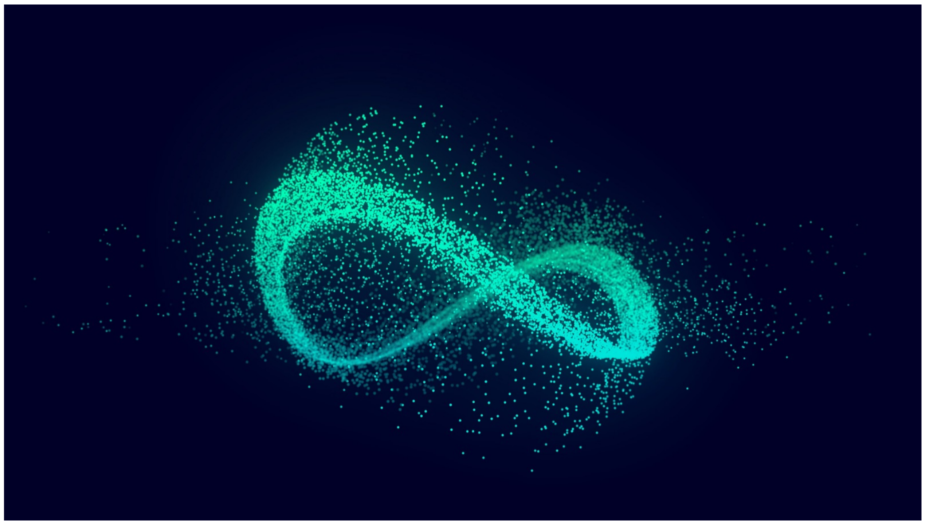
T44D3

2025-05-02

Deliverable Three



our team

Team Name: The Infinites

Team Number: 44

Mentor Name: Dr Carl Van Der Westhuizen

Team Members

1. Nation Dibakwane-222122307
2. Tsediso Mokwena-220068980
3. Karabo Ramanamane-221113164
4. Lehlogonolo Moseke-219067401

Table of Contents

Introduction

Software engineering focuses on identifying real-world problems and leveraging tools, techniques, and methodologies to develop effective solutions. As **The Infinites**, our project aims to build a **Student Hub**—a centralized platform where UJ students can discover and engage with societies, day residences, sports teams, and university events efficiently.

Background

The University of Johannesburg (UJ), founded in 2005, fosters innovation, diversity, and student development. With over 50,000 students—including 3,000+ international students from 80 countries—UJ spans four campuses and offers a variety of extracurricular activities:

- **Day Houses**: Student-led communities for social and academic engagement.
- **Societies & Clubs**: Groups centered on shared interests (technology, arts, sports).
- **Sports Teams**: Competitive and recreational teams requiring structured management.
- **UJ Events**: University-wide activities (networking sessions, guest lectures). Despite these offerings, there is no unified platform for students to discover and join these communities, leading to low engagement and missed opportunities.

Current Systems & Challenges

Recruitment Methods

- In-person outreach and referrals.
- Informal Day House challenges (e.g., loser invites others).
- Google Forms for membership data, manually transferred to spreadsheets.

Membership & Event Management

- Society events require manual proposals for UJ administrator approval.
- Attendance tracked manually via messaging apps.
- No structured budgeting or engagement analytics.

These disconnected, manual processes hinder both student discovery and community administration.

Problem Statement

UJ's fragmented student engagement systems create two major issues:
1. **Discovery Barrier**: Students struggle to find and join campus communities. 2. **Administrative Inefficiency**: Community admins face challenges in membership management, event coordination, and participation tracking.

Overall, these issues reduce student engagement and complicate administrative tasks.

Solution

UJ Student Hub: A unified digital platform that streamlines student engagement.

- **For Students**:
  - Discover and join communities and events in one place.
  - Receive tailored news, announcements, and event alerts.
- **For Admins**:
  - Manage community profiles, memberships, and events with integrated tools.
  - Submit and track budget requests and attendance analytics.

Requirements

Functional Requirements

1. **User Management & Authentication**
    - Registration via email/UJ credentials
    - User Must be able to login Securely Using their email and passowrds
    - A user must be able to seamlesly edit user profiles
2. **Community Management**
    - Community managers should be able to create and/or update community profiles
    - Students should be able to enrol and be part of a community
    - Students should have the option to leave a community providing a reason to do so
    - A Search/filter function should enable students to search/filter communities and events
3. **Event Management**
    - Community Admins should be able to submit an event proposals to the SDP for approval
    - Community admins should be able to book for a venue for an event if required
    - Community admins should be able to request event funds if required to SDP
    - Community admins should be able to request a transport if required
    - Students should be able to Confirm their attandance to an event
    - At an event, students should be able to add an attandance for an event using the event's geneated attandance code.
    - Students should be able to recieve notifications for event updates including upcoming events, cancellations, and reschedules

Non-Functional Requirements

- **Security & Privacy**: Students and users's data should be stored securely
- **Scalability**: Handlling a growing user base and traffic is neccessary.
- **Compatibility**: must support both Web and mobile.

Use Cases

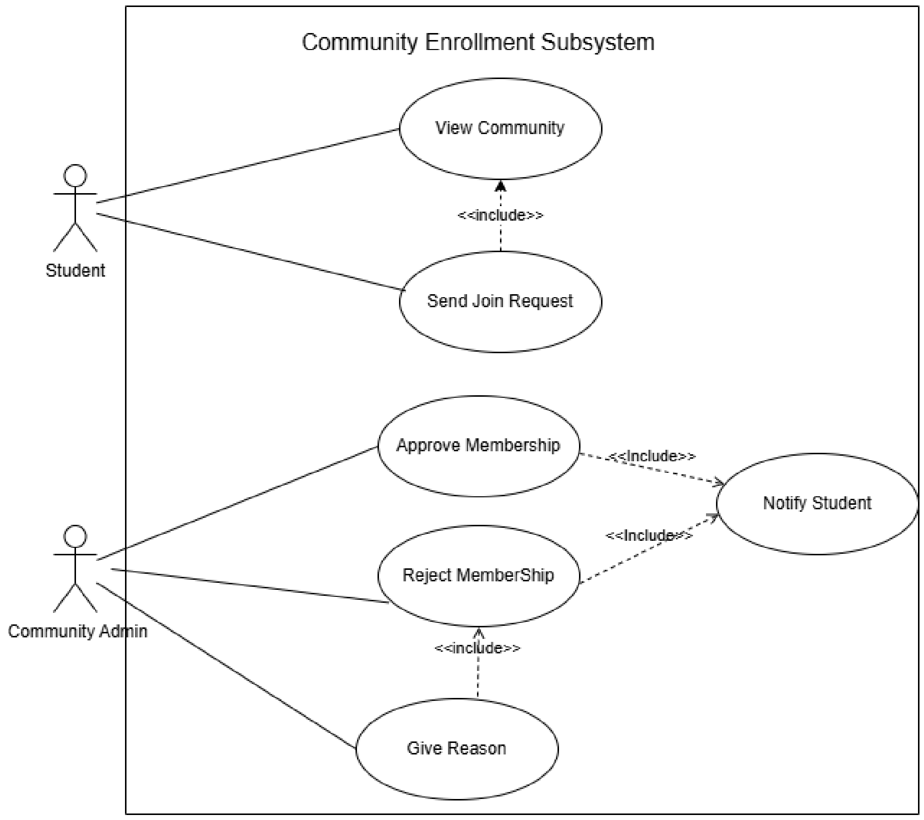


Figure 1: Community enrollment

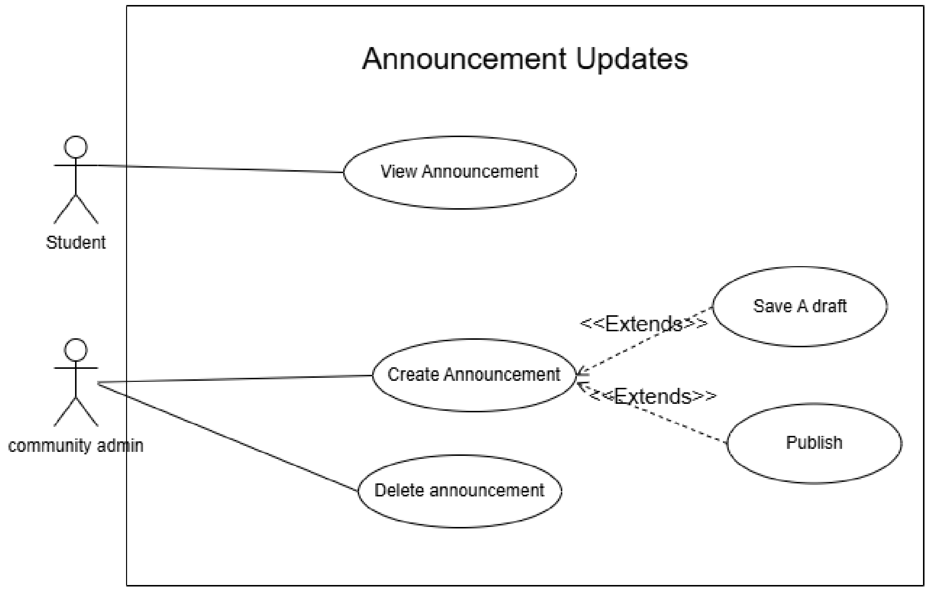


Figure 2: Announcement updates

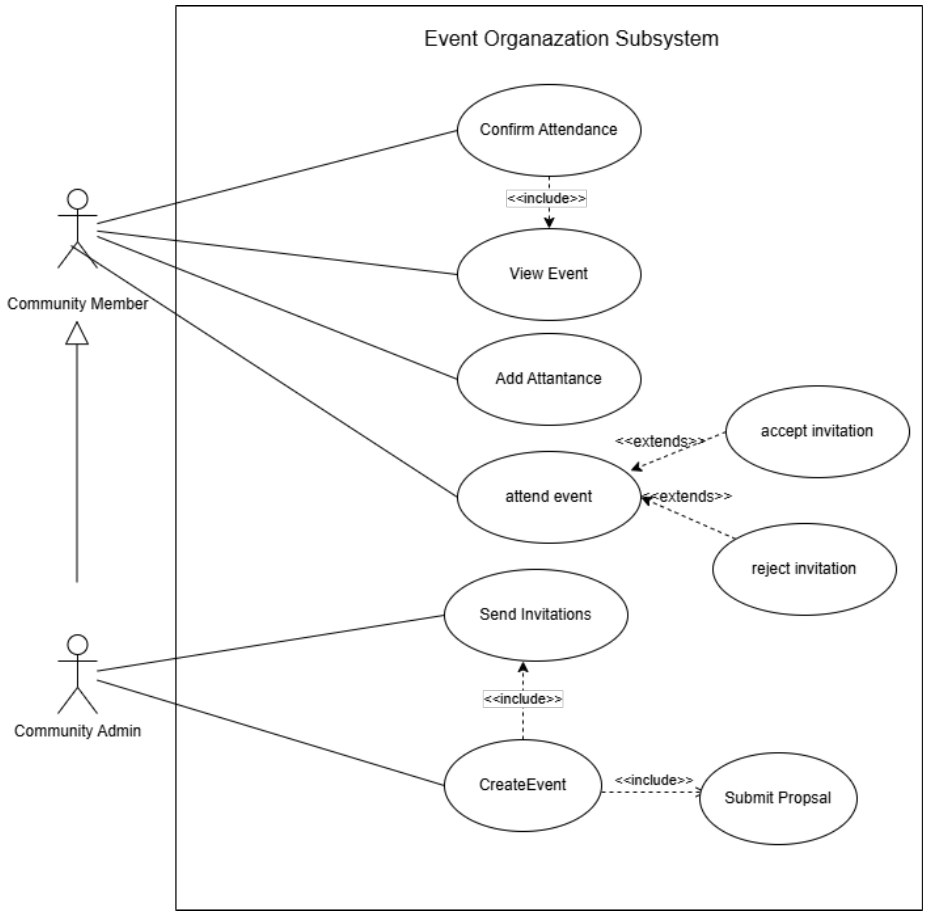


Figure 3: Event Organization

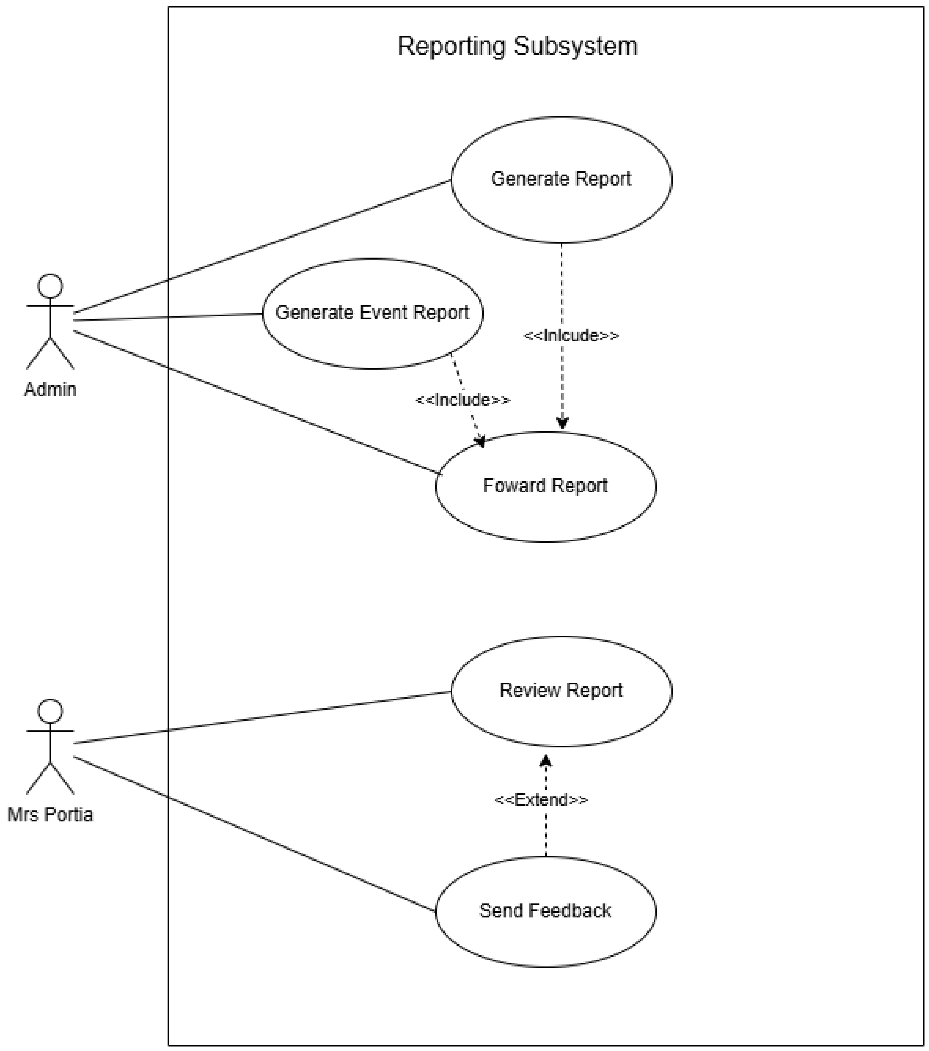


Figure 4: Reporting

Feasibility and Risk Study

System Feasibility

The Student Hub will integrate student engagement, sports, societies, day houses into a single platform. While UJ already has some existing systems that offer partial solutions, they are scattered across different platforms and lack interactivity. Below is a comparison with current UJ systems.

