Add Post etc)	Med Risk	Abdullah Sarfraz	0%	11/10/2023	21
[Use Case] LDF Part 2 (Show Posts etc)	High Risk	Muneeb Akmal	0%	11/10/2023	21
[Use Case] Scheduler	Med Risk	Owais Ahsan	0%	11/10/2023	14
[Use Case] Academic Record (Transcript)	Low Risk	Nauman Ijaz	0%	11/10/2023	10
[Use Case] Chat	High Risk	Khizar Nawab	0%	11/10/2023	10



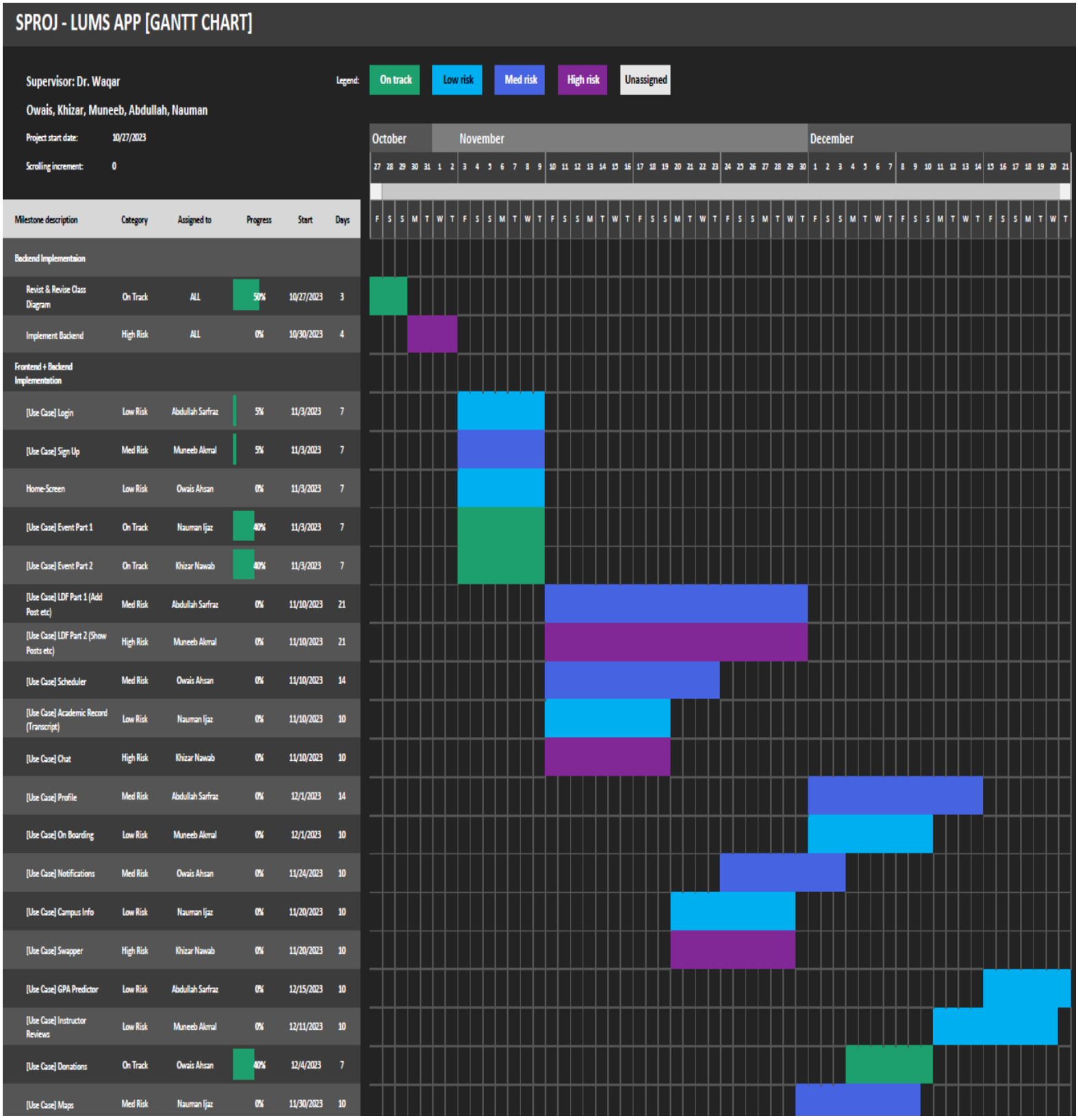
[Use Case] Profile	Med Risk	Abdullah Sarfraz	0%	12/1/2023	14
[Use Case] On Boarding	Low Risk	Muneeb Akmal	0%	12/1/2023	10
[Use Case] Notifications	Med Risk	Owais Ahsan	0%	11/24/2023	10
[Use Case] Campus Info	Low Risk	Nauman Ijaz	0%	11/20/2023	10



[Use Case] GPA Predictor	Low Risk	Abdullah Sarfraz	0%	12/15/2023	10
[Use Case] Instructor Reviews	Low Risk	Muneeb Akmal	0%	12/11/2023	10
[Use Case] Donations	On Track	Owais Ahsan	40%	12/4/2023	7
[Use Case] Maps	Med Risk	Nauman Ijaz	0%	11/30/2023	10



Compiled Form:

