# *Software Design Specification (SDS)*

## *Project Title: University Society Management System*

| | |
|---|---|
| ***Course Code*** | ***CS3009- Software Engineering*** |
| ***Instructor*** | ***Miss. Mahrukh*** |
| ***Project Team*** | ***Ramin Raees(22k-4654)***<br>***Muniba Mehboob(22k-4338)***<br>***Rakhshanda(22k-4461)*** |
| ***Submission Data*** | ***06 - May - 2025*** |

## Definitions, Acronyms, and Abbreviations

**UI** – *User Interface*: The graphical interface through which users interact with the system, including buttons, forms, dashboards, and other visual elements.

**UX** – *User Experience*: The overall experience and satisfaction of users when interacting with the system, focusing on ease of use and intuitive design.

**DB** – *Database*: A structured collection of data that can be easily accessed, managed, and updated. This system uses MySQL as the relational database.

**HTTP** – *HyperText Transfer Protocol*: A communication protocol used for transferring data over the web between client (browser) and server.

**CRUD** – *Create, Read, Update, Delete*: The four essential operations used for managing data in any software application.

**HTML** – *HyperText Markup Language*: The standard language used to create and design the structure of web pages.

**CSS** – *Cascading Style Sheets*: A stylesheet language used for describing the visual presentation of web pages written in HTML.

**SQL** – *Structured Query Language*: A programming language used for managing and querying relational databases.

**Admin** – *Administrator*: A user with the highest level of access who can manage societies, events, users, and system data.

**Student User**: A general user (student) of the system who can view societies, apply for memberships, and receive notifications.

**Society**: A university-approved group that organizes extracurricular activities for students, managed through the system.

**Event:** Any activity organized by a society such as workshops, competitions, or seminars, and managed through the platform.

**Membership:** A student's association with a society, recorded in the system after admin approval.

**Notification:** A message generated by the system to inform users of upcoming events, inductions, or announcements.

## Table of Contents

# 1. Introduction

## 1.1 Purpose of Document

This document provides the Software Design Specification for the University Society Management System. It outlines the design approach, system functionalities, and user interactions in detail. The primary audience includes software developers, project supervisors, testers, and university administration. The design of this system follows a structured design methodology, ensuring clear data flow and logical module separation throughout the system.

## 1.2 Intended Audience

The intended readers of this document are:

- Developers responsible for implementing the system.
- Testers who will validate the system functionality.
- University administrators who will use the system.
- Supervisors or instructors evaluating the project.
- Stakeholders interested in system features and performance.

## 1.3 Document Convention

This document is written using the Calibri font, with:

- Headings in font size 16pt – 18pt
- Body text in font size 11pt – 12pt
  Numbering conventions (e.g., 1.1, 1.2) are used for clear referencing.

## 1.4 Project Overview

The University Society Management System is a web-based application designed to streamline the management of university societies. It enables students to register/login, explore societies, apply for membership, and view upcoming events. Administrators can create societies, approve applications, post events, track attendance, manage financials, and send notifications. The system is built using:

- HTML and CSS for a responsive frontend,

- Python for backend logic,
- MySQL as a relational database to store structured data.

### 1.5 Scope

The system covers the following:

- User authentication (student/admin)
- Society and event management
- Membership applications and approvals
- Notifications and announcements
- Attendance and finance tracking

The system does not include:

- Real-time chat between users
- AI-based recommendations
- Deep integration with external ERP systems (in current version)

# 2. Design Considerations

## 2.1 Assumptions and Dependencies

### Assumptions

- Users have access to a stable internet connection and a modern web browser.
- Users (students/admins) possess basic computer literacy.
- All societies listed are officially recognized by the university.
- Users provide correct and honest information during registration and applications.
- The system will be used within a limited number of concurrent users (university scale).

**Dependencies**

- The system requires a web server that supports HTML, CSS, Python, and MySQL.
- The MySQL database must remain available and properly configured.
- Role-based access control must be enforced for secure admin and student operations.
- Future updates or features may depend on external libraries or APIs (e.g., for notifications).
- Browser compatibility is assumed for Chrome, Firefox, and similar major browsers.

## 2.2 Risks and Volatile Areas

The development and deployment of the University Society Management System involve certain risks and volatile areas that must be considered to ensure the success and stability of the project:

**1. Data Security and Privacy**

There is a risk of unauthorized access to sensitive student and admin data (e.g., login credentials, personal information, financial records). Proper authentication, authorization, and encryption mechanisms must be implemented.

**2. Database Integrity**

Loss or corruption of relational data due to unexpected server crashes, poor query handling, or concurrent access can compromise the entire system. Proper transaction management and backup strategies are necessary.

**3. Scalability Issues**

As the number of users and societies grows, the system may experience performance degradation. Scalability in both the frontend and backend must be considered early in development.

**4. User Misuse or Misentry**

Students or admins may unintentionally submit incomplete or incorrect data (e.g., during society registration or event creation), leading to logical errors. Input validation and error handling must be enforced.

**5. Dependency on Internet Connectivity**

Since it is a web-based system, any internet outage or server downtime may impact the availability of the platform to its users, especially during peak times (e.g., event registration).