UJ uLink Portal

what it offers

1. Provides access to academic records, timetables, student information, and course content.

Limitations Compared to Our System

1. Does not support event discovery, community engagement, or sports team management.
2. Lacks RSVP functionality for event attendance confirmations.
3. does not engange with students except academically.
4. does not provide news updates.

How Our System Improves on It 1. Adds interactive student engagement features beyond academics. 2. Allows students to explore and join communities and explore different aspects of being a student.

Technical feasibility

Every software product needs to be evaluated and assessed whether it is appropriate to build. This feasibility study is done to evaluate whether the system is economically, technically, operationally, and financially sound. The student Hub Will require several hardware and software requirements, including Hosting solutions, Device compatibility, the compatible operating system(s), tools for building front end and back end, scalable database for handling growing traffic.

Hardware requirements

1. Devices to access the student hub.
    - Any compatible smart device to access the system with an internet connection including smartphones, tablets, computers and laptops
2. Hosting Solution
    - Hosting the student Hub via a physical server will be costly, especially for a Third-year project, since it will require building, configuration and maintenance.
    - Free Cloud based hosting solutions will be necessary. This must be able to handle a fair amount of traffic since the system will be used mainly by the university students only.

Software requirements

1. Compatible Operating systems
    - Smart device must be running either on windows, iOS or android
2. Front End Tools
    - The student hub is a simple student friendly system, so it won't require any advanced tools for building the front end. HTML, CSS, JavaScript, and bootstrap will do the work for the front end for the web application.
    - The mobile application will utilize tools such as react native for building an easy cross platform mobile application
3. Backend Tools
    - Since the system will storing data such as user profiles, community profiles, memberships, news, events, a reliable, scalable database will be required.
    - The system must also handle user authentication, so authentication tools will be necessary.
    - The system must also handle membership payments, thereby a payment gateway will be required
    - The system must also be able to send event reminders, even without an internet connection, thereby a service worker provider tool will be necessary

Economic Feasibility

The cost for developing the Student Hub will be minimal since the project will be developed using free, open-source software and tools. Running, updating, and maintaining the system will not cost any amount since these are primarily voluntary efforts by the group members. hosting and storage could have its costs which could range between R0 – R400 depending on which domain the system will be using. Since this is a Third year project, direct revenue generation is not the primary focus, the benefits offered by the project are academic success, improvement in software engineering skills and knowledge and improved personal portfolios. Overall, the system will cost between the ranges of R0 to R400, thereby, this project is economically feasible due to minimal costs.

Operational Feasibility

The aim of building the Student hub to address the needs of the problem and improve overall student engagement,sencondly, it is to adress address the lack of awareness and access of information and lastly it is to adress the inefficient membership management for communities. By building the student hub, as the infinites, we aim to provide a platform for students to discover and connect with communities and increase student engagement and participation in campus life. We would also be streamlining interactions between the community admins and the Student Development Practitioner as well as interactions between members and community admins.

Use Case Descriptions

UCD1: View Events

Actor: Student
Input: None
Output: List of upcoming events

Basic Flow:
1. Student clicks “Events”.
2. System retrieves events.
3. System displays event list.

Preconditions : User must be Logged in and on landing page

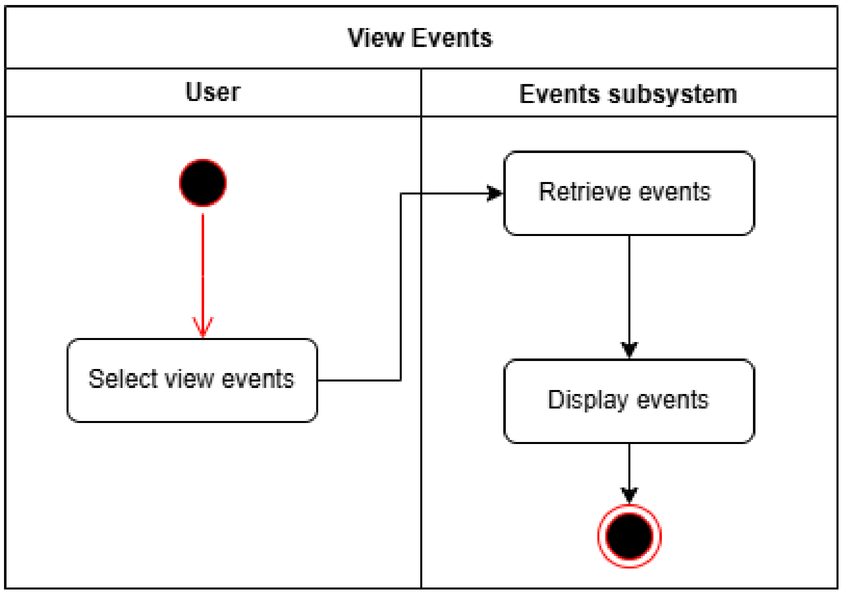


Figure 5: View Events

UCD2: Add Attendance

Actor: Student
Input: Attendance code
Output: Confirmation message

Basic Flow:
1. Student selects an event.
2. Student clicks "Attendance."
3. System shows code entry form.
4. Student enters code.
5. System validates code.
6. System records attendance.
7. System shows confirmation.

Extension – Invalid Code:
- System displays error; student retries.

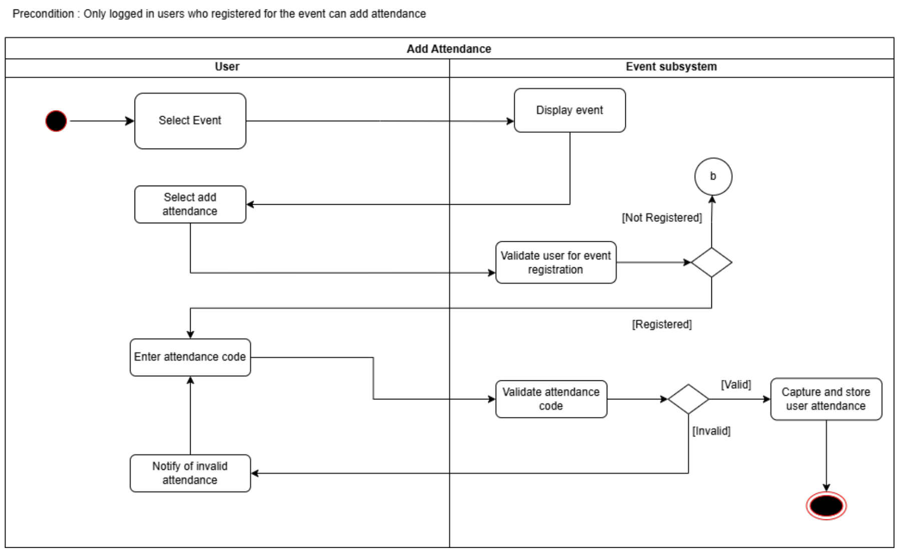

Precondition : Only logged in users who registered for the event can add attendance
Add Attendance
User
Event subsystem
Select Event
Display event
b
Select add attendance
[Not Registered]
Validate user for event registration
[Registered]
Enter attendance code
Validate attendance code
[Valid]
Capture and store user attendance
[Invalid]
Notify of invalid attendance

Figure 6: Add Attendance

UCD3: Create Event

Actor: Community Admin
Input: Event details
Output: Proposal status

Basic Flow:
1. Admin selects “Create Event.”
2. System displays proposal form.
3. Admin fills details and submits.
4. System forwards to SDP (Miss Portia).
5. SDP approves/rejects.
6. System notifies admin.

Extension – Invalid Details:
- System prompts to correct missing fields.

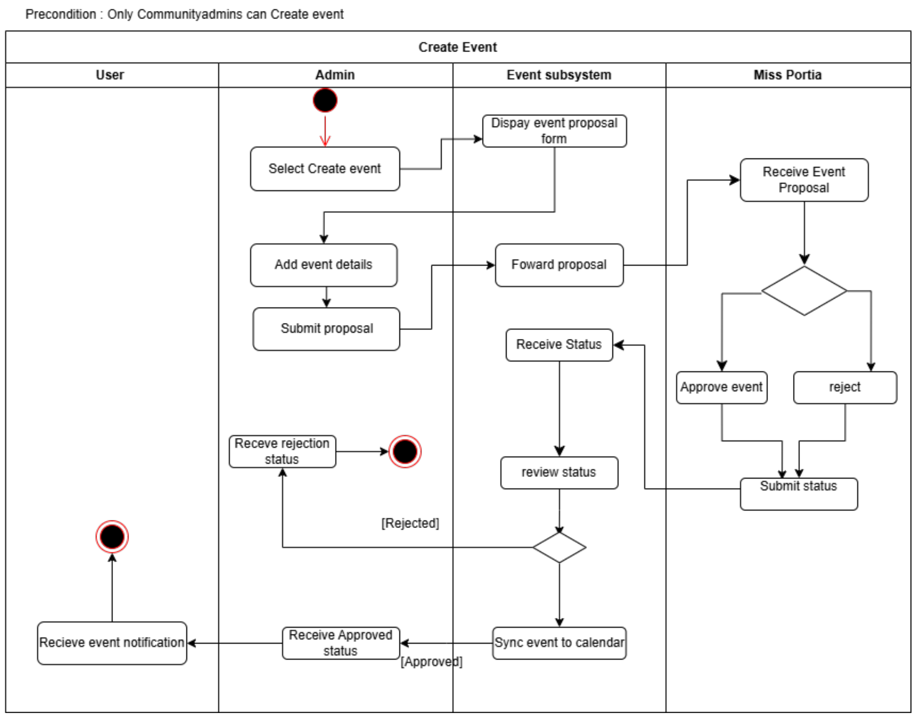


Figure 7: Create Event

UCD4: Cancel Event

Actor: Community Admin
Input: None
Output: Cancellation confirmation

Basic Flow:
1. Admin selects event.
2. Admin clicks "Cancel."
3. Admin confirms.
4. System removes event and updates calendar.
5. System notifies users and SDP.

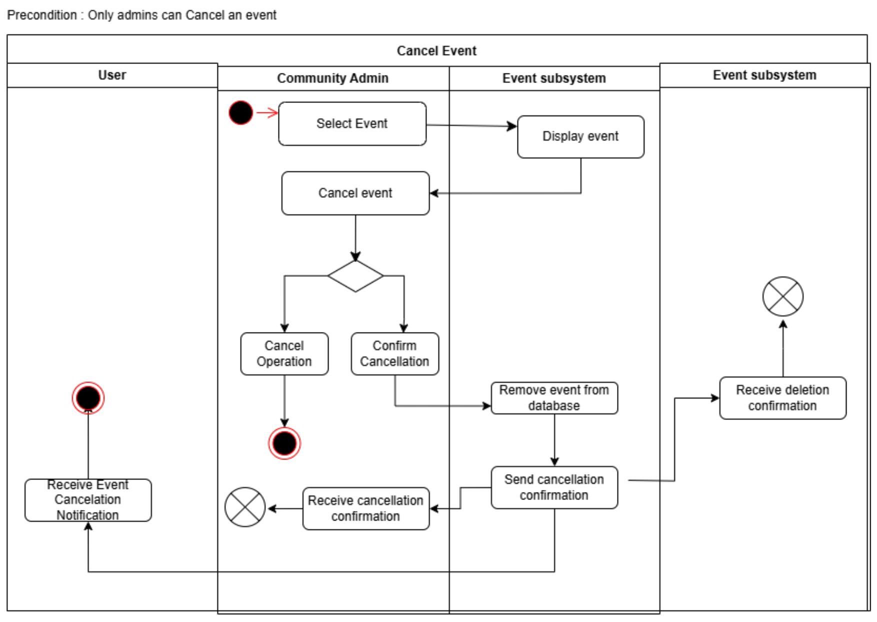

Precondition : Only admins can Cancel an event
Cancel Event
User
Community Admin
Event subsystem
Event subsystem
Select Event
Display event
Cancel event
Cancel Operation
Confirm Cancellation
Remove event from database
Receive deletion confirmation
Send cancellation confirmation
Receive cancellation confirmation
Receive Event Cancelation Notification

Figure 8: Cancel Event

UCD5: View Event Registrations

Actor: Community Admin
Input: None
Output: List of registrations

Basic Flow:
1. Admin selects event.
2. System retrieves registrations.
3. System displays list.