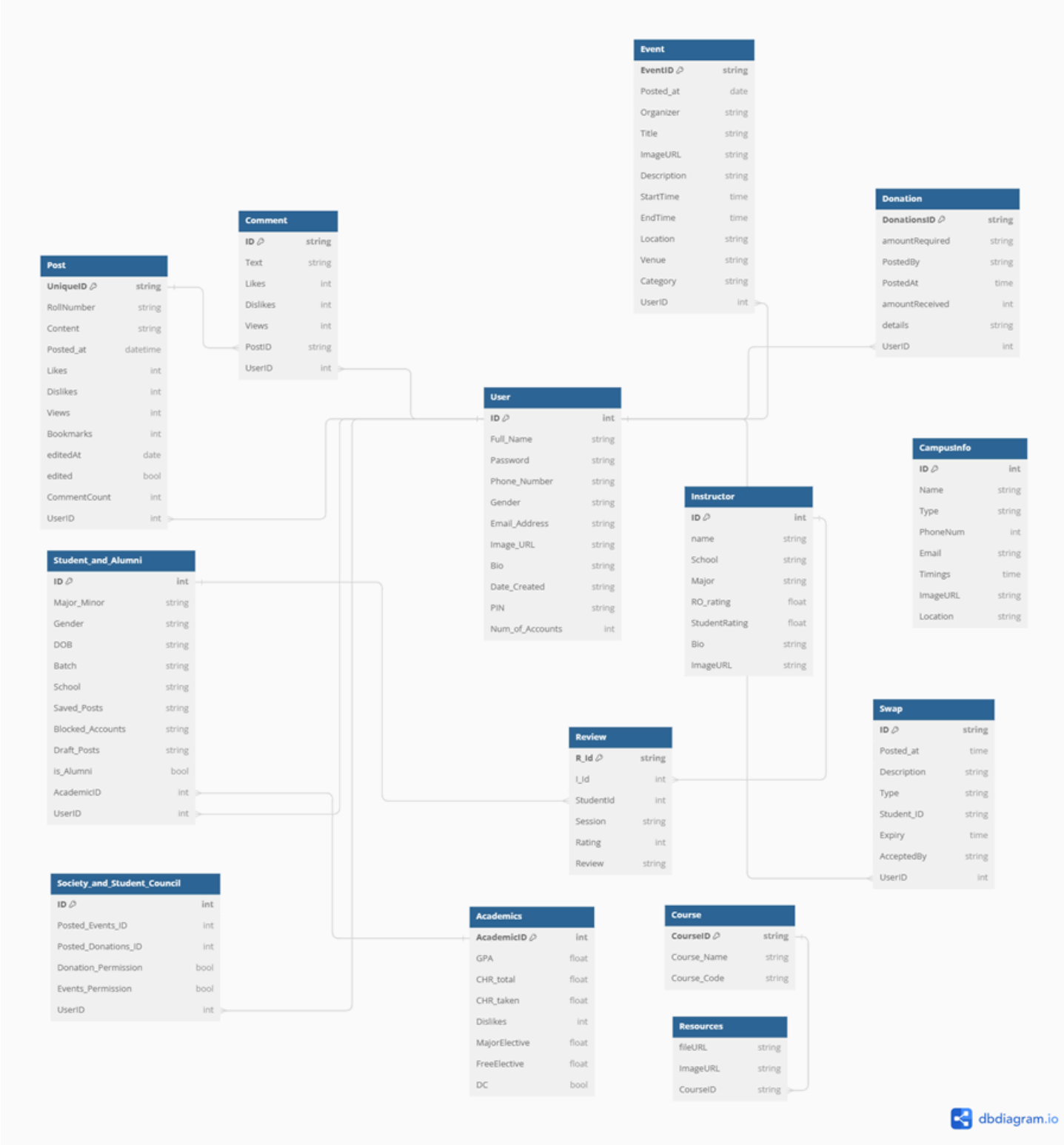


4. Database Design and Web Services

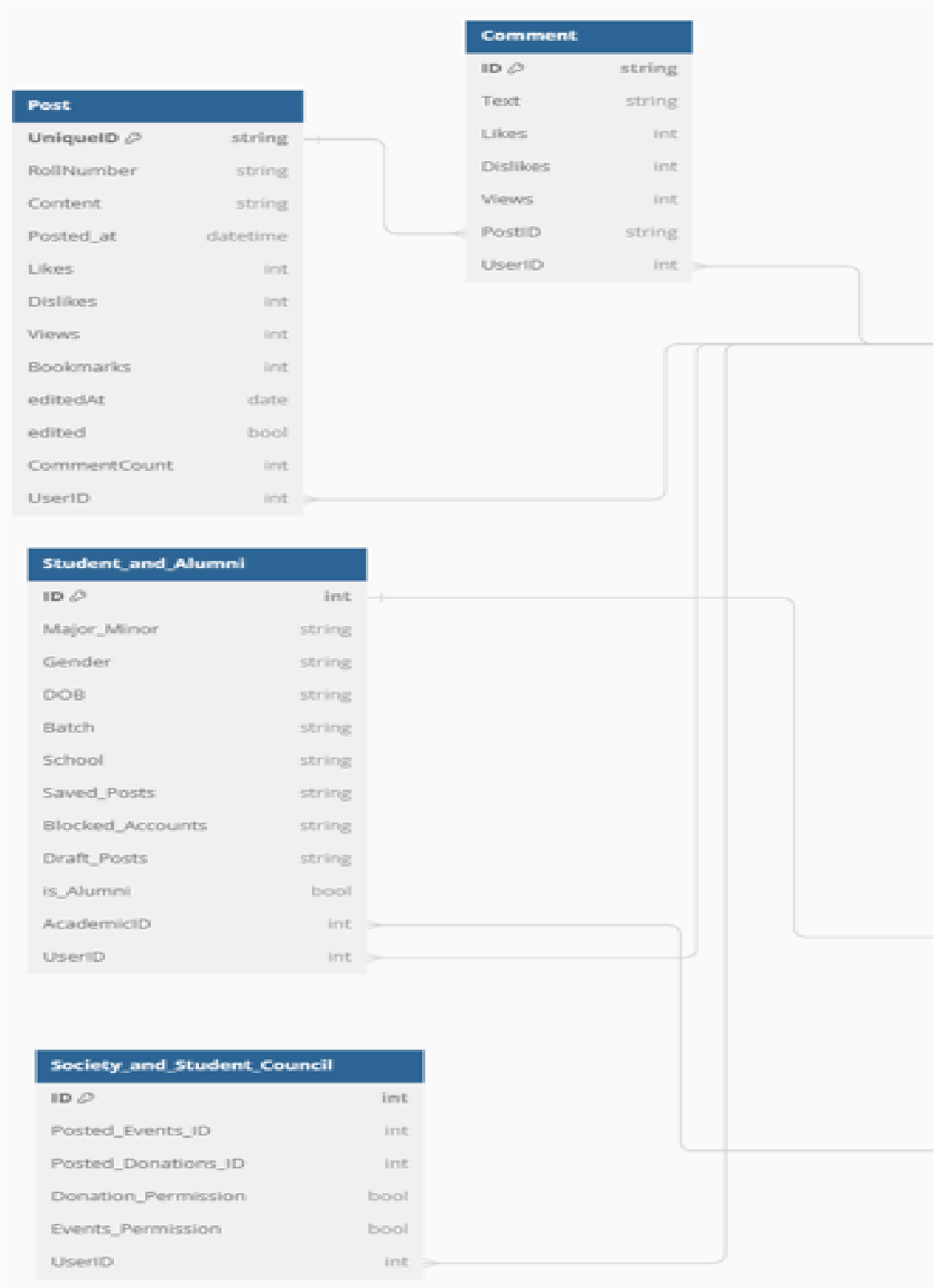
Due to using MongoDB, our database design is highly flexible, adaptive, and iteratively changing. This means that we do not have or require a predefined schema, however, we do have defined mongoose models for each object configured on our server.