**6. Technical Compatibility**

Inconsistencies in browser behavior, screen sizes (desktop vs. mobile), or server environments (Python, MySQL versions) can cause unexpected bugs or UI glitches.

**7. Role Management Conflicts**

Incorrect assignment or overlap of roles (e.g., a student assigned admin privileges by mistake) may lead to functionality misuse. Clear role-based access control must be maintained.

**8. Maintenance and Updates**

Frequent updates, new features, or bug fixes may cause instability if not tested thoroughly. A proper version control and testing mechanism should be implemented.

**9. Incomplete Requirement Gathering**

Changes in university policies or incomplete understanding of admin/student workflows can lead to rework or missing features. Continuous stakeholder feedback should be maintained.

# 3. System Architecture

This section provides a high-level overview of how the University Society Management System is organized into subsystems and how these components interact to fulfill system functionalities. The architecture supports modular design, maintainability, and scalability. The design aims to separate concerns between user interactions, business logic, and data management.

## 3.1 System Level Architecture

The system is decomposed into three primary elements:

- **Client Side (User Interface)**
  This includes HTML/CSS pages accessed by students and admins. It handles user interaction such as login, form submissions, viewing society data, and receiving notifications.

- **Server Side (Application Logic)**
  The Python backend processes requests from the frontend, applies the business rules (e.g., approving membership, adding events), and communicates with the database.

- **Database (MySQL)**
  All data is stored in a relational MySQL database including user credentials, society information, events, reports, and financial records.

**Relationships Between Elements**

- The frontend communicates with the backend via HTTP.

- The backend uses SQL queries to interact with the database.
- The backend handles form validation, session management, and access control.

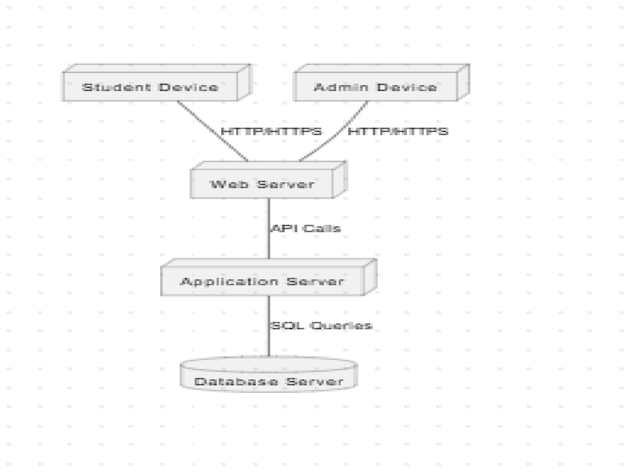
**External Interfaces**

- The system may optionally integrate with an email system for notifications.
- Future integration with the university's ERP system is considered.

**Design Considerations**

- The system is hosted on a web server that supports Python (e.g., Flask or Django) and MySQL.
- All components are designed to work together over a secure HTTPS connection.
- Error handling is managed in the backend using exception handling and user alerts on the frontend.

### Deployment Diagram:



## 3.2 Software Architecture

The software architecture follows a **three-tier model** that separates responsibilities across different layers to improve clarity and maintainability.

**1. User Interface Layer (Frontend)**

- Built using **HTML and CSS**
- Manages user input (forms, buttons, etc.)
- Displays societies, events, and reports to students and admins
- Sends requests to the backend via HTTP

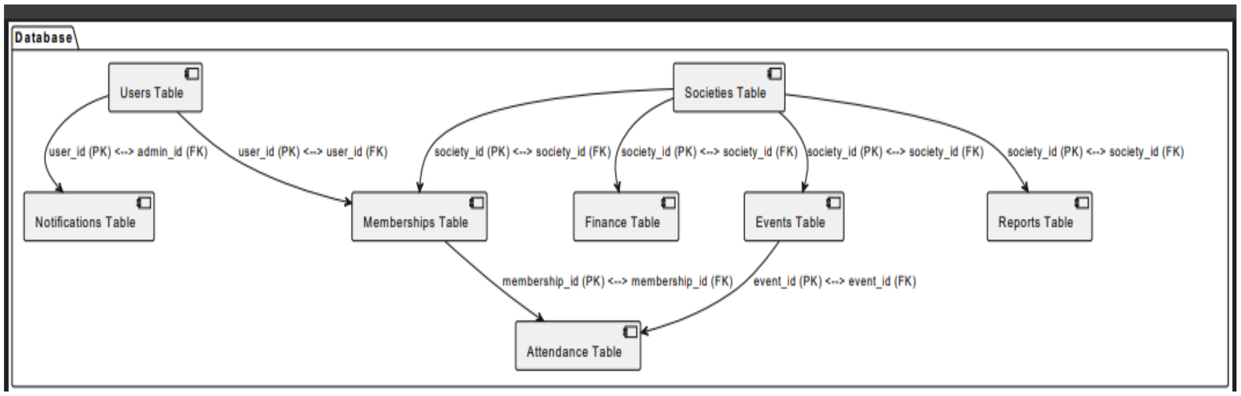
**2. Middle Tier (Application Logic / Backend)**