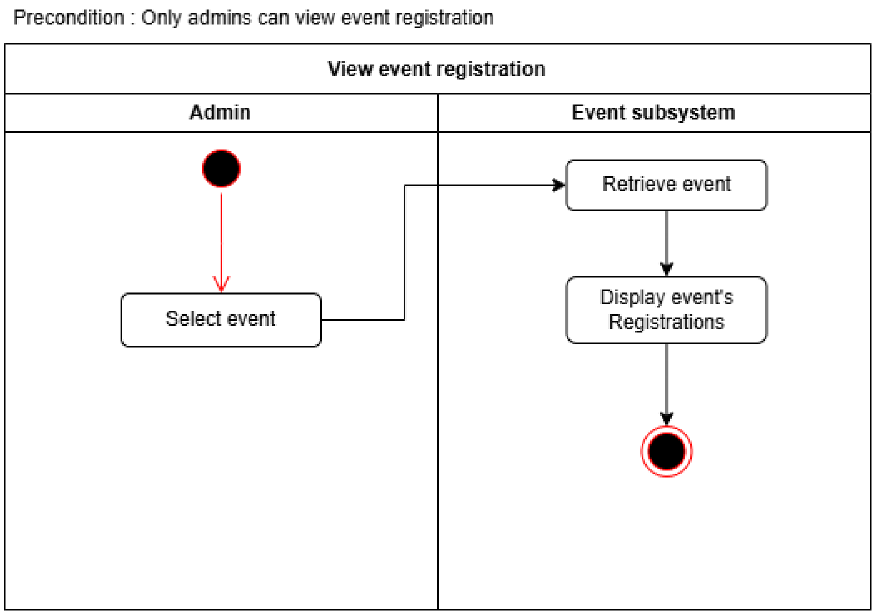

Precondition : Only admins can view event registration
View event registration
Admin
Event subsystem
Retrieve event
Select event
Display event's Registrations

Figure 9: View Registrations

UCD6: Join Community

Actor: Student
Input: Community selection
Output: Join confirmation

Basic Flow:
1. Student views community page.
2. Student clicks "Join."
3. System records membership request.
4. Admin approves.
5. System confirms to student.

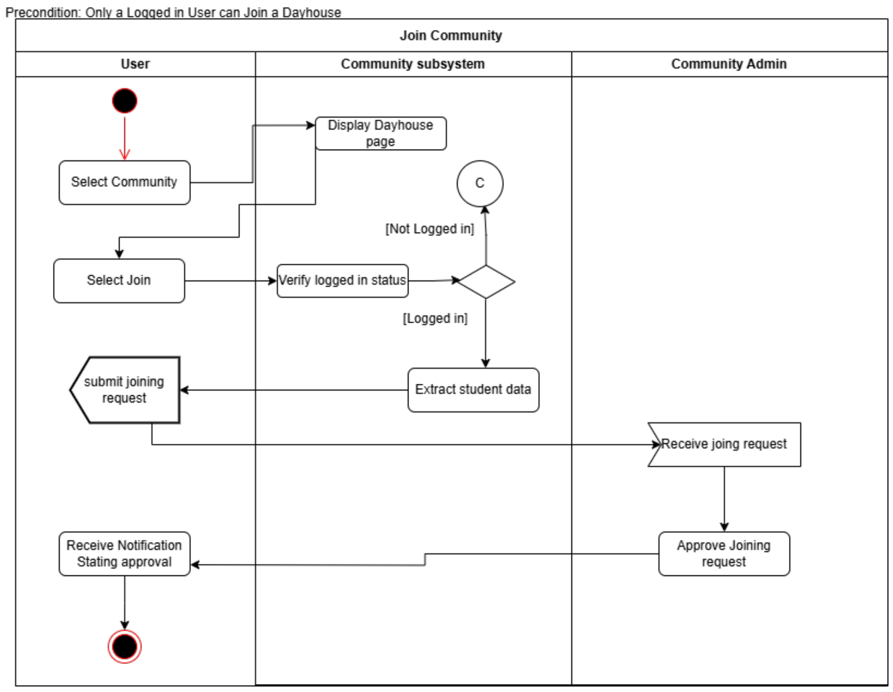


Figure 10: Join Community

UCD7: Generate Event Report

Actor: Community Admin
Input: Report details
Output: Submitted report

Basic Flow:
1. Admin selects "Generate Report."
2. System displays template.
3. Admin fills and submits.
4. System stores and forwards to SDP.

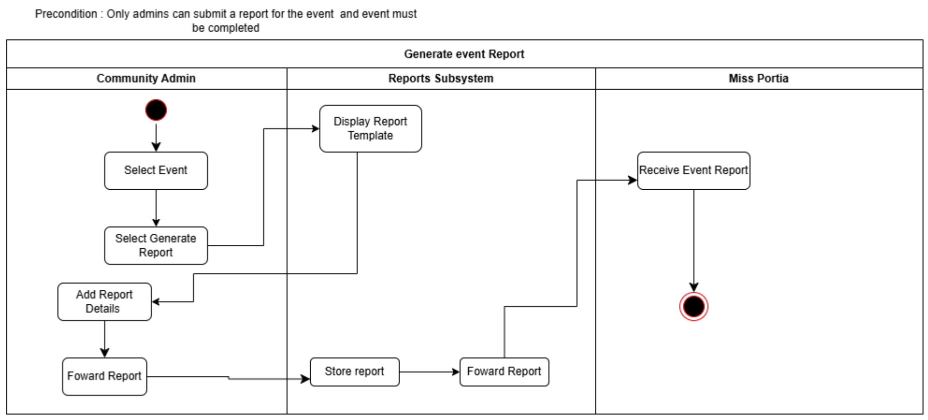


Figure 11: Generate Event Report

UCD8: Create Announcement

Actor: Community Admin
Input: Title, content, attachment
Output: Published announcement

Basic Flow:
1. Admin enters details.
2. Admin clicks “Publish.”
3. System saves and displays announcement.
4. System sends notifications.

Extensions:
- Missing details → prompt user.
- Notifications disabled → announcement only in feed.