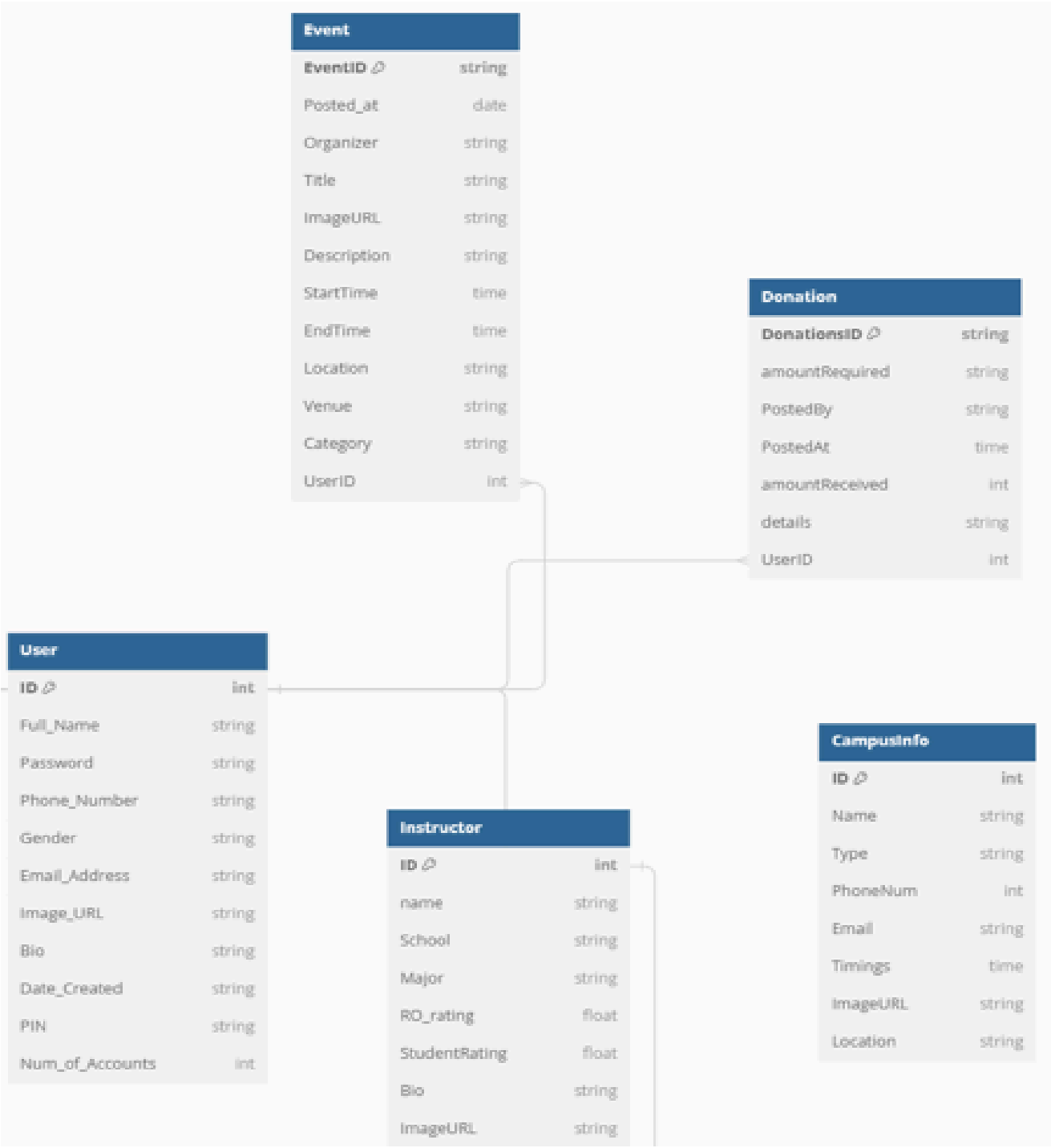
We have utilized a number of web services in our application, ranging from GPS to Content Delivery. These services are consolidated within our app to provide one seamless experience.

a. Database Design



Zooming in


Comment
ID string
Text string
Likes int
Dislikes int
Views int
PostID string
UserID int
Post
UniqueID string
RollNumber string
Content string
Posted_at datetime
Likes int
Dislikes int
Views int
Bookmarks int
editedAt date
edited bool
CommentCount int
UserID int
Student_and_Alumni
ID int
Major_Minor string
Gender string
DOB string
Batch string
School string
Saved_Posts string
Blocked_Accounts string
Draft_Posts string
is_Alumni bool
AcademicID int
UserID int
Society_and_Student_Council
ID int
Posted_Events_ID int
Posted_Donations_ID int
Donation_Permission bool
Events_Permission bool
UserID int


Event
EventID string
Posted_at date
Organizer string
Title string
ImageURL string
Description string
StartTime time
EndTime time
Location string
Venue string
Category string
UserID int
Donation
DonationsID string
amountRequired string
PostedBy string
PostedAt time
amountReceived int
details string
UserID int
User
ID int
Full_Name string
Password string
Phone_Number string
Gender string
Email_Address string
Image_URL string
Bio string
Date_Created string
PIN string
Num_of_Accounts int
Instructor
ID int
name string
School string
Major string
RO_rating float
StudentRating float
Bio string
ImageURL string
CampusInfo
ID int
Name string
Type string
PhoneNum int
Email string
Timings time
ImageURL string
Location string

Details of a few main classes are as follows:

1. **User**: The User class represents an individual with various attributes including personal information like Password, Full Name, and contact details such as Email. Additional features like a profile image URL, bio, and a security PIN are also included.
2. **Posts**: The Posts class manages user-generated content in the system. It includes attributes such as UniqueID (Unique identifier for the post), Roll_number (User's roll number), PostedAt (Date of creation), Likes (Number of likes), Dislikes (Number of dislikes), Views (Number of views), Bookmarks (Number of times bookmarked), editedAt (Date of last edit), edited (Flag indicating if the post has been edited), and CommentCount (Number of comments).

3. **Comments**: The Comments class manages individual comments of each post within the system. It encompasses attributes such as text (the content of the comment), ID (unique identifier), likes (number of likes received), dislikes (number of dislikes received), post_id (associated post identifier), user_id (associated user identifier).
4. **Societies and Student Council**: This class inherits the “User” class and forms a different category of users. It limits the interactivity of the user to relevant features, which in this case include LDF posts, Comments, Events and exclude academic features.
5. **Students and Alumni**: This class also inherits the “Users” class and limits the interactivity of a subset of users to relevant features and manages rights to add events and donations.
6. **Review**: The Review class manages user-generated reviews within the system. It includes attributes like R_ID (Review ID), I_ID (Instructor ID), Student_ID (Student's ID), Session (Academic session for the review), and Rating (Numeric rating provided by the student).
7. **Instructors**: The "instructor" table serves as a comprehensive repository for vital information about instructors, containing attributes such as a unique identifier (id), the instructor's name (name), affiliated school (school), major or minor the instructor is affiliated with (major), ratings from both the official ratings from RO (RO_rating) and students (StudentRating), a brief biography (bio), and an image URL (imageURL).
8. **Events**: The Events class encapsulates information about organized events. It includes attributes such as EventID, Posted_at (date of posting), organizer, Title, ImageURL, Description, StartTime, EndTime, Location, venue, and category.
9. **Courses**: The "courses" class stores information related to academic courses. Each record includes details such as the title of the course (Course_Name), the course code (Course_Code).

b. API Specification

This sub-section will contain the list of external APIs’ that you have used in your project.