- Developed using **Python**
- Handles all logic related to authentication, society management, event processing, and financial operations
- Validates and processes data received from the frontend
- Communicates with the database

**3. Data Access Layer (Database)**

- Built using **MySQL**
- Stores all persistent data: users, societies, events, notifications, etc.
- Ensures referential integrity with foreign keys
- Responds to queries and updates triggered by the backend

## Component Diagram:


Database
Users Table
Societies Table
user_id (PK) <--> admin_id (FK)
user_id (PK) <--> user_id (FK)
society_id (PK) <--> society_id (FK)
society_id (PK) <--> society_id (FK)
society_id (PK) <--> society_id (FK)
society_id (PK) <--> society_id (FK)
Notifications Table
Memberships Table
Finance Table
Events Table
Reports Table
membership_id (PK) <--> membership_id (FK)
event_id (PK) <--> event_id (FK)
Attendance Table

# 4. Design Strategy

The design of the University Society Management System is guided by principles of modularity, scalability, and ease of use. The system is structured using a three-tier architecture (User Interface, Application Logic, and Database Layer), allowing clean separation of concerns and facilitating future upgrades.

### Future System Extension or Enhancement

To support future improvements, such as mobile app integration, online payment systems, or ERP linkage, the architecture is designed to be modular. Each component (e.g., user management, society data, finance tracking) is developed independently, allowing isolated updates without affecting the entire system.

### System Reuse

Common features like login authentication, form validation, and role-based access control are developed as reusable modules. This ensures code reuse across various parts of the system and simplifies maintenance.

### User Interface Paradigms

The system uses a clean, responsive web interface built with HTML and CSS. Layouts follow a user-centered design approach, prioritizing ease of navigation, clarity, and accessibility for both students and administrators. Forms and dashboards are logically organized and mobile-friendly.

### Data Management

All persistent data is stored in a MySQL relational database. Proper normalization and use of foreign keys ensure data consistency. CRUD operations are handled via backend scripts, ensuring security and validation. Data such as reports, membership records, and event logs are retained over time for future access and analysis.

### Concurrency and Synchronization

Although this version of the system assumes single-session interactions per user, backend scripts are designed to safely handle multiple user requests concurrently. Form submissions and database updates are managed to avoid race conditions, ensuring consistent and accurate data storage.