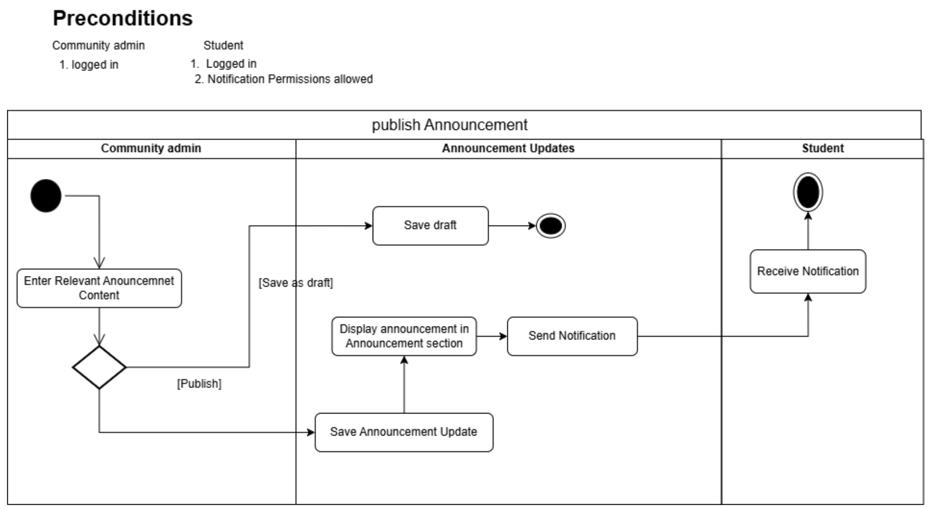


Figure 12: Create Announcement

Database Design

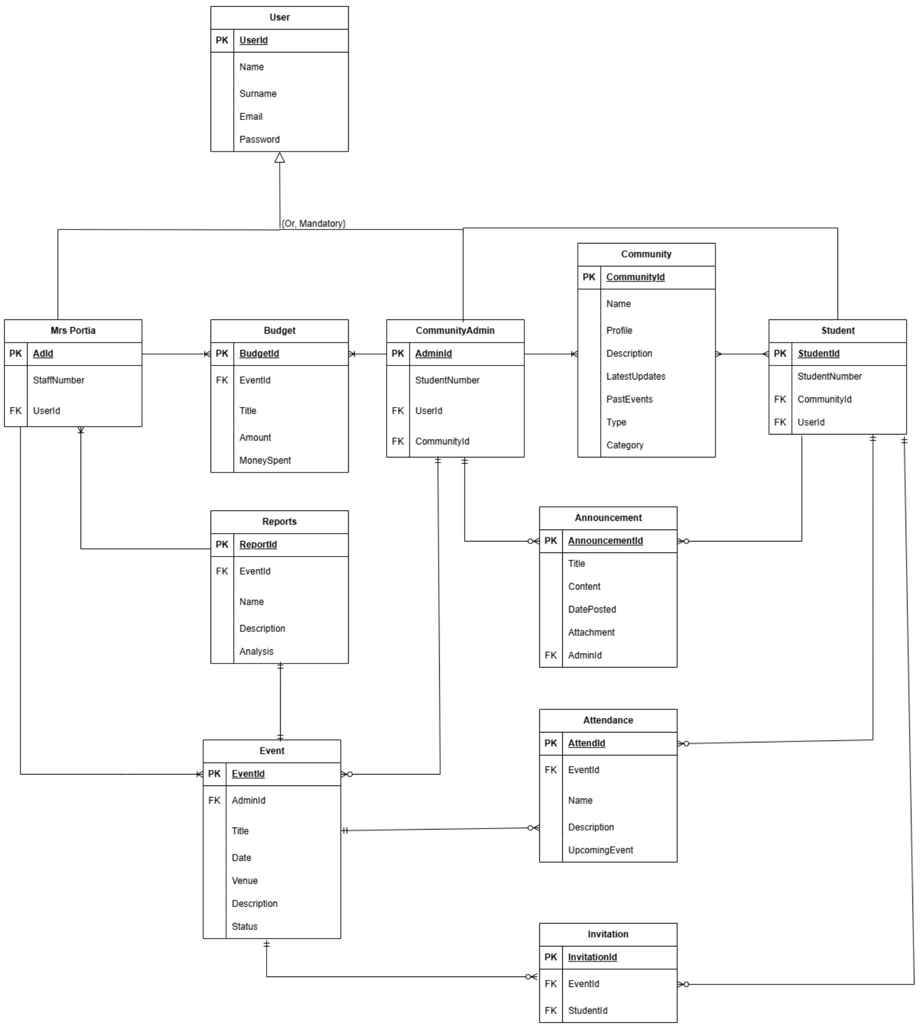


Figure 13: Database Design

Class Diagrams

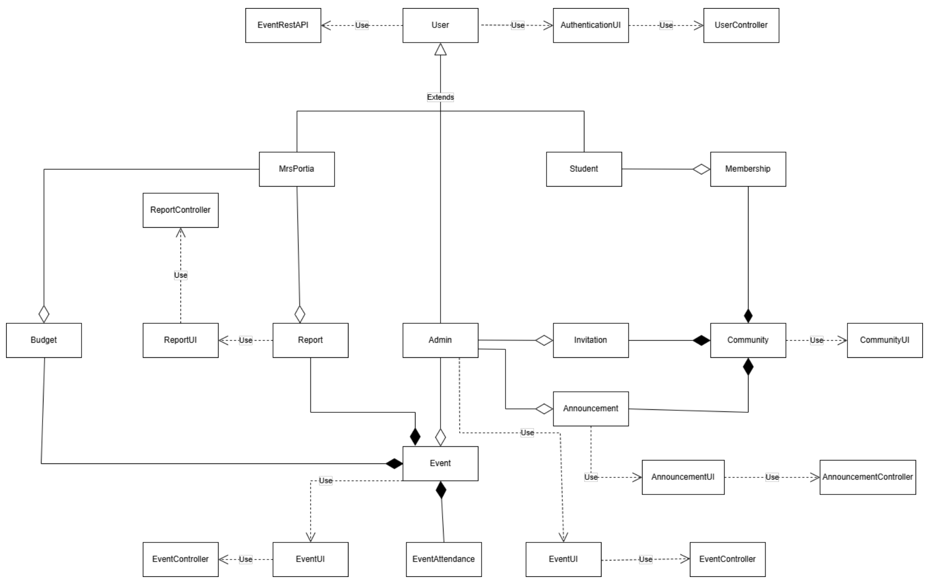


Figure 14: Class Diagram Relationships

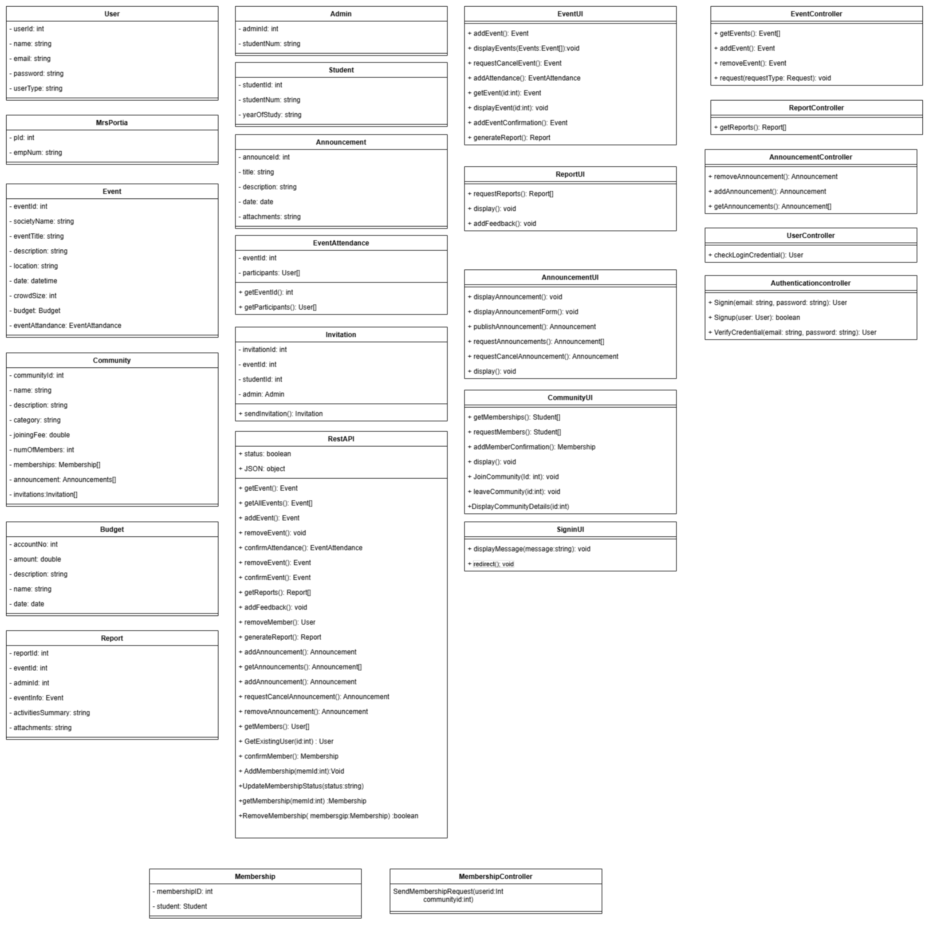

User
- userId: int
- name: string
- email: string
- password: string
- userType: string
MrsPortia
- pId: int
- empNum: string
Event
- eventId: int
- societyName: string
- eventTitle: string
- description: string
- location: string
- date: datetime
- crowdSize: int
- budget: Budget
- eventAttandance: EventAttandance
Community
- communityId: int
- name: string
- description: string
- category: string
- joiningFee: double
- numOfMembers: int
- memberships: Membership[]
- announcement: Announcements[]
- invitations:Invitation[]
Budget
- accountNo: int
- amount: double
- description: string
- name: string
- date: date
Report
- reportId: int
- eventId: int
- adminId: int
- eventInfo: Event
- activitiesSummary: string
- attachments: string
Admin
- adminId: int
- studentNum: string
Student
- studentId: int
- studentNum: string
- yearOfStudy: string
Announcement
- announceId: int
- title: string
- description: string
- date: date
- attachments: string
EventAttendance
- eventId: int
- participants: User[]
+ getEventId(): int
+ getParticipants(): User[]
Invitation
- invitationId: int
- eventId: int
- studentId: int
- admin: Admin
+ sendInvitation(): Invitation
RestAPI
+ status: boolean
+ JSON: object
+ getEvent(): Event
+ getAllEvents(): Event[]
+ addEvent(): Event
+ removeEvent(): void
+ confirmAttendance(): EventAttendance
+ removeEvent(): Event
+ confirmEvent(): Event
+ getReports(): Report[]
+ addFeedback(): void
+ removeMember(): User
+ generateReport(): Report
+ addAnnouncement(): Announcement
+ getAnnouncements(): Announcement[]
+ addAnnouncement(): Announcement
+ requestCancelAnnouncement(): Announcement
+ removeAnnouncement(): Announcement
+ getMembers(): User[]
+ GetExistingUser(id:int) : User
+ confirmMember(): Membership
+ AddMembership(memId:int):Void
+UpdateMembershipStatus(status:string)
+getMembership(memId:int) :Membership
+RemoveMembership(membersgip:Membership) :boolean
EventUI
+ addEvent(): Event
+ displayEvents(Events:Event[]):void
+ requestCancelEvent(): Event
+ addAttendance(): EventAttendance
+ getEvent(id:int): Event
+ displayEvent(id:int): void
+ addEventConfirmation(): Event
+ generateReport(): Report
ReportUI
+ requestReports(): Report[]
+ display(): void
+ addFeedback(): void
AnnouncementUI
+ displayAnnouncement(): void
+ displayAnnouncementForm(): void
+ publishAnnouncement(): Announcement
+ requestAnnouncements(): Announcement[]
+ requestCancelAnnouncement(): Announcement
+ display(): void
CommunityUI
+ getMemberships(): Student[]
+ requestMembers(): Student[]
+ addMemberConfirmation(): Membership
+ display(): void
+ JoinCommunity(Id: int): void
+ leaveCommunity(id:int): void
+DisplayCommunityDetails(id:int)
SigninUI
+ displayMessage(message:string): void
+ redirect(); void
EventController
+ getEvents(): Event[]
+ addEvent(): Event
+ removeEvent(): Event
+ request(requestType: Request): void
ReportController
+ getReports(): Report[]
AnnouncementController
+ removeAnnouncement(): Announcement
+ addAnnouncement(): Announcement
+ getAnnouncements(): Announcement[]
UserController
+ checkLoginCredential(): User
Authenticationcontroller
+ Signin(email: string, password: string): User
+ Signup(user: User): boolean
+ VerifyCredential(email: string, password: string): User
Membership
- membershipID: int
- student: Student
MembershipController
SendMembershipRequest(userid:Int
communityid:int)

Figure 15: Class Diagram

Sequence Diagrams

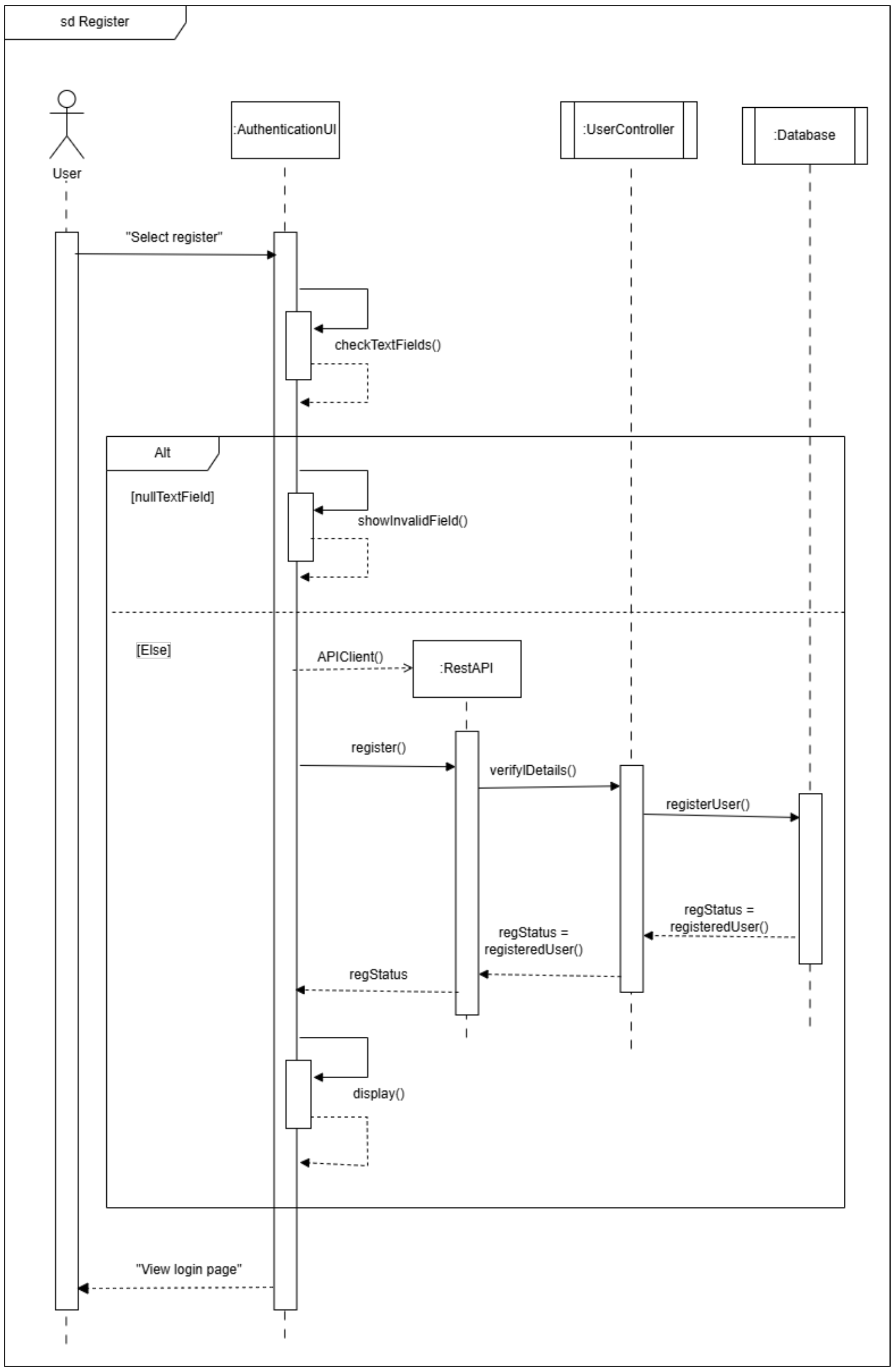
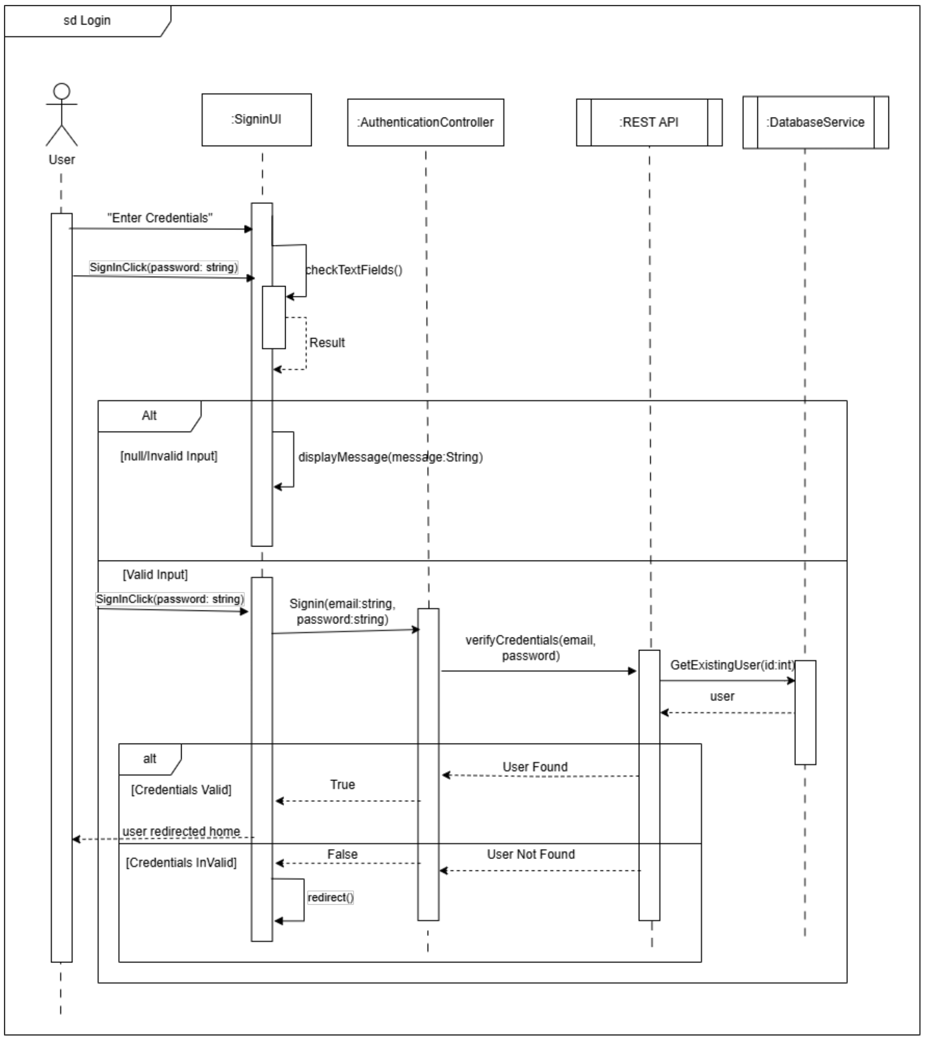
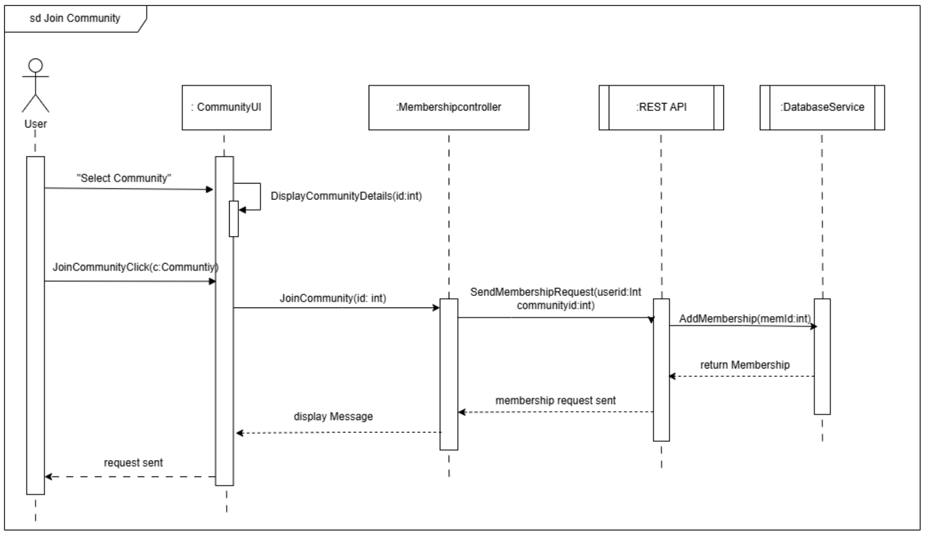
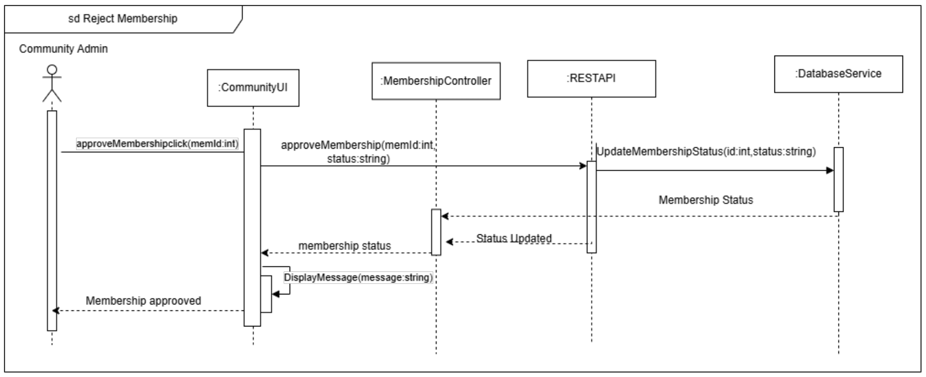
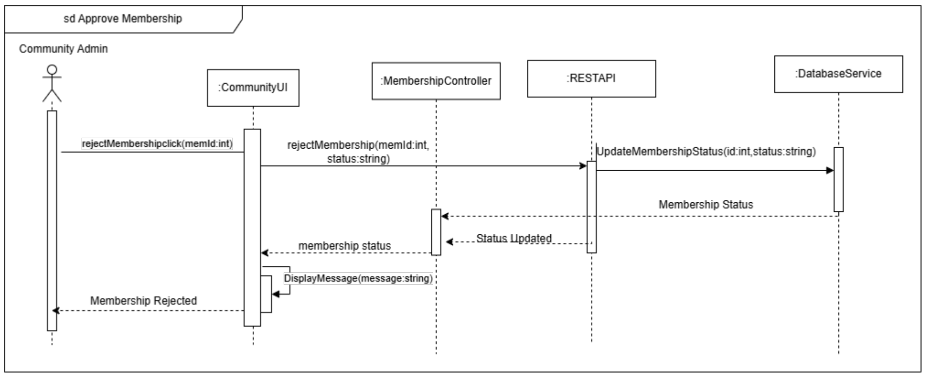