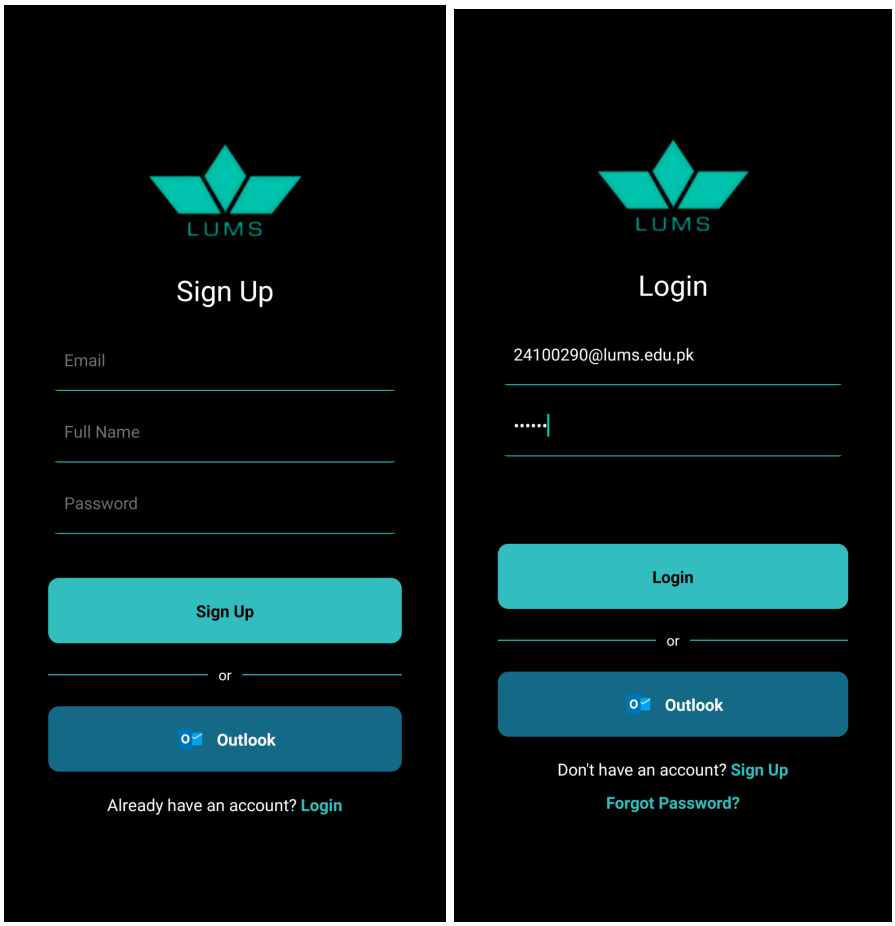
1. Google Maps API
2. Expo Push Notifications (using Firebase under the hood)
3. Firebase Realtime Database for managing user Push Tokens
4. EAS (Expo Application Services) for development and production builds

5. Cloudinary CDN for media
6. Sendgrid for sending OTP emails

5. System User Interface

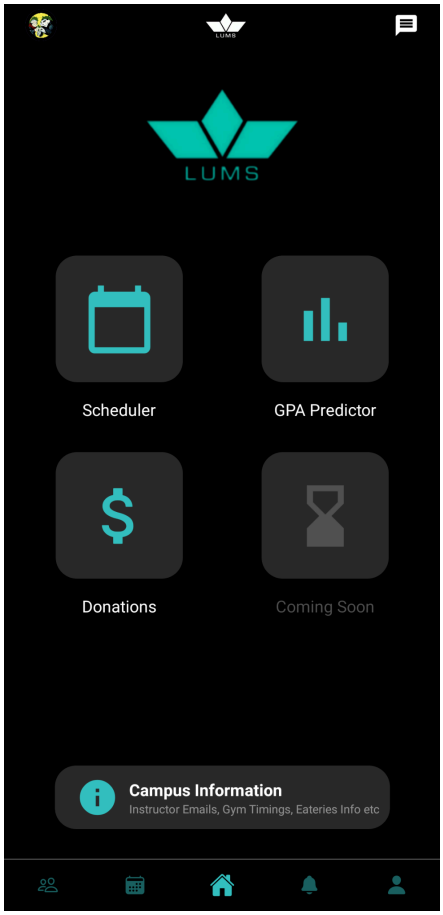
This section provides a quick interface guide for users, helping them explore the features of our application.

Signup/Login:



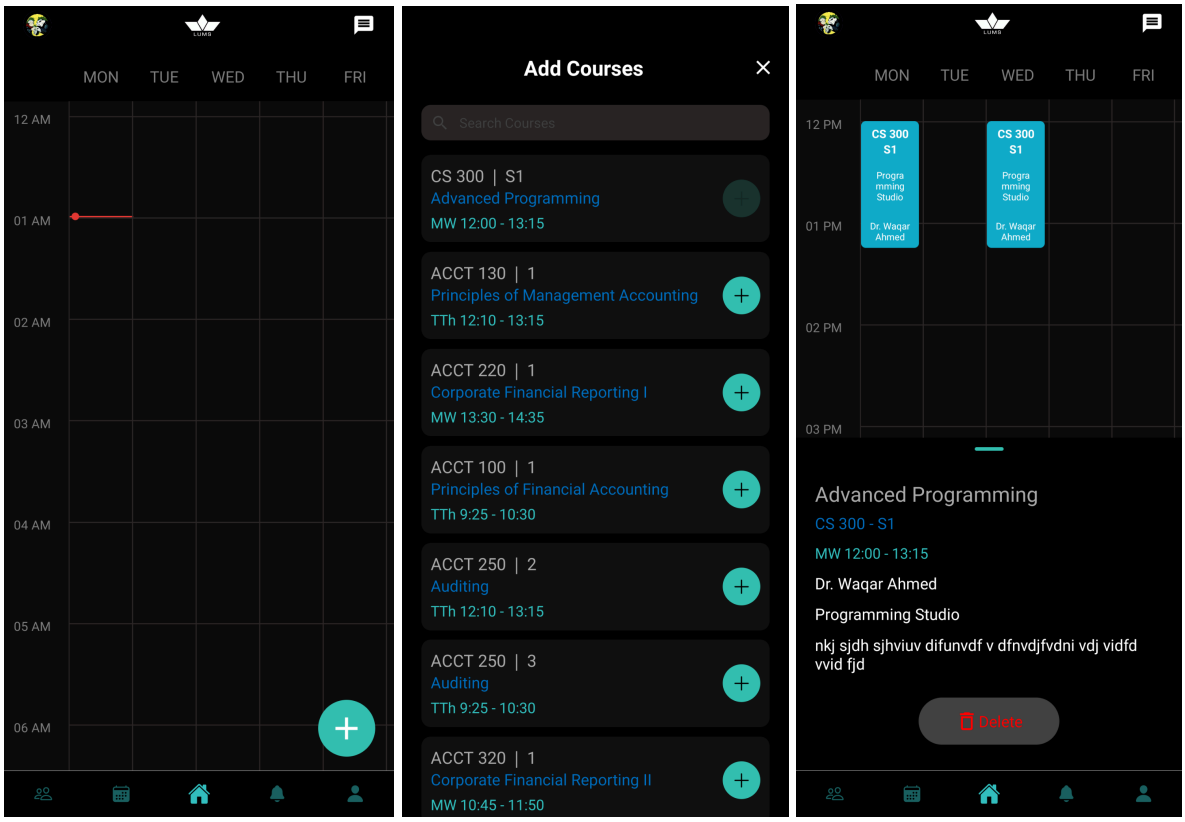
On launching the app, the user is presented the Signup/Login screens and prompted for credentials.