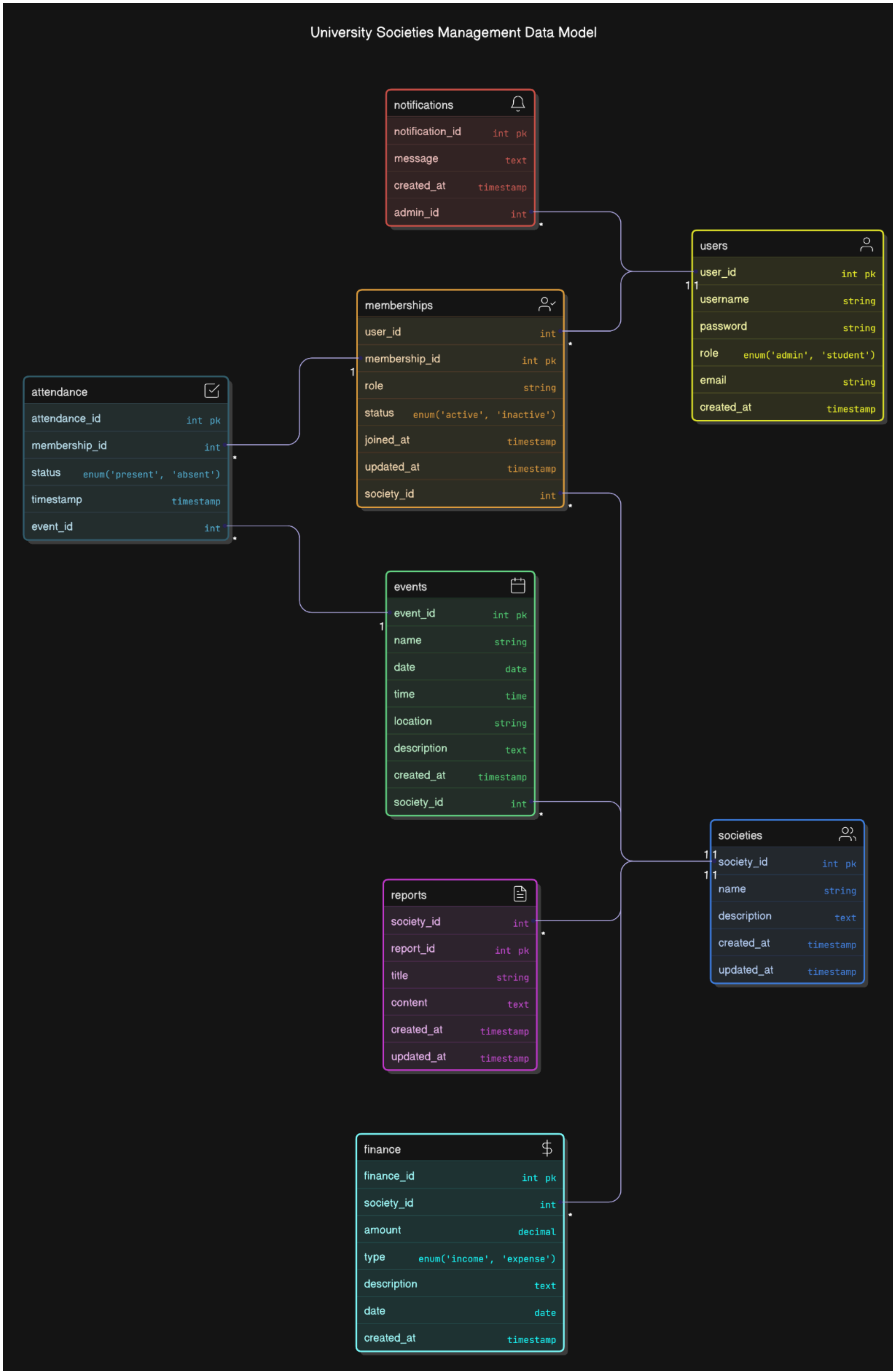
# 5. Detailed System Design

This section elaborates the internal structure of the University Society Management System including class structure, database modeling, and user interface screens.

## 5.1 Database Design

The backend of the system uses MySQL for structured data storage. Relationships between tables are handled via foreign keys and normalization.

### 5.1.1 ER Diagram


University Societies Management Data Model
notifications
notification_id int pk
message text
created_at timestamp
admin_id int
users
user_id int pk
username string
password string
role enum('admin', 'student')
email string
created_at timestamp
memberships
user_id int
membership_id int pk
role string
status enum('active', 'inactive')
joined_at timestamp
updated_at timestamp
society_id int
attendance
attendance_id int pk
membership_id int
status enum('present', 'absent')
timestamp timestamp
event_id int
events
event_id int pk
name string
date date
time time
location string
description text
created_at timestamp
society_id int
societies
society_id int pk
name string
description text
created_at timestamp
updated_at timestamp
reports
society_id int
report_id int pk
title string
content text
created_at timestamp
updated_at timestamp
finance
finance_id int pk
society_id int
amount decimal
type enum('income', 'expense')
description text
date date
created_at timestamp

### 5.1.2 Users Table

- **Purpose:** Stores the login credentials and personal details of all users (students and admins) in the system.
- **Attributes:**
  user_id (INT, Primary Key): Unique identifier for each user.
  username (VARCHAR): Username for login.
  password (VARCHAR): Encrypted password for security.
  role (ENUM: 'admin', 'student'): Defines the user role. Admins have full access, while students have limited access.
  email (VARCHAR): User's email address.
  created_at (TIMESTAMP): Timestamp of when the account was created.

### 5.1.3 Societies Table

- **Purpose:** Holds information about the various societies in the university.
- **Attributes:**
  society_id (INT, Primary Key): Unique identifier for each society.
  name (VARCHAR): Name of the society (e.g., "ACM", "PROCOM").
  description (TEXT): Detailed description of the society's purpose, goals, and activities.
  created_at (TIMESTAMP): Timestamp of when the society was created.
  updated_at (TIMESTAMP): Timestamp of the last update to the society's information.

### 5.1.4 Events Table

- **Purpose:** Stores event details for each society. An event could be an induction, seminar, workshop, or any other activity hosted by a society.
- **Attributes:**
  event_id (INT, Primary Key): Unique identifier for each event.
  society_id (INT, Foreign Key to Societies): Links to the society hosting the event.
  name (VARCHAR): Name of the event (e.g., "Annual Tech Fest").
  date (DATE): Date of the event.
  time (TIME): Time of the event.
  location (VARCHAR): Location where the event is held.
  description (TEXT): Brief description of the event.
  created_at (TIMESTAMP): Timestamp when the event was created.

### 5.1.5 Memberships Table

- **Purpose:** Manages the relationship between students and the societies they are a part of.

- **Attributes:**
  membership_id (INT, Primary Key): Unique identifier for each membership.
  user_id (INT, Foreign Key to Users): Links to the user who is a member of the society.
  society_id (INT, Foreign Key to Societies): Links to the society to which the user is a member.
  role (VARCHAR): Role the member holds in the society (e.g., "President", "Member", "Treasurer").
  status (ENUM: 'active', 'inactive'): Status of the membership.
  joined_at (TIMESTAMP): Timestamp of when the student joined the society.
  updated_at (TIMESTAMP): Timestamp when the membership details were last updated.

### 5.1.6 Reports Table

- **Purpose:** Stores reports related to societies. These reports can include financial statements, activities, and performance summaries.

- **Attributes:**
  report_id (INT, Primary Key): Unique identifier for each report.
  society_id (INT, Foreign Key to Societies): Links to the society the report is related to.
  title (VARCHAR): Title of the report (e.g., "Financial Report", "Annual Activity Report").
  content (TEXT): Detailed content of the report.
  created_at (TIMESTAMP): Timestamp of when the report was generated.
  updated_at (TIMESTAMP): Timestamp of when the report was last updated.