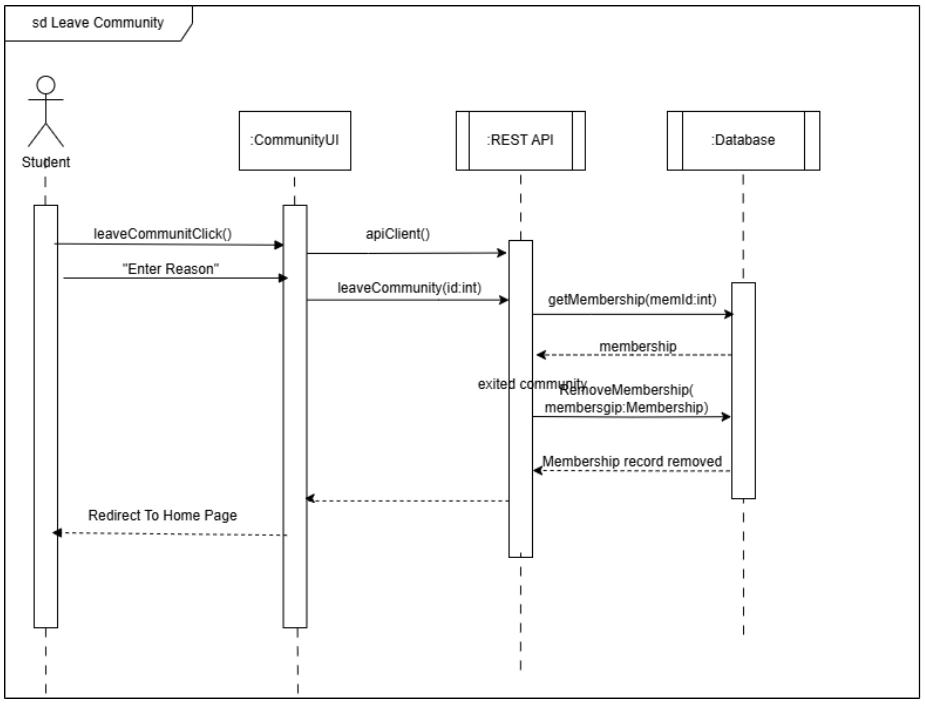
Figure 16: Signup

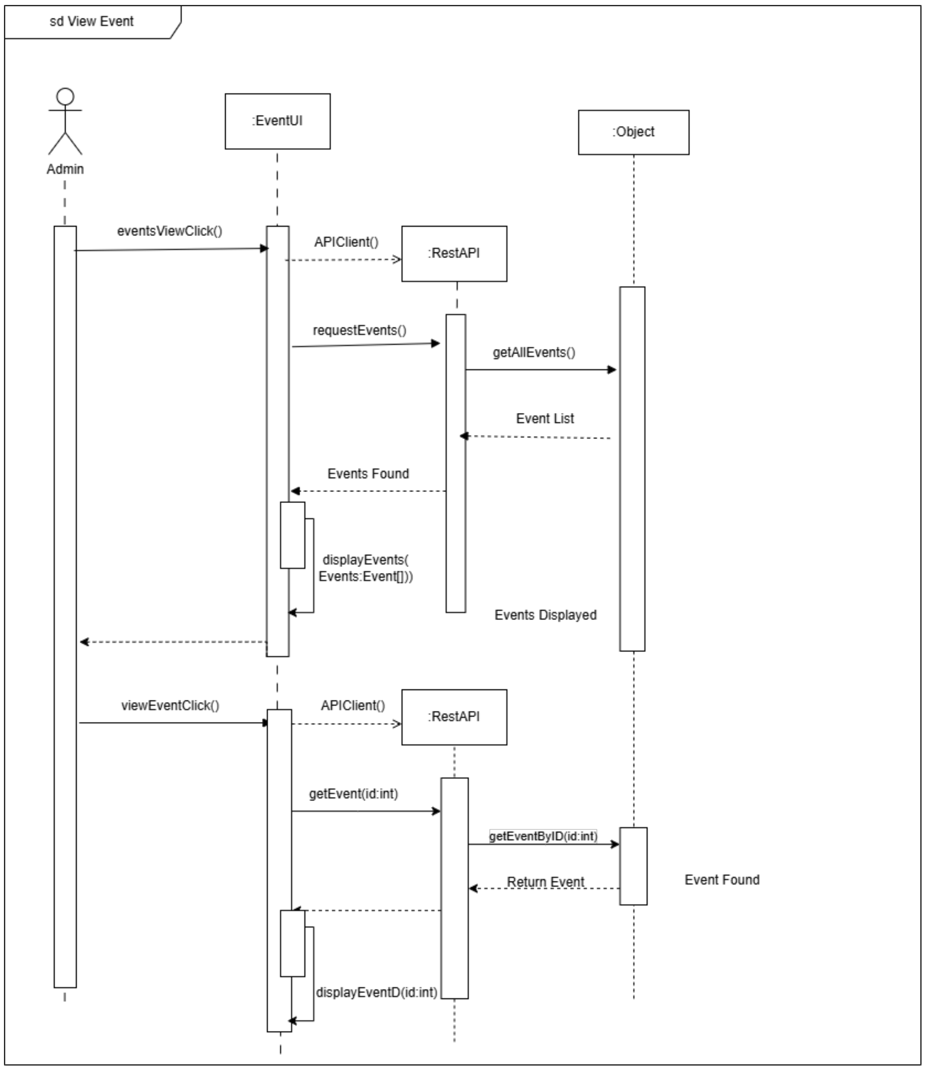

sd Login
User
:SigninUI
:AuthenticationController
:REST API
:DatabaseService
"Enter Credentials"
SignInClick(password: string)
checkTextFields()
Result
Alt
[null/Invalid Input]
displayMessage(message:String)
[Valid Input]
SignInClick(password: string)
Signin(email:string, password:string)
verifyCredentials(email, password)
GetExistingUser(id:int)
user
alt
[Credentials Valid]
User Found
True
user redirected home
[Credentials InValid]
False
User Not Found
redirect()

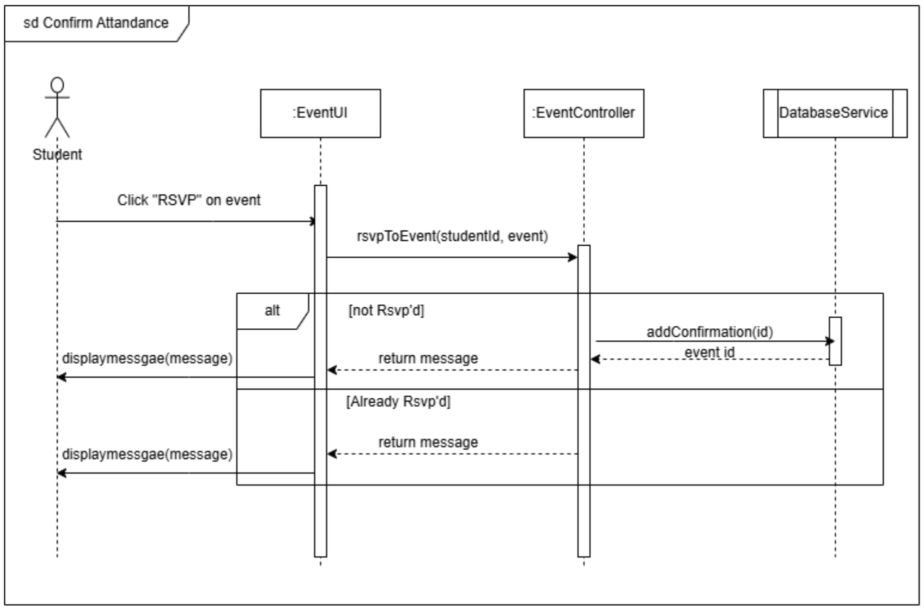

sd Join Community
User
: CommunityUI
:Membershipcontroller
:REST API
:DatabaseService
"Select Community"
DisplayCommunityDetails(id:int)
JoinCommunityClick(c:Communtiy)
JoinCommunity(id: int)
SendMembershipRequest(userid:Int communityid:int)
AddMembership(memId:int)
return Membership
membership request sent
display Message
request sent

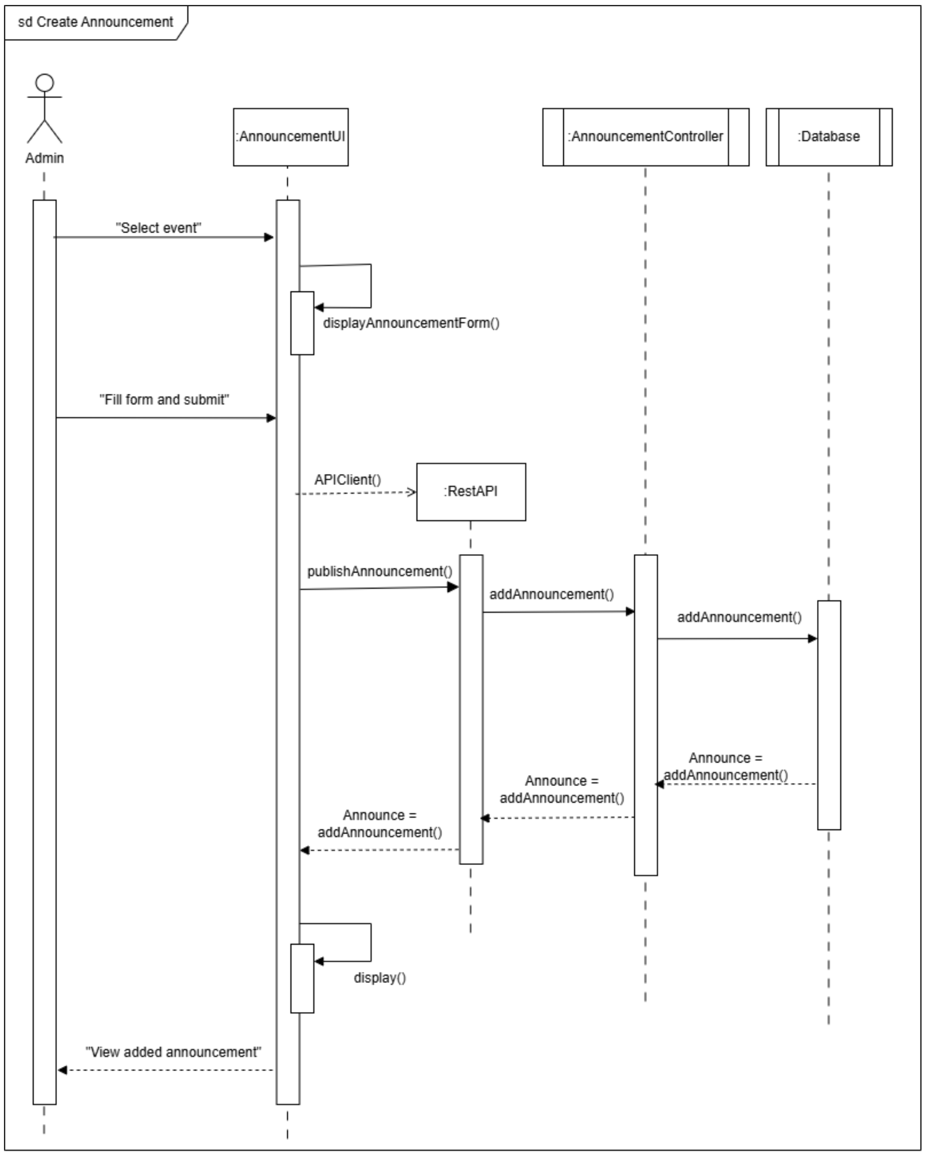

sd Reject Membership
Community Admin
:CommunityUI
:MembershipController
:RESTAPI
:DatabaseService
approveMembershipclick(memId:int)
approveMembership(memId:int, status:string)
UpdateMembershipStatus(id:int,status:string)
Membership Status
Status Updated
membership status
DisplayMessage(message:string)
Membership approoved


sd Approve Membership
Community Admin
:CommunityUI
:MembershipController
:RESTAPI
:DatabaseService
rejectMembershipclick(memId:int)
rejectMembership(memId:int, status:string)
UpdateMembershipStatus(id:int,status:string)
Membership Status
Status Updated
membership status
DisplayMessage(message:string)
Membership Rejected


sd Leave Community
Student
:CommunityUI
:REST API
:Database
leaveCommunitClick()
"Enter Reason"
apiClient()
leaveCommunity(id:int)
getMembership(memId:int)
membership
exited community
RemoveMembership(
membersgip:Membership)
Membership record removed
Redirect To Home Page


sd View Event
Admin
:EventUI
:Object
:RestAPI
eventsViewClick()
APIClient()
requestEvents()
getAllEvents()
Event List
Events Found
displayEvents(
Events:Event[]))
Events Displayed
viewEventClick()
APIClient()
:RestAPI
getEvent(id:int)
getEventByID(id:int)
Return Event
Event Found
displayEventD(id:int)


sd Confirm Attandance
Student
:EventUI
:EventController
DatabaseService
Click "RSVP" on event
rsvpToEvent(studentId, event)
alt
[not Rsvp'd]
addConfirmation(id)
event id
return message
displaymessgae(message)
[Already Rsvp'd]
return message
displaymessgae(message)


sd Create Announcement
Admin
:AnnouncementUI
:AnnouncementController
:Database
"Select event"
displayAnnouncementForm()
"Fill form and submit"
APIClient()
:RestAPI
publishAnnouncement()
addAnnouncement()
addAnnouncement()
Announce = addAnnouncement()
Announce = addAnnouncement()
Announce = addAnnouncement()
display()
"View added announcement"