Homepage:



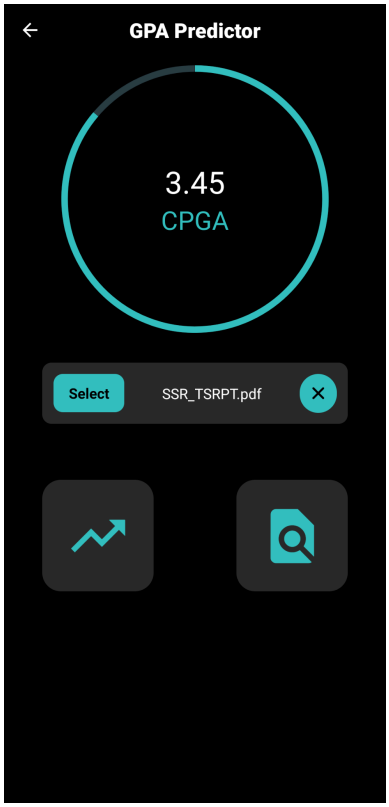
The homepage includes all 5 tabs of the application at the bottom, as well as 4 functionalities on the main page. These include Scheduler, GPA/Transcript and Donations.

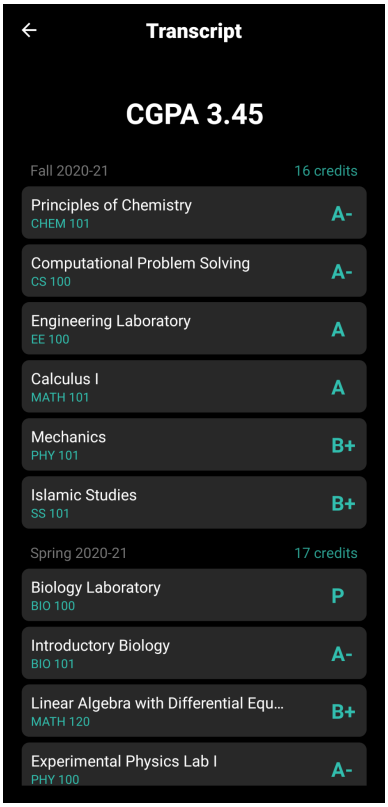
Scheduler:



On navigating to the scheduler, the user is able to view, add, and delete courses as shown.

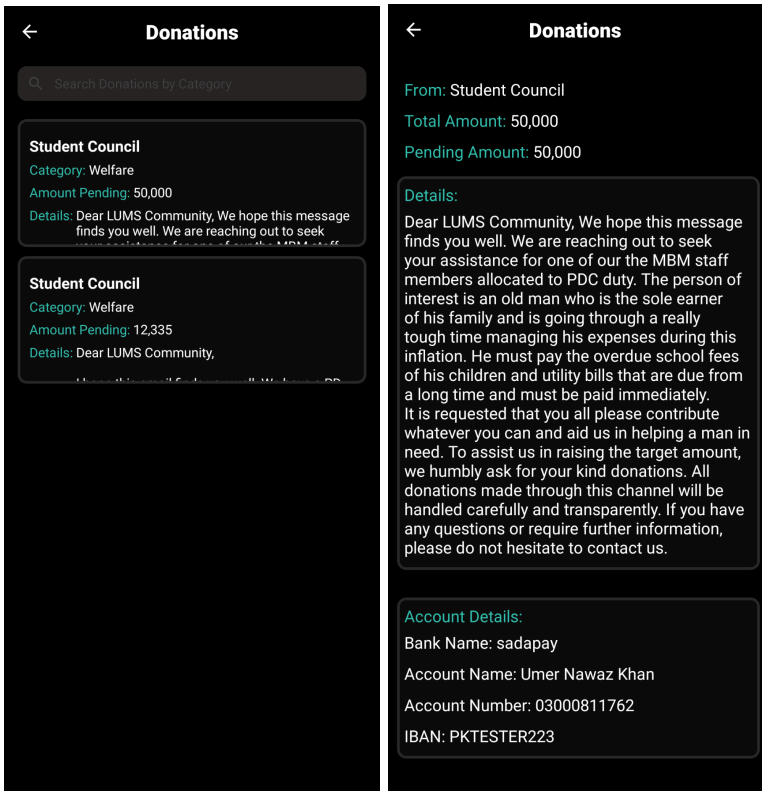
GPA/Transcript:





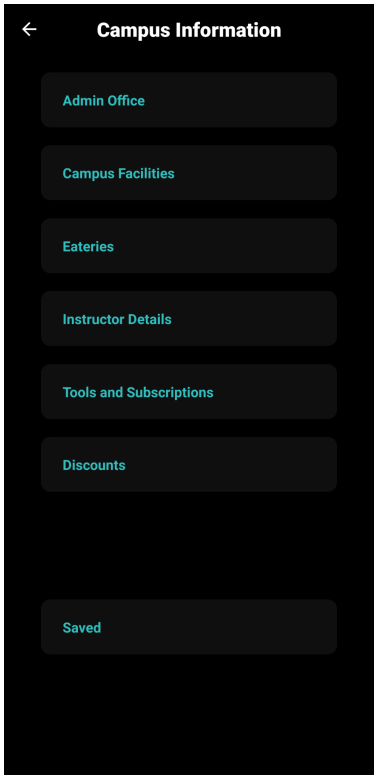
On navigating to the transcript page, the user is able to upload a transcript, which is automatically parsed by the server, and view their academic performance at a glance.