### 5.1.7 Notifications Table

- **Purpose:** Keeps track of all notifications posted by the admin, such as event announcements, inductions, and other important updates.

- **Attributes:**
  notification_id (INT, Primary Key): Unique identifier for each notification.
  admin_id (INT, Foreign Key to Users): Links to the admin who posted the notification.
  message (TEXT): The content of the notification message.
  created_at (TIMESTAMP): Timestamp when the notification was posted.

### 5.1.8 Finance Table

- **Purpose:** Tracks the financial records of each society, including income, expenditures, and balance.

- **Attributes:**
  finance_id (INT, Primary Key): Unique identifier for each financial record.
  society_id (INT, Foreign Key to Societies): Links to the society the financial record belongs to.
  amount (DECIMAL): Amount of the transaction (could be income or expense).
  type (ENUM: 'income', 'expense'): Specifies whether the transaction is an income or expense.
  description (TEXT): Description of the transaction.
  date (DATE): Date of the transaction.
  created_at (TIMESTAMP): Timestamp when the financial record was created.

### 5.1.9 Attendance Table

- **Purpose:** Records the attendance of society members during events, meetings, or activities.

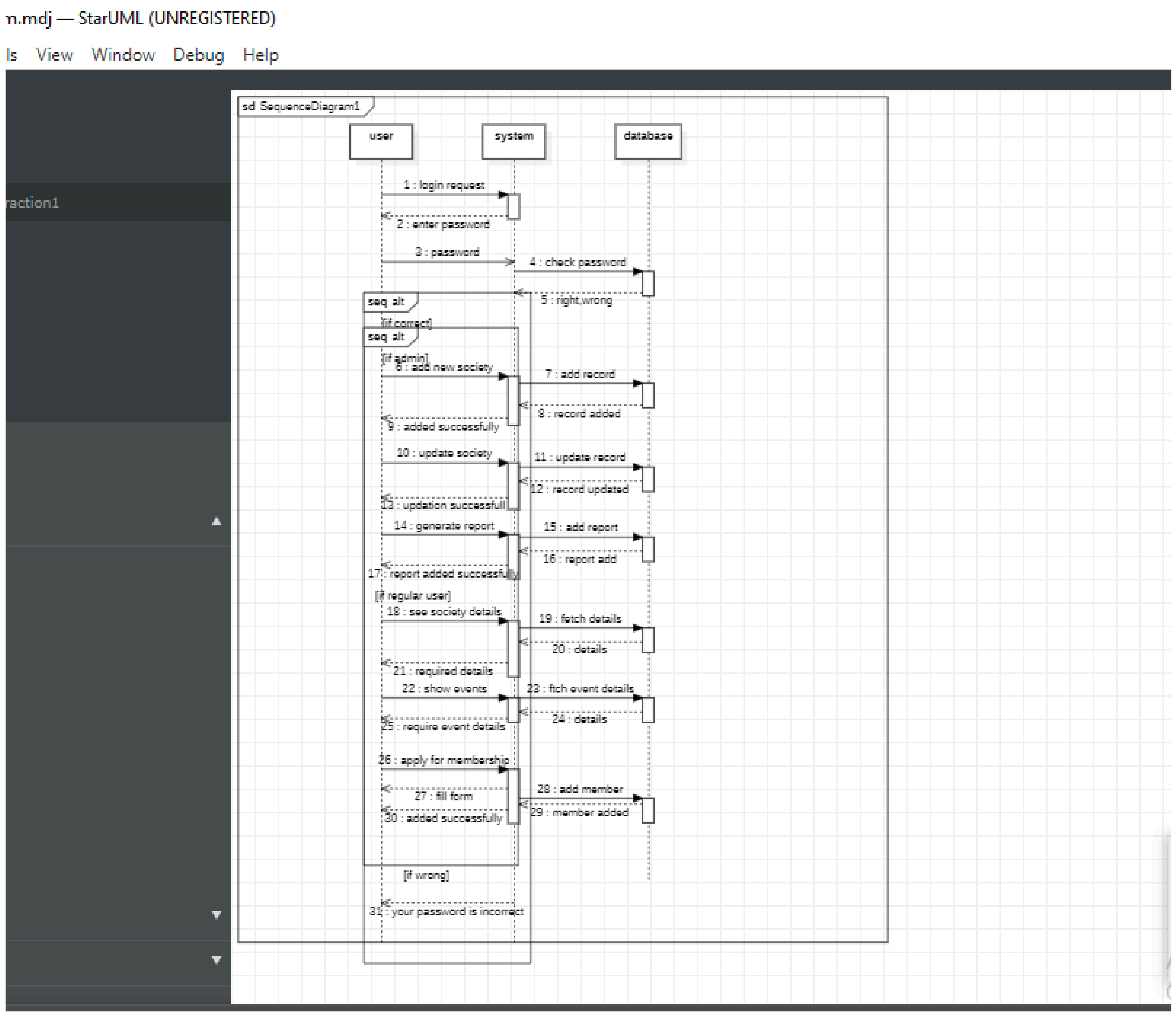
- **Attributes:**
  attendance_id (INT, Primary Key): Unique identifier for each attendance record.
  membership_id (INT, Foreign Key to Memberships): Links to the membership of the student attending the event.
  event_id (INT, Foreign Key to Events): Links to the event where the attendance is being tracked.
  status (ENUM: 'present', 'absent'): Attendance status of the student.
  timestamp (TIMESTAMP): Timestamp when the attendance was marked.