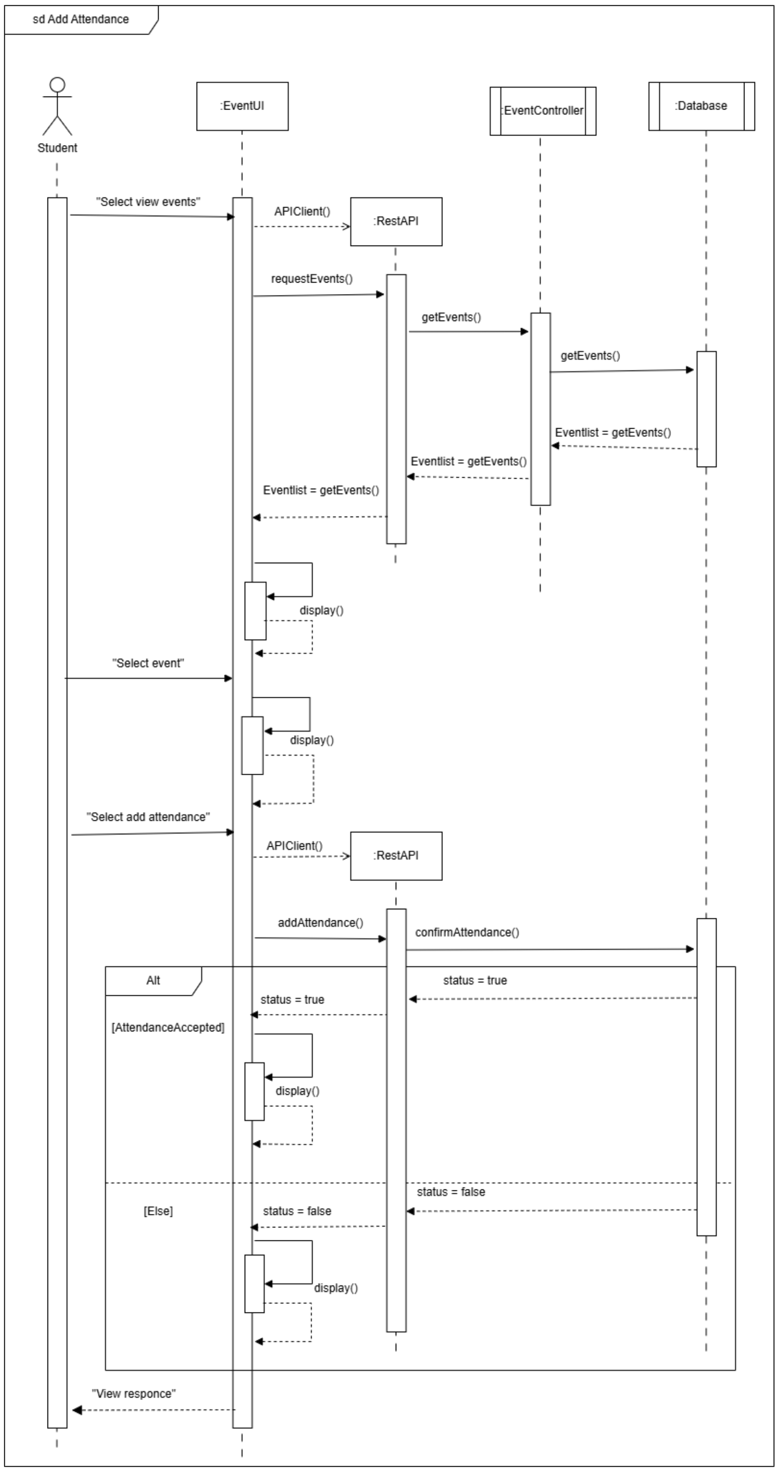
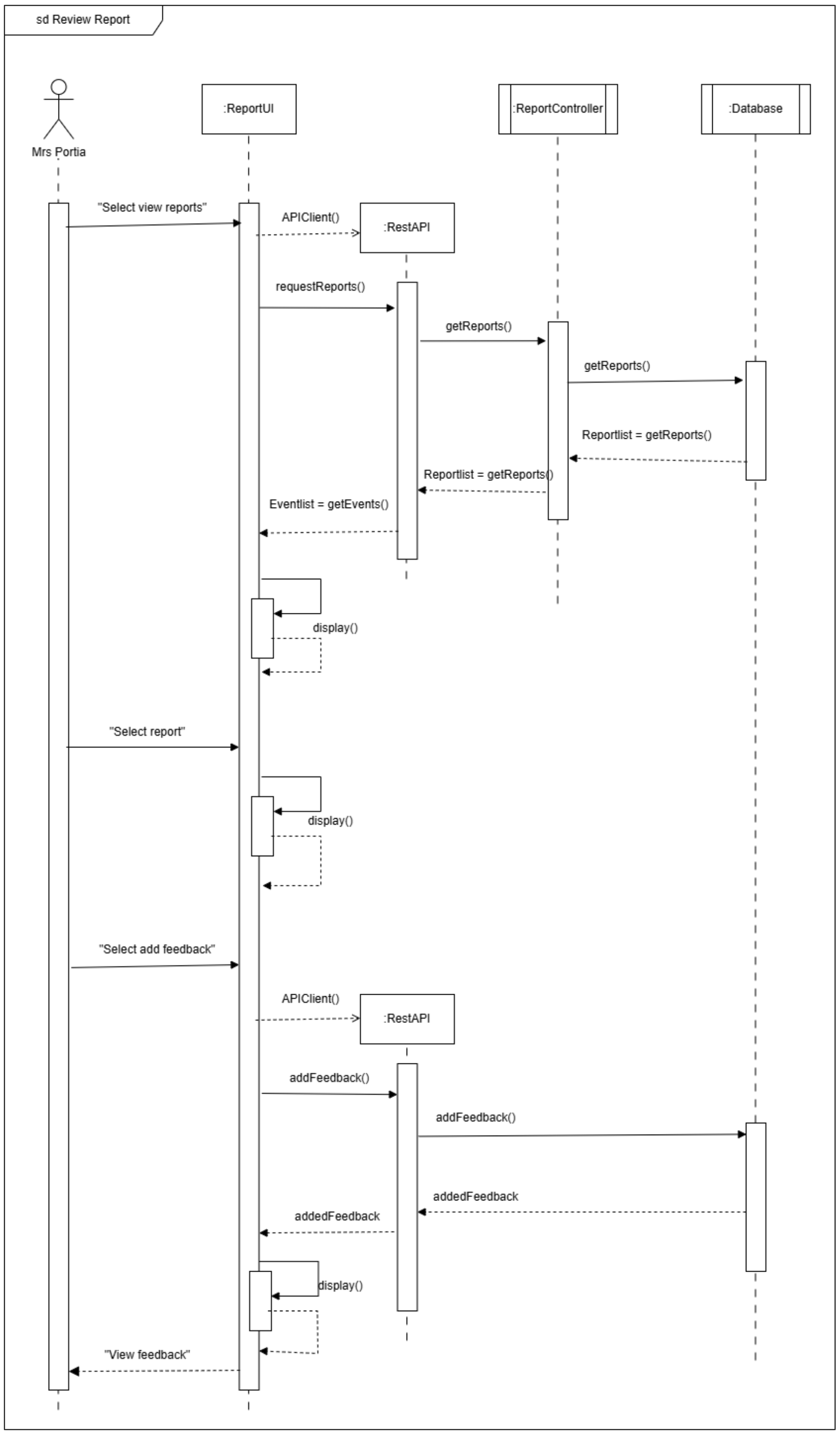


Figure 17: Add Attendance


sd Review Report
Mrs Portia
:ReportUI
:ReportController
:Database
"Select view reports"
APIClient()
:RestAPI
requestReports()
getReports()
getReports()
Reportlist = getReports()
Reportlist = getReports()
Eventlist = getEvents()
display()
"Select report"
display()
"Select add feedback"
APIClient()
:RestAPI
addFeedback()
addFeedback()
addedFeedback
addedFeedback
display()
"View feedback"