Donations:



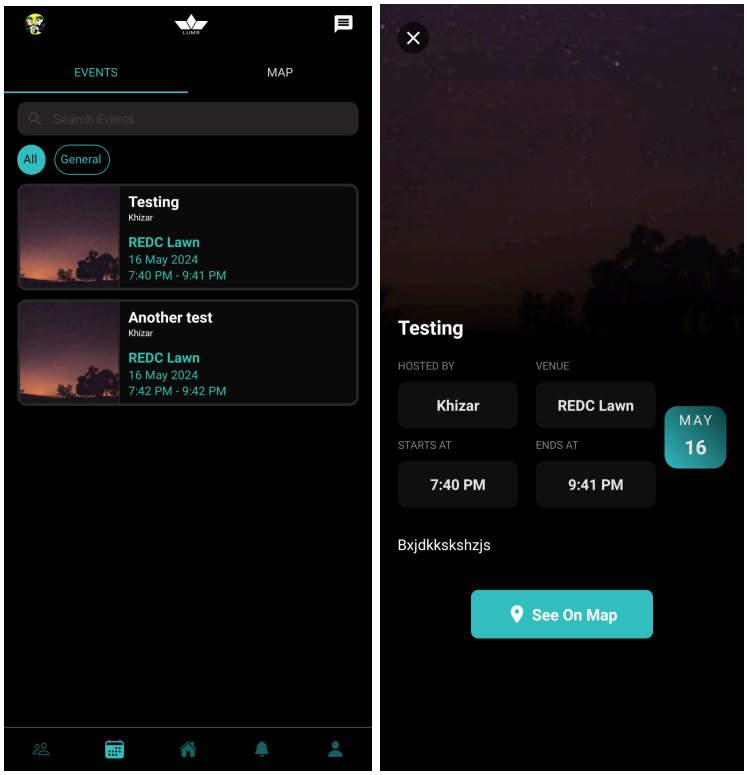
On navigating to the donations page, the user is shown all currently ongoing donation campaigns from the student council.

Campus Information:



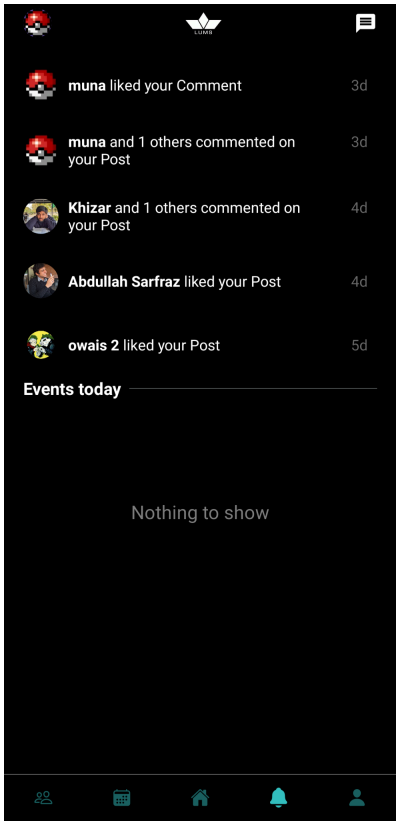
On navigating to the Campus Information screen, the user is able to explore a wide array of campus information as they please.

Events:



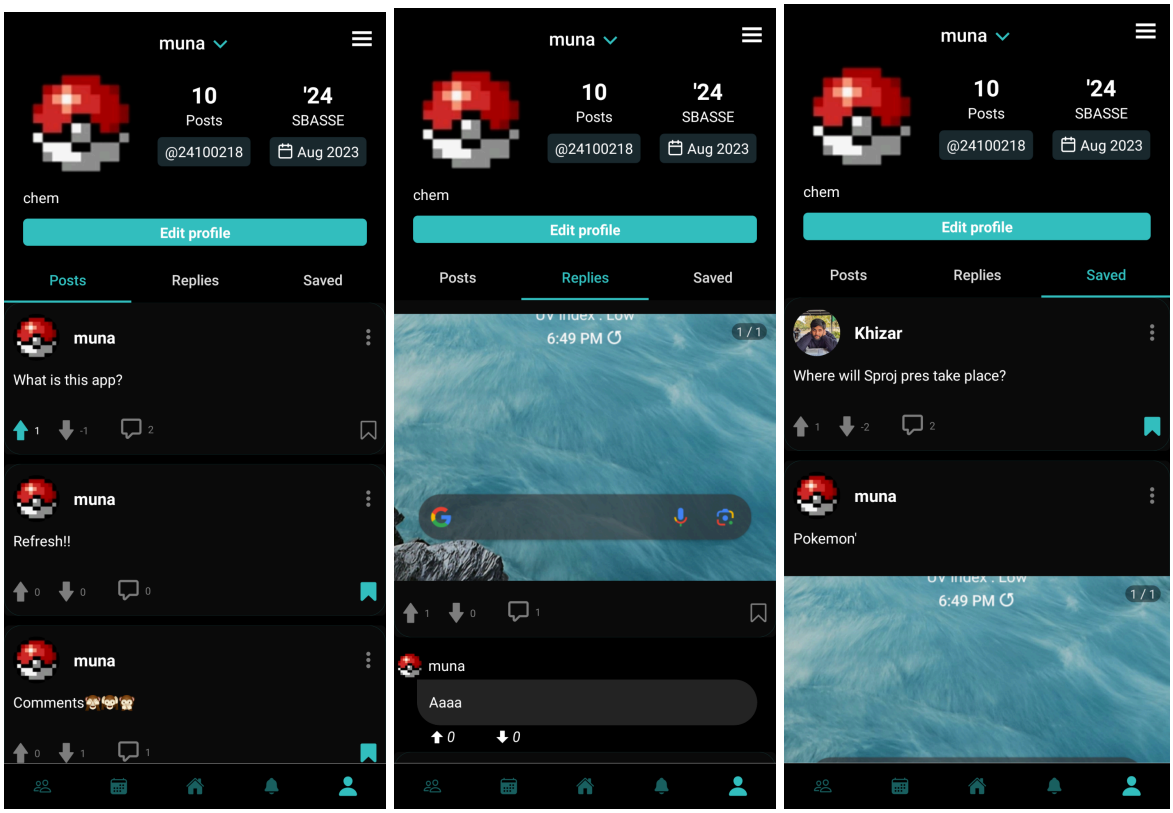
On navigating to the Events tab, the user is able to view a list of events, filter by tags, and expand an event to view details.