## 5.2 Application Design

The application design for the **University Society Management System** includes various diagrams to represent the structure, flow, and interactions of the system. These diagrams offer a detailed view of how the system works, including user actions, transitions, and data flow.

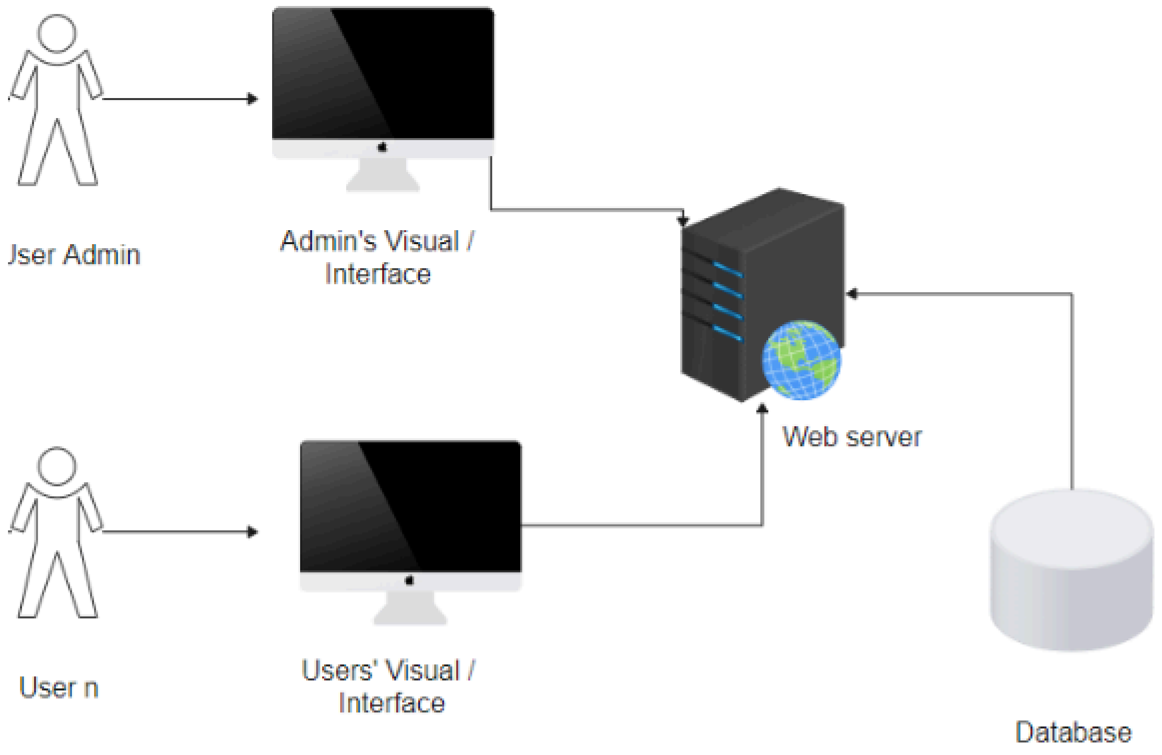
### 5.2.1 Sequence Diagram

The sequence diagram illustrates the interaction between the user and the system during various activities, such as user registration, login, and society management. It shows the flow of requests and responses between the frontend and backend with MySQL as the user interacts with the system.