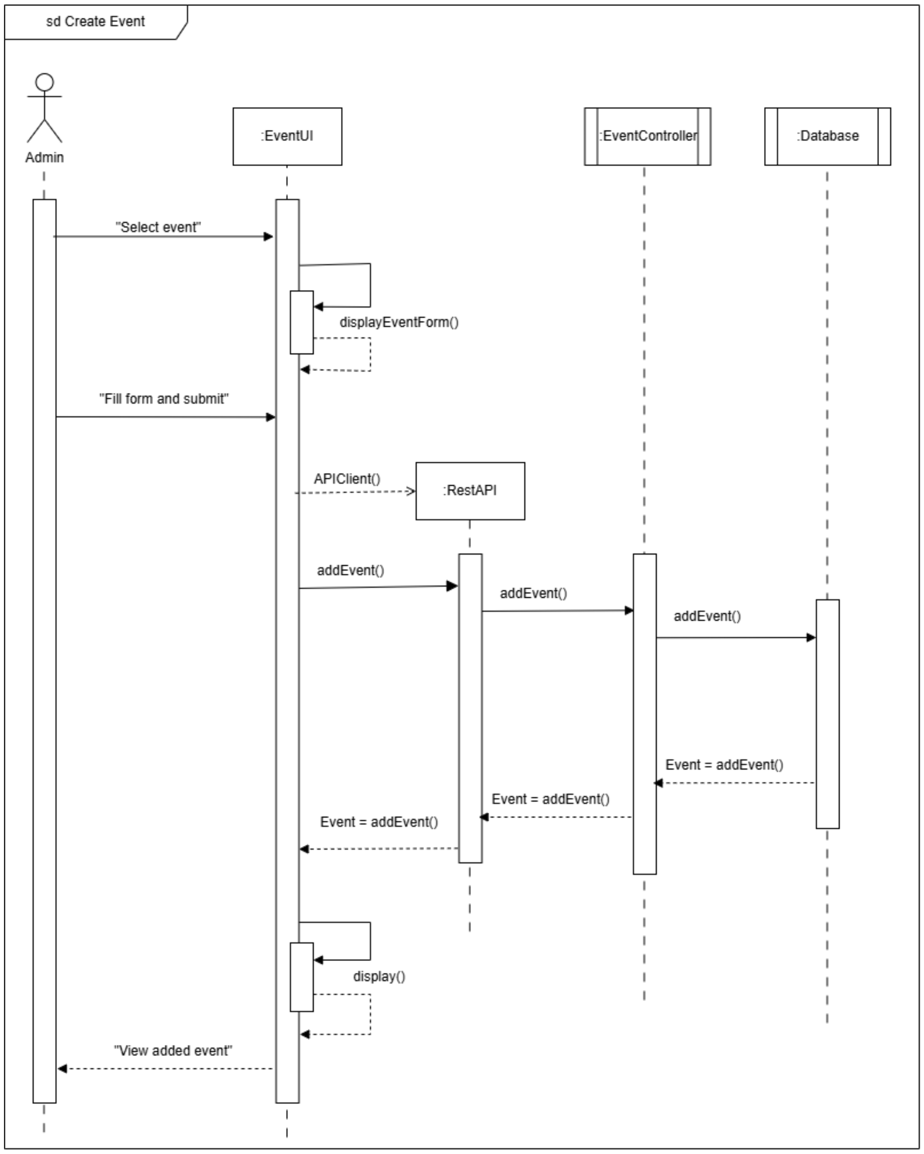


Figure 18: Create Event

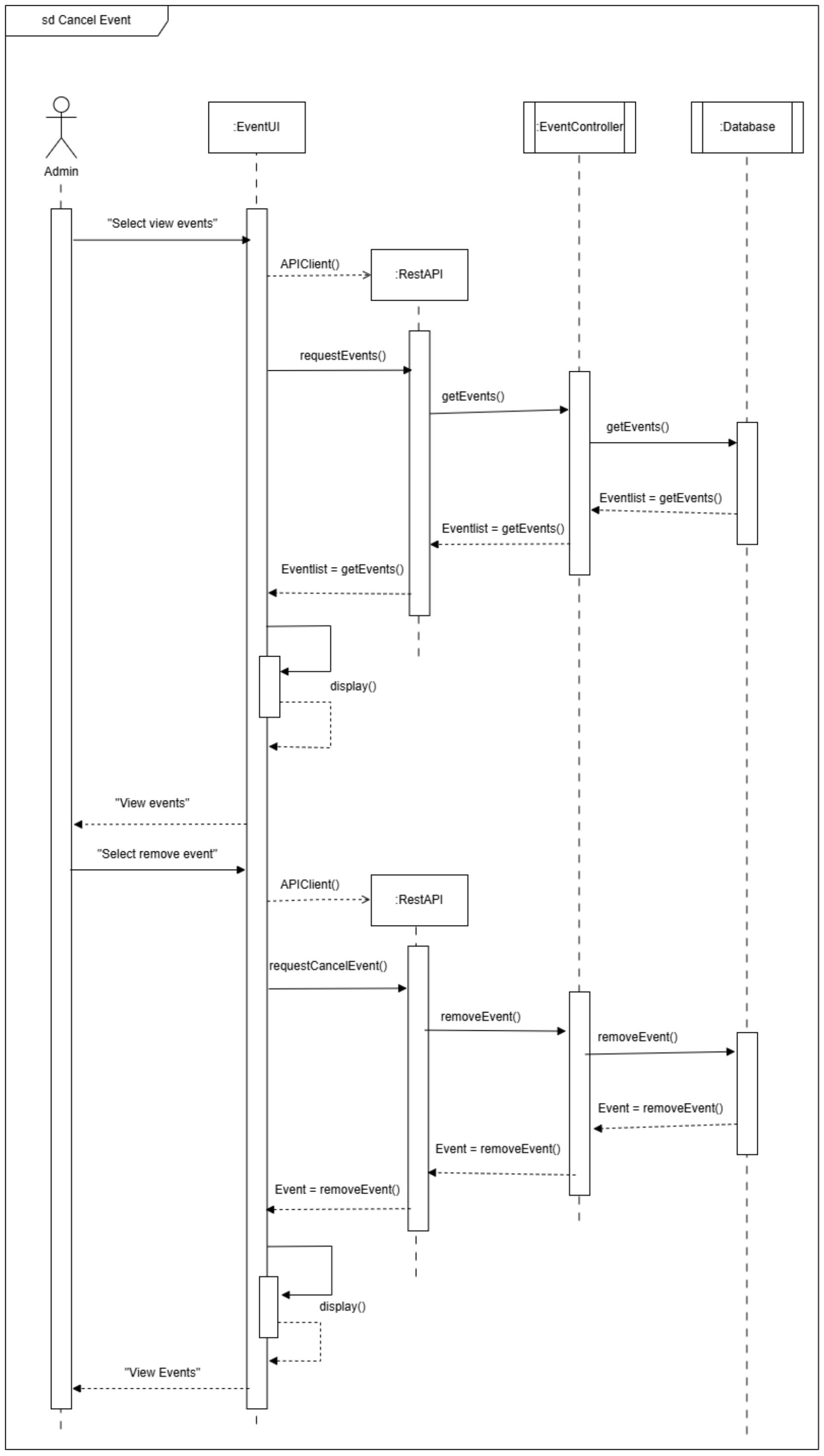


Figure 19: Cancel Event

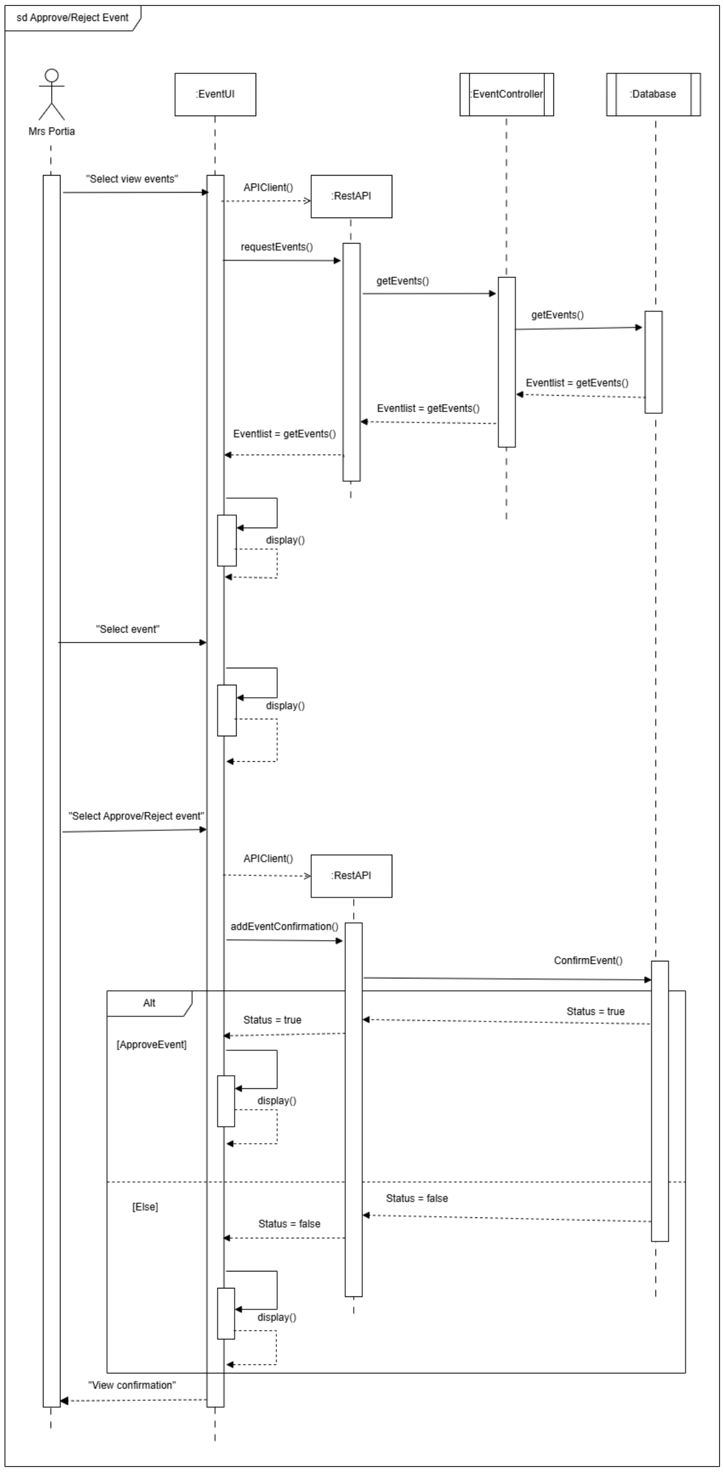


Figure 20: Approve/Reject Event

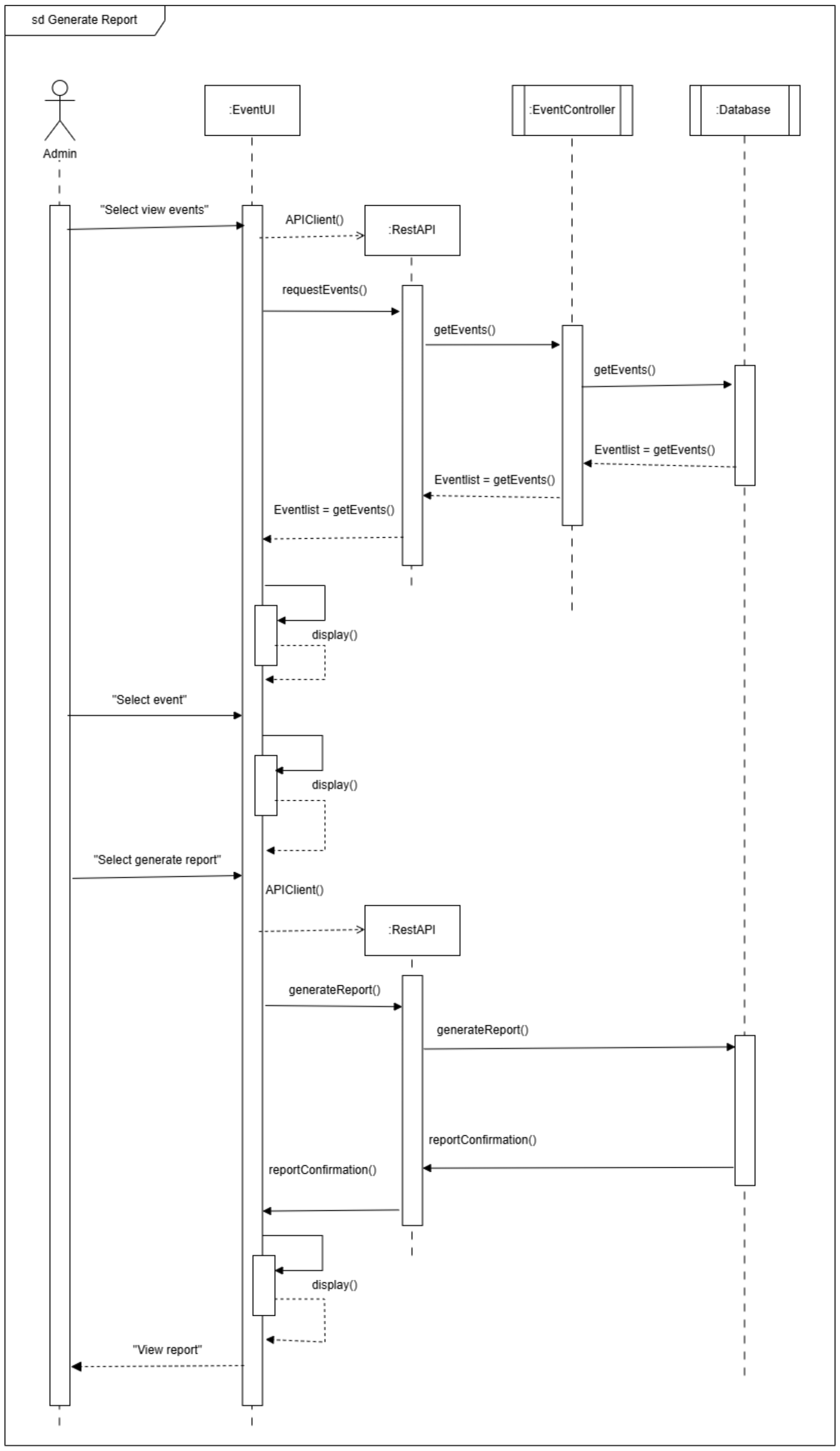


Figure 21: Generate Report

Component and Deployement Diagram

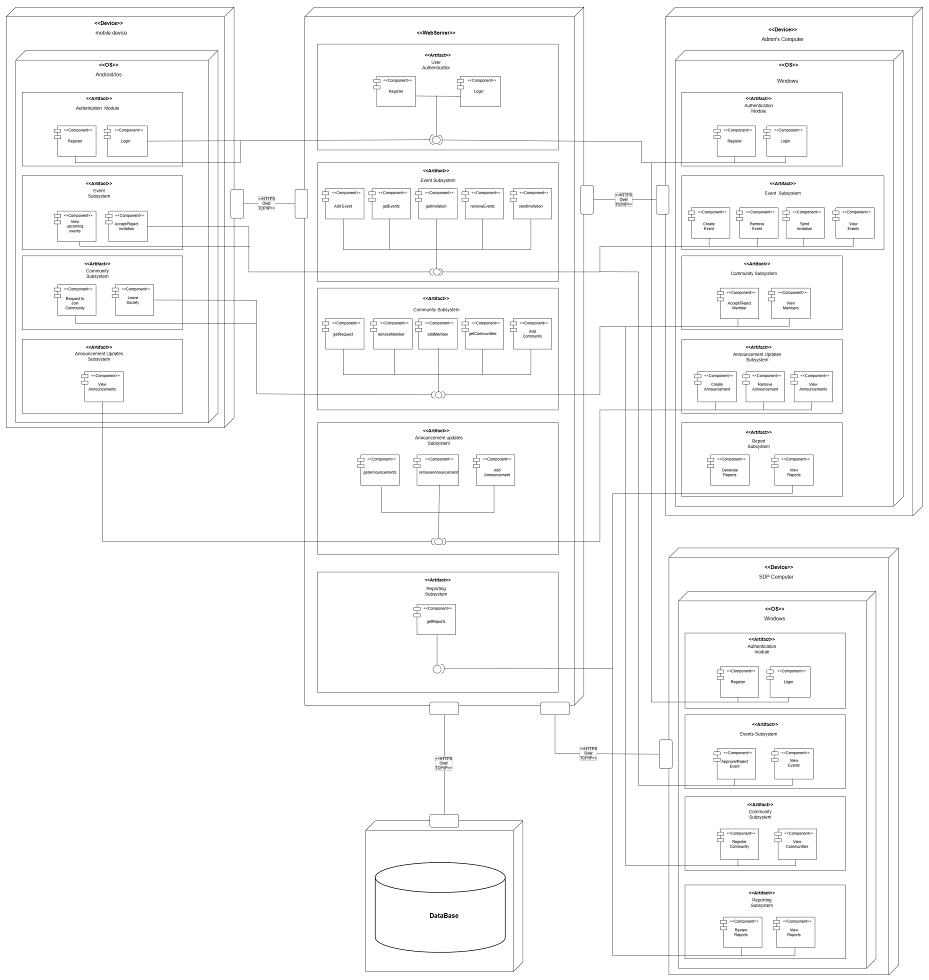


Figure 22: Component and Deployement Diagram

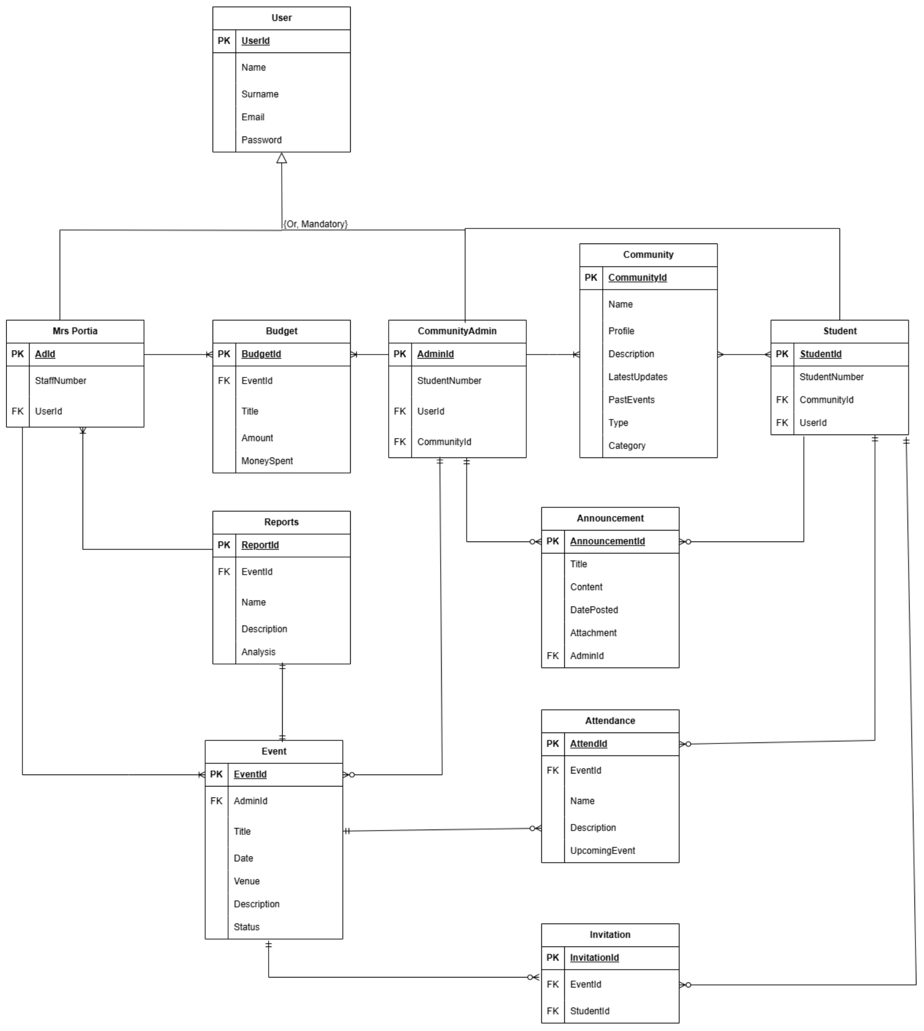


Figure 23: Database Design

Verification and Validation Plan

This Verification & Validation plan ensures that the UJ Student Hub satisfies all specified requirements, is implemented correctly, and meets stakeholder expectations.