Notifications:



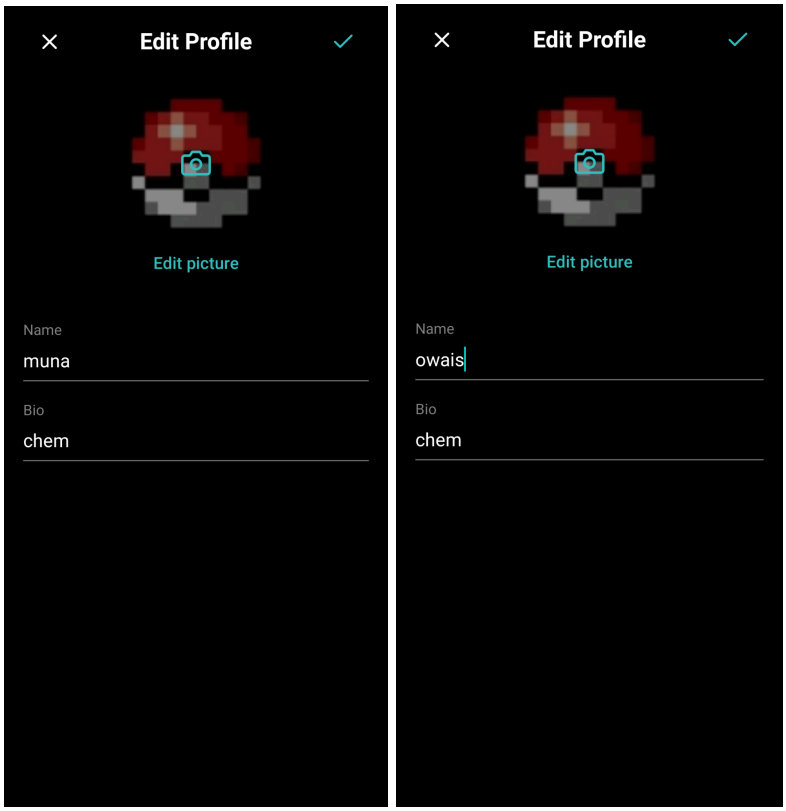
On navigating to the notifications tab, the user is shown their recent notifications, as well as Events scheduled for the current day.

Profile Tab:



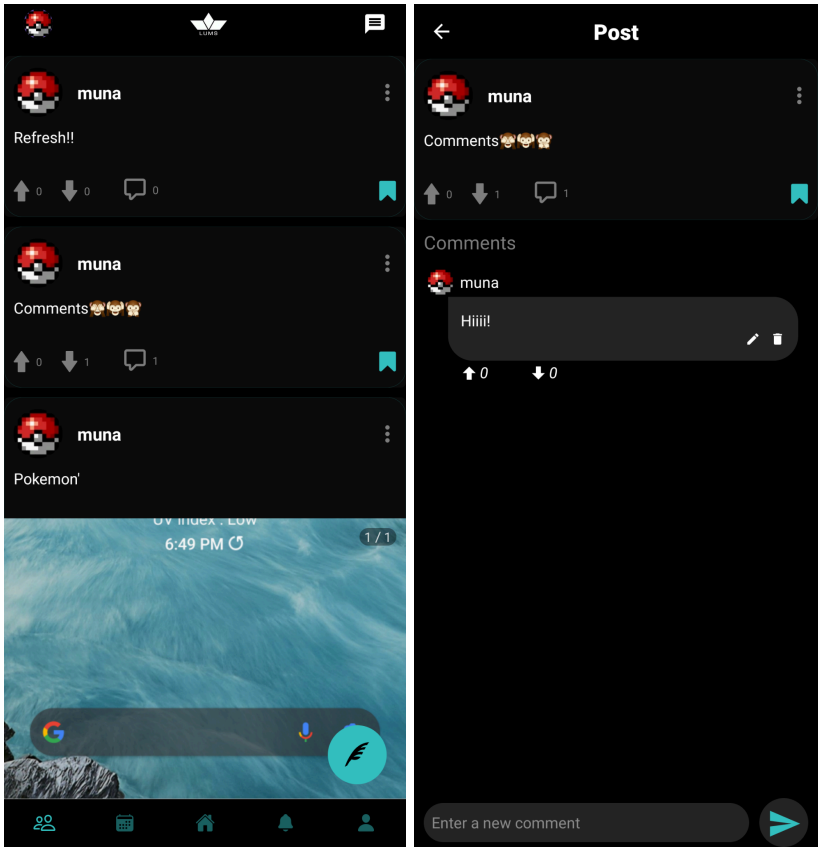
On navigating to the Profile tab, the user is able to view their posts, replies, and bookmarks with ease.

Edit Profile:



On navigating to the Edit Profile page from the Profile tab, the user is able to update their icon, username, and bio.

Forum:



On navigating to the LDF tab, the user is able to view recent LDF posts, as well as expand a single post and view, post, edit, and delete comments.

6. Project Security

Sr #	Security Risks	Potential Losses	Controls
1	Injection	• Unauthorized viewing or deletion of sensitive data. • Attacker may gain admin rights to database.	• Implement server-side input validation. • Ensure parametrized queries with input sanitization.
2	Identification and Authentication Failures	• Attackers may gain access to user accounts via credential stuffing, common passwords or weak password recovery/reset processes. • Brute force attacks are possible in case of weak session management.	• Encourage users to use strong passwords and check against common weak passwords. • Implement multi-factor authentication to prevent automated credential stuffing and brute force. • Use a server-side session manager that generates session IDs with high entropy at every login.
3	Security Logging and Monitoring Failures	• Failure in detecting breaches. • Logs may be visible to an attacker, causing information leakage. • Locally stored logs may be completely lost in the event of a server failure.	• Ensure all login or access-control failures are logged in a consumable format with sufficient context. • Implement effective log-monitoring and alerts in case of suspicious activity.

			• Maintain at least one off-server copy of the logs. • Ensure users have at most minimal access to logs.

Scanning Tools

Sr#	Security Tool Name	Brief description (Why the tool is suitable for your project)
1	SonarQube	SonarQube supports a wide range of languages, including TypeScript, HTML/CSS, and JavaScript, which will be used in our project.
2	OWASP ZAP (Zed Attack Proxy)	It is an open-source dynamic application security testing tool. It helps identify vulnerabilities in web applications by actively scanning and probing for security issues. It also works with the languages and technologies used in our project.

7. Risk Management

Potential Risks and Mitigation Strategies

Sr.	Risk Description	Mitigation Strategy
1.	Staff Illness	Reorganize the team so that there is more overlap of work and people therefore understand each other's jobs.
2.	Database performance does not meet requirements	Investigate the possibility of buying a higher-performance database.

3.	The time required to develop the software is underestimated.	Historical data can provide insights into how long similar projects took. We will also incorporate a contingency in the project schedule to accommodate potential delays.
4.	Software development tools cannot be integrated	Conduct a thorough evaluation of tools before selection to ensure they are compatible.
5.	Changes in requirements that require major design rework are proposed.	Maintain a flexible and modular design that can accommodate changes with minimal disruption.
6.	Communication Breakdown	Establish clear communication channels and protocols for the project. Hold regular status meetings to keep all stakeholders informed and aligned.
7.	Technology Failures	Implement redundancy and backup systems for critical technologies.
8.	Quality Assurance Failures.	Maintain a focus on quality throughout the project lifecycle, not just during the final stages.
9.	Legal and Regulatory Compliance.	Conduct thorough legal and regulatory assessments at the project's outset. Document and track legal and regulatory compliance activities and findings.
10.	Lack of Faculty Guidance.	Maintain regular and scheduled meetings with faculty advisors to review project progress. Develop contingency plans for faculty unavailability.