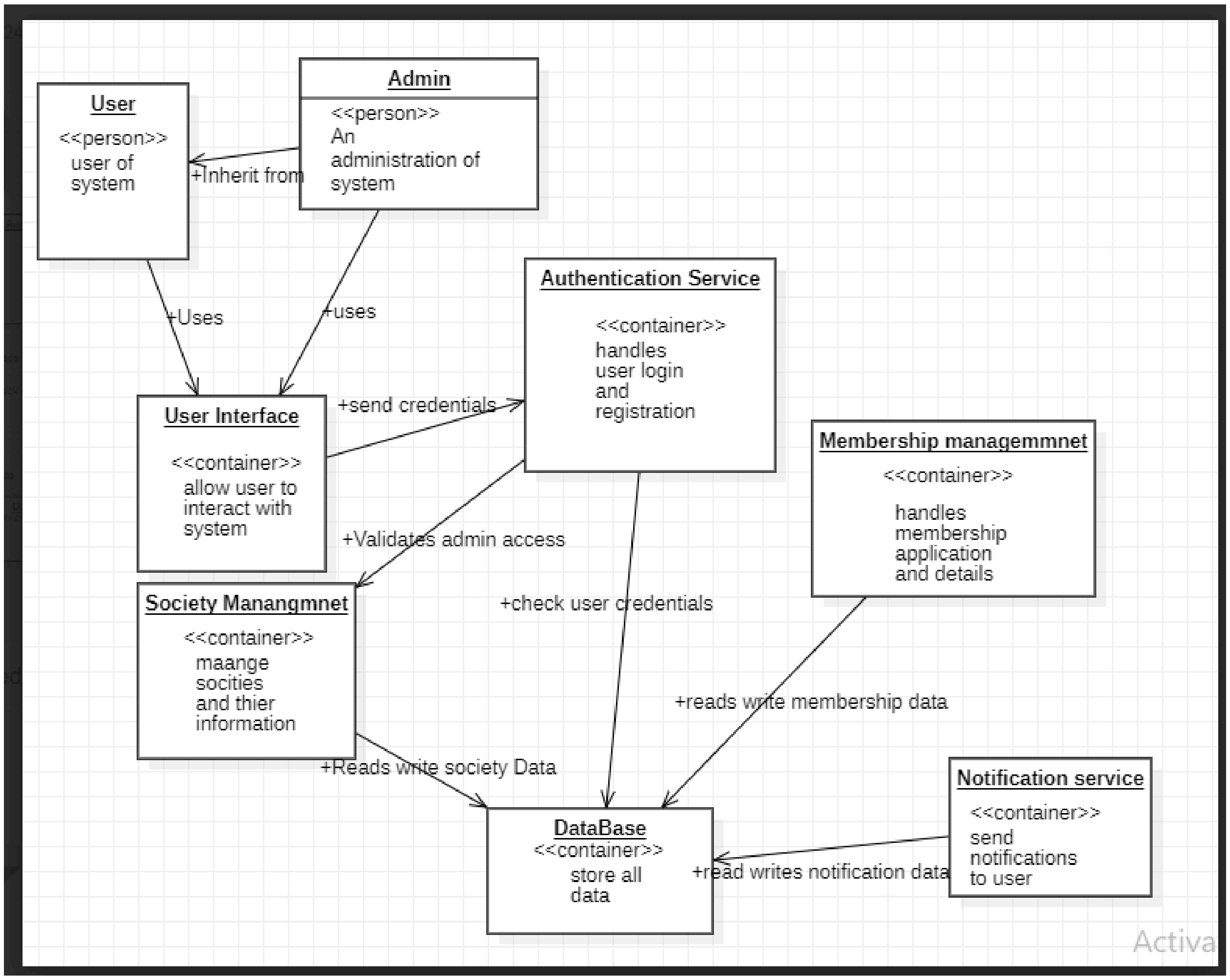
## 5.2.2 Main Architecture Diagram

The main architecture diagram depicts the high-level structure of the system, highlighting the interaction between different components like the frontend, backend, database, and the flow of data.