Conformation of software to requirement

Event Management Subsystem

requirement 1: Submit Event Proposal

Descrition: Community Admins submit an event proposal to SDP for approval.

Confirmation Activities:

- Check that the proposal form includes all required fields including the Name, date, time(start and end time), self catering requirements and budget and transport requirements.
- Review code handling the submitting of a Proposal to ensure correct payload to SDP endpoint.
- Unit-test submission logic with valid/invalid inputs.
- Perform end-to-end flow whereby the community admin fills form, SDP receives proposal, SDP Approves the Proposal admin sees confirmation.

requirement 2: Book Venue

Descrition: Community Admins can reserve a venue when creating an event, if a venue is required for the event.

Confirmation Activities:

- Verify if the system's user interface displays venue checkbox when creating an event.
- Write integration tests simulating successful/failed bookings for a venue
- run a real-world scenario to create event that requires a venue and confirm booking status in database and calendar

requirement 3:Request Funds & Transport

Descrition: Admins may request event funds for an event or transport through a workflow

Confirmation Activities:

- Check that the Request Form Has all the required fields form including the pickup dates and drop off dates and driver's particulars.
- Test the API endpoints /request/funds and /request/transport with mock SDP responses.
- Validate notifications sent to SDP and admin for approvals and confirmations.

- Emitate a stakeholder walkthrough whereby community admin submits request SDP dashboard shows request.

requirement 4: Confirm Attendance

Description: Students mark their intention to attend an invite event through RSVP.

Confirmation Activities:

- Verify the RSVP control appears on event details.
- Unit-test the RSVP API with both choices.

requirement 5: Add Attendance via generated Code

Description: at the event, students enter a generated attendance code when the event is live which they will receive if they confirm their attandace.

Confirmation Activities:

- Check code-entry form appears only when event is live and when student the student scan the event QR Code.
- Test code-validation logic against valid/invalid attandance codes.
- Simulate rush conditions whereby there are mulltiple entries of codes from multiple attandes simultaneously.

requirement 6: Receive Notifications

Description: Students get notified of event updates (creations, cancellations, reschedules)

Confirmation Activities:

- Verify user enrollment to the community settings in user profile.
- Mock push and/or email notifications for each event update.
- Test How it perfoms both on web/mobile clients.

Community Management Subsystem

requirement 1: Create/Update Community Profiles

Descrition: community admins define community and and community branding

Confirmation Activities

- Ensure Create Community form covers all profile fields, name, mission, vision, Category
- Unit-test CRUD operations on community through the /community endpoint
- Integration testing ensures community list updates in real time.
- simulate admin walkthrough: create & update community, verify changes appear and are visible to community members.

requirement 2: Join Community

Descrition: Students enrol in a community, pending admin approval

Confirmation Activities

- Check Join button only appears for members who are not part of the community
- Test enrolment API, whether the status transitions from pending to approved for a member.
- emulate student workflows: submit membership request , Mock admin approval, student dashboard home dashboard updated

requirement 3: Search/Filter Communities & Events

description: Students locate communities/events via keywords, filters

Confirmation Activities

- Confirm search/filter UI controls on both community and event pages.
- Load-test search API with large datasets to ensure a less than 500 ms response form the search/filter.
- Test combinations of filters including by event/community category, event date, community members (asc/desc).

Announcement Subsystem

Requirement 1: Create Announcement

Description: Admins publish announcements with a title, content and optional attachment

Confirmation Activities:

- Ensure the design supports Input fields for the title, content and file attachments
- Confirm that announcements are Saved and viewable in a page
- Trigger notification for users
- Write tests to:
    1. Publish announcements
    2. Verify visibility to target users
    3. Confirm correct handling issues

Requirement 2: View Announcement

Description: Users view the announcements feed and receive push/email alerts

Confirmation Activities:

- Confirm UI Announcements tab lists all the latest items from recent to oldest.
- Unit-test the getAnnouncements() in the /announcement endpoint and that API returns correct dataset.
- test the push notifications/email modules for receiving announcemnt updates intergratively

Reporting Subsystem

Requirement 1: Generate Event Report

Description: Community Admins can generate and submit event reports

Confirmation Activities

- A file upload interface supporting a pdf or a docx format
- A guildeline pop up showing what the report should contain(e.g attendance summary, challenges, photos, etc.)
- Verify that the uploaded files are stored securely and linked to the correct event and community
- Write tests to:
    1. Upload valid report files(pdf/docx)
    2. Reject unsupported file formats
    3. Confirm uploads are associated with correct events
    4. Confirm error handling for missing files or incorrect formats

Requirement 2: Review Event Report

Description: Community Admins generate and submit event reports

Confirmation Activities

- Confirm View Reports UI shows list of generated reports with the options to filter /sort
- Unit-test the getReports() in the /reports endpoint and API returns matching entries.
- Integration test: after RP-1, retrieve report display correct field values.
- UAT: stakeholder opens report and verifies accuracy.

Testing Plan

This plan defines how we will verify and validate the UJ Student Hub's Event Management, Community Management, Announcements Updates, and Reporting subsystems. Its objectives are to ensure each requirement is implemented correctly, defects are caught early, and stakeholder expectations are met.

Scope

- **In Scope**
  - Event Management
  - Community Management'
  - Announcements Updates
  - Reporting
- **Out of Scope**
  - User authentication & profile management
  - Third-party integrations outside SDP endpoints

Objectives

- Verify if every feature to be built aligns to its requirements
- Validate completeness, consistency, and correctness of functionality
- Confirm if the sytstem is usable and reliable in real-world scenarios

Test Environment

- **Hardware:**
  - **Developer Workstations:** Standard personal-issued laptops (Triple core processor)
  - **Test Server:** Shared UJ Server
  - **Mobile Devices:** Android phone & tablet, iPhone simulator
- **Software:**
  - **Backend:** Java 17 Spring Boot services
  - **Frontend:** React served via Nginx; React Native (Android/iOS) on Expo
  - **Database:** MySQL
  - **CI/CD:** GitHub Actions for build/test; Docker Compose for local integration
  - **Test Frameworks:**
    - Unit: JUnit 5, Jest
    - Integration: Postman
    - System/UI: Cypress, Expo-CLI

Test Strategies

Level	Purpose
Unit Testing	Verify individual modules & functions
Integration Testing	Validate service-to-service interactions
Performance Testing	Benchmark key APIs under load
User Acceptance Testing (UAT)	Confirm stakeholder satisfaction

Test Cases

1. User Registration: Ensure a new user can successfully register using their credentials.
2. User Login: Verify that a registered user can log in successfully.
3. Event Creation: Validate that an admin can create a new event by submitting pre-required requests(event proposals,event funding/transportation(if required)).
4. Event attandance confirmation: Ensure a user can confirm attandance for an event and that they can be successfully registred.
5. Add Event Attendance: Confirm that a user can add attendance code for an event

Documentation Comprehensive documentation will be maintained throughout the testing process, including: - Test Plans: Outline the overall testing strategy and approach.
- Test Cases: Detailed steps, inputs, and expected results for each test case. - Test Results: Records of test execution and outcomes. - Defect Reports: Logs of identified defects, their status, and resolution details.

Defect Management

- **Reporting**: Any issue must be logged in our GitHub Issues tracker with a clear title, description, and reproduction steps.

- **Severity Levels**:
  - **High**: Blocks core functionality-must be fixed before the next demo.

  - **Medium**: Affects major features but has a workaround-fix in the current sprint.

  - **Low**: Minor bugs or UI glitches-address if time allows.

- **Workflow**:
  1. Team member assigns the issue to themselves.

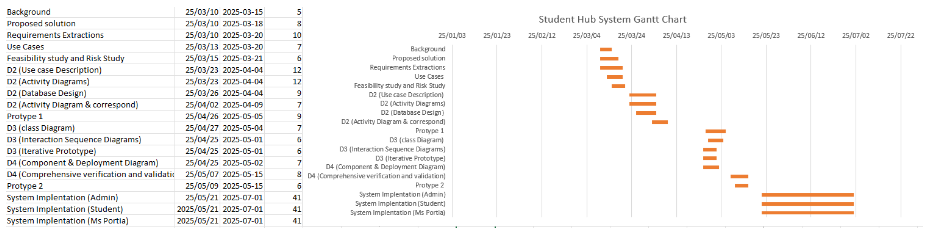
2. Fix the bug and add a short test or screenshot.
3. Mark the issue “Ready for Review”
4. Another team member verifies the fix and closes the issue.

Test Case Design

Test Name	Input	Condition / Test	Expected Output
Display upcoming events	Student clicks **Events**	Upcoming events exist	List of upcoming events shown, ordered by date
Display “no events” message	Student clicks **Events**	No events in database	Message is “No upcoming events”
Add attendance (success)	Valid attendance code	Event is live and the code matches event	Attendance recorded; confirmation toast
Add attendance – invalid code	Incorrect code	Event is live but code invalid	Invalid code error; record null
Add attendance – duplicate	Previously used code	Student already marked present	Attendance already captured message
Submit event proposal	Complete event form	Logged-in community admin	Proposal status is **“Pending”**; admin sees success banner
Submit event proposal – missing field	Event form missing date fields	Logged-in community admin	validation highlights missing field; proposal not sent
Cancel upcoming event	Admin clicks **Cancel**	Event status is “Scheduled”	Event status is **“Cancelled”**; notifications sent
Cancel event – unauthorized	Student clicks **Cancel**	User role not admin	Access denied
View registrations list	Admin selects **Registrations**	At least one registration	Table of registered students displayed
View registrations – none	Admin selects **Registrations**	Zero registrations	No registrations yet message shown
Join community request	Student clicks **Join**	Student not a member	Membership request saved; admin notified
Join community – already member	Student clicks **Join**	Student already approved	Info banner stating You are already a member

Test Name	Input	Condition / Test	Expected Output
Generate event report (PDF)	Valid *.pdf* file upload	Logged-in community admin	Report stored; SDP receives notification
Generate event report – invalid file	Upload *.txt* file	Logged-in community admin	“Unsupported format” validation error
Publish announcement	Title, body entered	Admin role	Announcement appears in feed; push/email sent
Publish announcement – missing title	Body only	Admin role	Title field flagged; announcement not saved
Receive cancellation notification	System triggers event cancel	Student subscribed to community	Push/email alert contains event name and **cancelled** status
Filter events by category	Select category to **Sports**	Sports events exist	List shows only sports events
Search communities	Keyword **Debate**	Communities containing keyword exist	Matching communities listed with join buttons

Gantt Chart



Background	25/03/10	2025-03-15	5
Proposed solution	25/03/10	2025-03-18	8
Requirements Extractions	25/03/10	2025-03-20	10
Use Cases	25/03/13	2025-03-20	7
Feasibility study and Risk Study	25/03/15	2025-03-21	6
D2 (Use case Description)	25/03/23	2025-04-04	12
D2 (Activity Diagrams)	25/03/23	2025-04-04	12
D2 (Database Design)	25/03/26	2025-04-04	9
D2 (Activity Diagram & correspond)	25/04/02	2025-04-09	7
Protype 1	25/04/26	2025-05-05	9
D3 (class Diagram)	25/04/27	2025-05-04	7
D3 (Interaction Sequence Diagrams)	25/04/25	2025-05-01	6
D3 (Iterative Prototype)	25/04/25	2025-05-01	6
D4 (Component & Deployment Diagram)	25/04/25	2025-05-02	7
D4 (Comprehensive verification and validatic	25/05/07	2025-05-15	8
Protype 2	25/05/09	2025-05-15	6
System Implentation (Admin)	25/05/21	2025-07-01	41
System Implentation (Student)	2025/05/21	2025-07-01	41
System Implentation (Ms Portia)	2025/05/21	2025-07-01	41

Student Hub System Gantt Chart
25/01/03
25/01/23
25/02/12
25/03/04
25/03/24
25/04/13
25/05/03
25/05/23
25/06/12
25/07/02
25/07/22
Background
Proposed solution
Requirements Extractions
Use Cases
Feasibility study and Risk Study
D2 (Use case Description)
D2 (Activity Diagrams)
D2 (Database Design)
D2 (Activity Diagram & correspond)
Protype 1
D3 (class Diagram)
D3 (Interaction Sequence Diagrams)
D3 (Iterative Prototype)
D4 (Component & Deployment Diagram)
D4 (Comprehensive verification and validation)
Protype 2
System Implentation (Admin)
System Implentation (Student)
System Implentation (Ms Portia)

Figure 24: Gannt Chart

Resource Breakdown

Team roles and contribution

Team Member	Design Role(s)	Key Contribution
Member 1	UX/UI Designer, Documentation	Designed wireframes, created personas, and wrote design-justification documents
Member 2	UI Designer, Researcher	Worked on visual design and conducted user research
Member 3	Project Planner, UX/UI Designer	Coordinated task timelines, reviewed prototypes, and refined wireframe feedback
Member 4	Information Architect, Editor	Structured site flow and edited documentation

Hardware

Equipment	Source
Laptops/computers	Personal devices
Wi-Fi access	University & home wifi routers

Software

Tool	Purpose	Cost
Draw.io	UI/UX design and wireframes	R 0
Word/markdown	document creation	R 0
VS Code	Simple HTML/CSS/javascript prototypes	R 0

Other resources

- Mentor feedback.
- Sponsor feedback
- Universit computer lab access.
- Online tutorials.

Estimated Total Project Cost

Resource	Cost	Notes
Draw.io	R 0	Free
markdown	R 0	Free and open-source
VS Code	R 0	Free and open-source
Internet	R 0	Existing university free Wi-Fi
comupters	R 0	Existing personal working devices

Most costs are zero because the tools are freely available to us ; only personal hardware and internet usage are considered.

Time Estimation

The following estimates cover the **design and prototyping phase**

Task	Estimated Time	Notes
User research & persona creation	2 weeks	Identify student/admin requirements
Wireframe design	4 days	initial rough skeches of the system
Documentation writing	Weekly	design documentation updated
Team collaboration & meetings	Weekly	plan weekly meetings with mentor and team members
Feedback & iterative updates	45 minutes	Incorporate sponsor and mentor feedback

Total estimated design-phase time: ~10 weeks (semester 1)