8. Testing and Evaluation

Testing Strategy

Functional Testing:

Positive Test Cases: Validate that the application behaves as expected when provided with valid input.

Negative Test Cases: Ensure the application handles invalid inputs gracefully and securely by providing error messages or other forms of feedback.

Boundary Value Testing: Test the limits of input fields and interactions to ensure the system can handle edge cases without failure.

Integration Testing:

Verify that different parts of the application work together as expected, focusing on data flow and dependency interactions between modules.

Performance Testing:

Assess the system's performance under varying loads to ensure it meets the required specifications and user expectations.

Security Testing:

Evaluate the application for vulnerabilities to protect sensitive data and ensure compliance with security standards.

Usability Testing:

Ensure the application is intuitive, easy to use, and accessible to all intended users.

Sample Test Cases

Use Case 1: Create Account

Positive Test Case:

Test Case ID: UC1-01

Objective: Verify that a user can create a new account using valid details.

Steps: Navigate to the sign-up page, enter valid user details (name, email, password), and submit.

Expected Result: Account is created successfully; user receives a confirmation email.

Negative Test Case:

Test Case ID: UC1-02

Objective: Ensure the system rejects duplicate email addresses.

Steps: Attempt to create a new account using an email address already in use.

Expected Result: Error message displayed stating that the email is already in use.

Use Case 2: Update Profile

Positive Test Case:

Test Case ID: UC2-01

Objective: Verify that a user can update their profile information.

Steps: Log in, navigate to the profile update section, change the user's name, and submit changes.

Expected Result: Profile is updated successfully; changes are reflected in user details.

Boundary Test Case:

Test Case ID: UC2-02

Objective: Check the maximum length allowed for the user's bio.

Steps: Attempt to update the bio with the maximum character limit specified.

Expected Result: Update is successful and no truncation of bio text.

Use Case 3: Search Functionality

Positive Test Case:

Test Case ID: UC3-01

Objective: Ensure the search functionality returns accurate results.

Steps: Enter a known keyword in the search bar and execute the search.

Expected Result: Search results correctly display items related to the keyword.

Use Case 4: Delete Account

Negative Test Case:

Test Case ID: UC4-01

Objective: Confirm that users cannot delete an account without re-entering their password.

Steps: Navigate to the account deletion page, attempt to delete without re-entering password.

Expected Result: Error message is displayed requiring password confirmation.

Use Case 5: Send Messages

Positive Test Case:

Test Case ID: UC5-01

Objective: Verify the messaging feature works between two users.

Steps: User A sends a message to User B; User B checks for the new message.

Expected Result: User B receives and can view the message from User A.

Use Case 6: Manage Notifications

Positive Test Case:

Test Case ID: UC6-01

Objective: Ensure that users can modify their notification settings.

Steps: User changes their notification preferences in settings.

Expected Result: Notification settings are updated as per the user's new preferences.

Tools for Automation

These test cases can be automated using the tools mentioned:

JUnit/TestNG: For backend logic tests such as validations in account creation and deletion.

Postman: For testing APIs involved in messaging and notifications.

9. Deployment Guidelines

List down the steps for deployment of your system. Start from where the code (link of the github repository) should be picked and then mention all the steps for deployment in a production environment. Also mention the online link where your application is hosted along with access information (user/password etc.).

Steps for Deployment:

- The source code for the frontend is stored at GitHub here.
    - Testing:
        - Open directory
        - Run “npm install” command
        - Run “npm start” command
        - Scan QR code
- The source code for the backend is stored at GitHub here.
    - Testing:
        - Open directory
        - Run “npm install” command
        - Run “npm run dev” command
- The backend is deployed on **Cyclic** at this URL.
- The frontend consists of an APK created using Expo Go managed workflow and stored here.
- In order to use the app, the APK needs to be installed on an Android mobile device.
- Credentials to login to the app are as follows:
    - Email: 24100225@lums.edu.pk
    - Password: 12345678

10. Conclusion

a. Summary

We approached our senior project with a structured and organized approach. Knowing the complexities that would lie ahead, we had multiple meetings to discuss the structure and implementation of our concepts before coding them. This led to a well-structured and steady workflow where everyone knew their task and how they would need to collaborate with other members to get their desired results. Throughout our project, we learned the importance of planning and the benefit it gives later on in the project. Moreover, it is almost as important to remember that even though we think our planning is foolproof, it never is and therefore we must

to be vigilant in solving any problems that arise while working. These can range from dependency issues to logical flaws in our design. Being able to adapt in these situations can greatly help fast track progress while working on a large complex project.

b. Challenges

Technical challenges range from dependency issues, such as the inability of some modules to work in tandem, forcing us to change our preplanned model. Other problems can be due to the depreciation of old tools and libraries and the lower ability to produce new products, severely hindering workflow. Another problem was financial constraints that lead to technical challenges as integrating free software it required much more overhead as compared to paid software and services as those that were free also had little to no support or documentation to help us to mold them to our specifications.

Non-technical challenges included managing different courses and additional co-curricular activities while working in a group. Moreover, trying to get everyone together required inefficient planning and could be better streamlined. Additionally, focusing on the documentation left us with less time to code than anticipated, which cost us while in our coding phase.

While all this being said, we adapted to these challenges by being open to change and new ideas. With this approach, we were able to tackle any and all challenges faced and whenever we were given a deadline, we banded together to solve any problem and were there for each other. This team work and can do attitude helped us in creating a team which was able to complete our senior project satisfactorily.

c. Future

Currently, we have decided to keep working on this on the side as it is a project close to our hearts. We believe we can add many functionalities that we missed before or new ones that we have not considered. Moreover, as our application can be generalized, we can extend this application to other universities and colleges that can benefit from these key features. Finally, working with OSA, we can integrate official documentation and procedures into our application to make this the official LUMS application, becoming the main point of contact for every LUMS student.

11. Review checklist

Before submission of this report, the team must perform an internal review. Each team member will review one or more sections of the deliverable.

Chapter/Section Name	**Reviewer Name(s)**
Intro, Risks	Muhammad Abdullah Sarfraz
System requirements, Conclusion	Nauman Ijaz
Software Development Methodology and Plan, System Architecture	Owais Ahsan

12. References

1. Expo Documentation: https://docs.expo.dev/
2. React Native: https://reactnative.dev/docs/getting-started
3. MongoDB: https://www.mongodb.com/docs/
4. Cloudinary: https://cloudinary.com/documentation
5. Firebase: https://firebase.google.com/docs
6. SendGrid: https://docs.sendgrid.com/
7. Postman: https://learning.postman.com/docs/introduction/overview/
8. GetStream: https://getstream.io/chat/docs/sdk/reactnative/