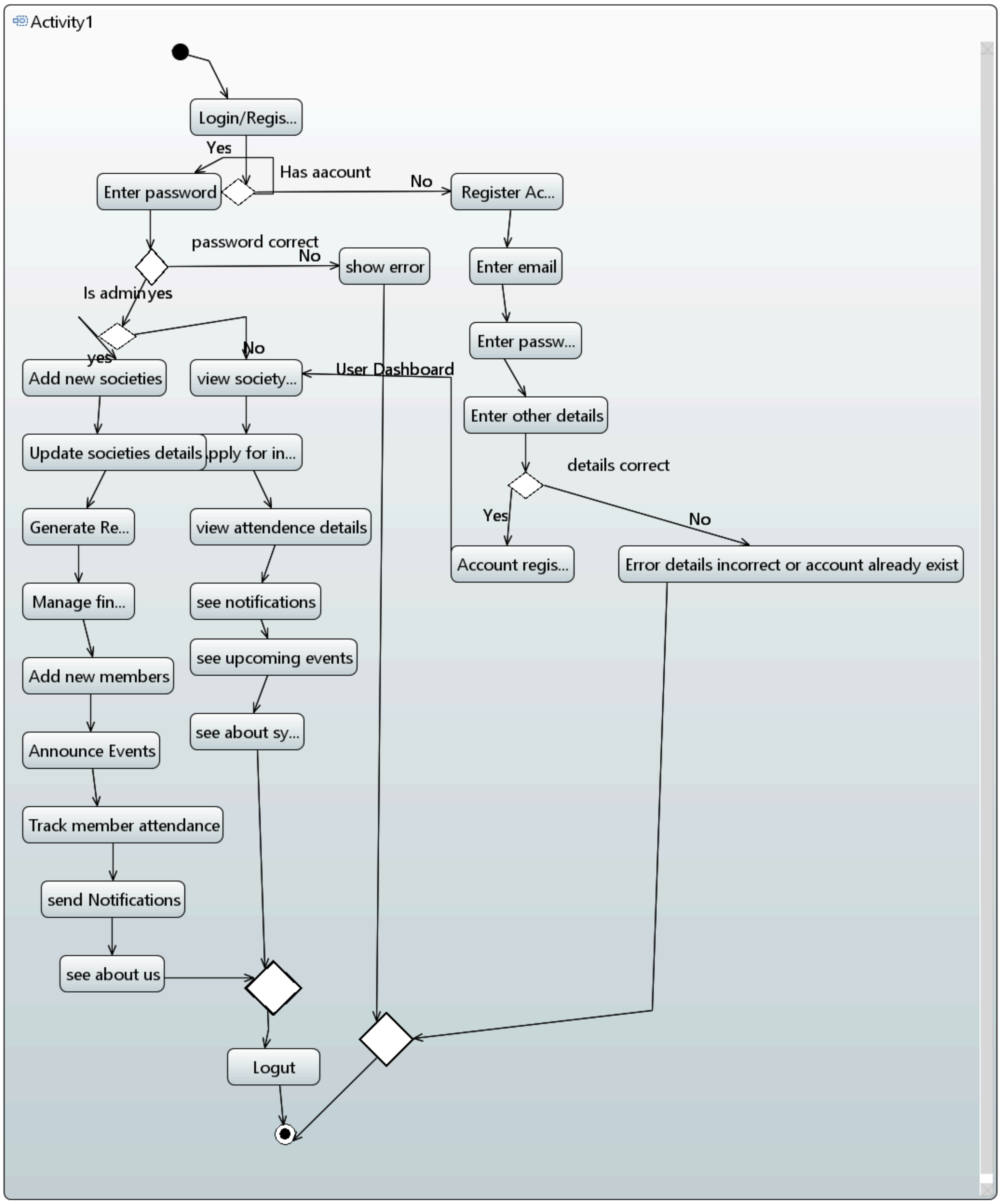
### 5.2.3 Object Diagram

The object diagram represents the static structure of the system, showcasing the instances of different classes and their relationships. For example, it could show instances of User, Society, and Event objects, illustrating how these objects interact and store data in the MySQL database**.**



### 5.2.4 Activity Diagram

The activity diagram outlines the dynamic workflow of a particular process, such as the user registration process or society creation. It shows the sequence of actions taken by the system, from the initial input (e.g., entering details) to the final output (e.g., user registration or society creation confirmation).


Activity1
Login/Regis...
Yes
Has aacount
No
Enter password
Register Ac...
password correct
No
show error
Enter email
Is admin
yes
Enter passw...
yes
No
User Dashboard
Add new societies
view society...
Enter other details
Update societies details
pply for in...
details correct
Yes
No
Generate Re...
view attendence details
Account regis...
Error details incorrect or account already exist
Manage fin...
see notifications
see upcoming events
Add new members
see about sy...
Announce Events
Track member attendance
send Notifications
see about us
Logut

## 6. References

**IEEE Standard for Software Engineering**
*Title:* IEEE 830-1998 - IEEE Recommended Practice for Software Requirements Specifications
*Date:* 1998
*Publishing Organization:* IEEE Computer Society
*URL:* https://ieeexplore.ieee.org/document/726752

**MySQL Workbench User Guide**
*Title:* MySQL Workbench 8.0 User Guide
*Date:* 2025
*Publishing Organization:* Oracle Corporation
*URL:* https://dev.mysql.com/doc/workbench/en/

**Material-UI Documentation**
*Title:* Material-UI: React components for faster and easier web development
*Date:* Ongoing (Last updated: 2025)
*Publishing Organization:* Material-UI
*URL:* https://mui.com/

**W3C HTML5 Specification**
*Title:* HTML5 - A vocabulary and associated APIs for HTML and XHTML
*Date:* 2025
*Publishing Organization:* W3C
*URL:* https://www.w3.org/TR/html5/

**CSS3 Documentation**
*Title:* CSS3 - Cascading Style Sheets Level 3
*Date:* 2025
*Publishing Organization:* W3C
*URL:* https://www.w3.org/TR/css3-roadmap/

## 7. Appendices

### Error Codes and Responses

The system provides structured error codes to handle failures and exceptions:

- **200 OK**: Request was successful.
- **400 Bad Request**: Invalid input or missing data.
- **401 Unauthorized**: User not authenticated or session expired.
- **404 Not Found**: Resource not found (e.g., society or event).
- **500 Internal Server Error**: Unexpected server error.

### UI Components

- **Login Form:** A simple form with fields for `username` and `password` to allow users to log in.
- **SignUp Form:** A form that allows new users to create an account by providing their username, password, and role (student or admin).
- **Society List:** Displays all available societies, with a link to view more details.
- **Society Details:** Provides a detailed view of the selected society, including upcoming events.

### Testing & Validation

Testing was performed at various stages of the development process to ensure the system's stability and functionality. Key aspects tested include:

- **Unit Testing:** Individual components and functions were tested to ensure they behave as expected.
- **Integration Testing:** The system's interaction between the backend and frontend was verified, ensuring data flows correctly between components.
- **User Acceptance Testing (UAT):** The system was tested by potential users to ensure it meets all functional requirements.

### Future Enhancements

- **Real-time Notifications:** Implement push notifications to inform users about new events or society updates.

- **Search Functionality:** Add search capabilities to allow users to find societies or events more easily.

- **Role-based Access Control:** Enhance user roles and permissions, allowing for more granular access to different parts of the system.

## THE END