Tab 1

Design Document

1. Introduction

1.1 Purpose of the Document

The purpose of this document is to provide a comprehensive technical blueprint for the development of the CampusHub - University Event Management System. This document details the system's architecture, component design, data models, and user interface specifications. It serves as the primary reference for the development team to ensure a consistent, scalable, and maintainable implementation that aligns with the project's functional and non-functional requirements.

1.2 Scope of the Design Phase

The scope of this design phase encompasses the complete structural and behavioral design of the CampusHub system. This includes:

System Architecture: Defining the high-level monolithic structure and the interaction between its major components.

Detailed Component Design: Specifying the classes, methods, and responsibilities within the Model-View-Controller (MVC) pattern.

User Interface (UI) Design: Creating wireframes and mockups for the primary user-facing screens.

Data Design: Outlining the database schema for data persistence and retrieval.

1.3 Intended Audience

This document is intended for the following stakeholders:

The Development Team: As a guide for implementation and to ensure architectural consistency.

The Project Instructor (Dr. Mohamed Sami Rakha): For evaluation and to verify that design requirements have been met.

Quality Assurance (QA) Personnel: To understand the system's inner workings for effective testing.

Future Developers and Maintainers: To aid in understanding, modifying, and extending the system post-delivery.

1.4 Overview of the Contents

This document is organized into five main sections. Section 1 provides an introduction. Section 2 gives a high-level overview of the system and its functions. Section 3 details the chosen system architecture and technology stack. Section 4 presents the detailed design, including the MVC pattern implementation, UML diagrams, UI mockups, and data schema. Finally, Section 5 summarizes the key design decisions.

2. System Overview

2.1 Brief Description of the System

The CampusHub is a comprehensive web-based event management system designed for university environments. The system serves three primary user roles: Students, Instructors, and Administrators. It facilitates the entire lifecycle of university events, from creation and promotion by instructors to registration and attendance tracking by students. Key features include a secure authentication system, robust event management with search and filtering capabilities, a registration system with capacity controls, an announcement module for communication, and an administrative dashboard for user management and system analytics. The primary goal is to streamline event

organization, enhance student engagement, and provide administrators with powerful tools for oversight and reporting.

2.2 Key Design Goals and Constraints

Design Goals:

Modularity: The system will be built using the MVC pattern to separate concerns, making the codebase easier to manage, test, and scale.

Usability: The user interface will be clean, intuitive, and responsive, ensuring a positive user experience for all user roles across different devices.

Security: User data will be protected through secure authentication mechanisms, password hashing, input validation, and session management to prevent common vulnerabilities.

Performance: The system will aim for a page load time of under 3 seconds and will efficiently handle database queries to ensure responsiveness, especially during peak registration times.

Constraints:

Technology Stack: The project must be developed using the agreed-upon technology stack.

Development Time: The entire project, including design, implementation, and testing, must be completed within the Fall-2025 semester timeline.

Monolithic Architecture: The system must be deployed as a single, self-contained unit, as per the project requirements.

3. Architectural Design

3.1 System Architecture Diagram

3.2 Discussion of Architectural Style and Components

Architectural Style: Monolithic Architecture

Our system employs a Monolithic Architecture, where all the application's logic (presentation, business, and data access) is combined into a single, deployable unit. This style was chosen for its simplicity and suitability for a project of this scope, being developed by a single team. It simplifies development, debugging, and deployment processes compared to a distributed architecture like microservices.

Key Components:

Web Server: Acts as the entry point for all user requests (HTTP/HTTPS). It handles static files (CSS, JS, images) and forwards dynamic requests to the Application Server.

Application Server (MVC Core): This is the heart of the system. It contains the entire application logic, structured into the Model, View, and Controller layers.

Database: This layer is responsible for the persistent storage of all application data, such as user profiles, event details, registration records, and **announcements. We will use a relational database like MySQL.**

3.3 Technology Stack and Tools

Backend:

Language: [e.g., Python 3.9]

Framework: [e.g., Flask 2.0]

Database: [e.g., MySQL 8.0]

ORM/Database Connector: [e.g., SQLAlchemy]

Frontend:

Markup: HTML5

Styling: CSS3 with [e.g., Bootstrap 5]

Scripting: JavaScript (ES6+)

Development & Deployment Tools:

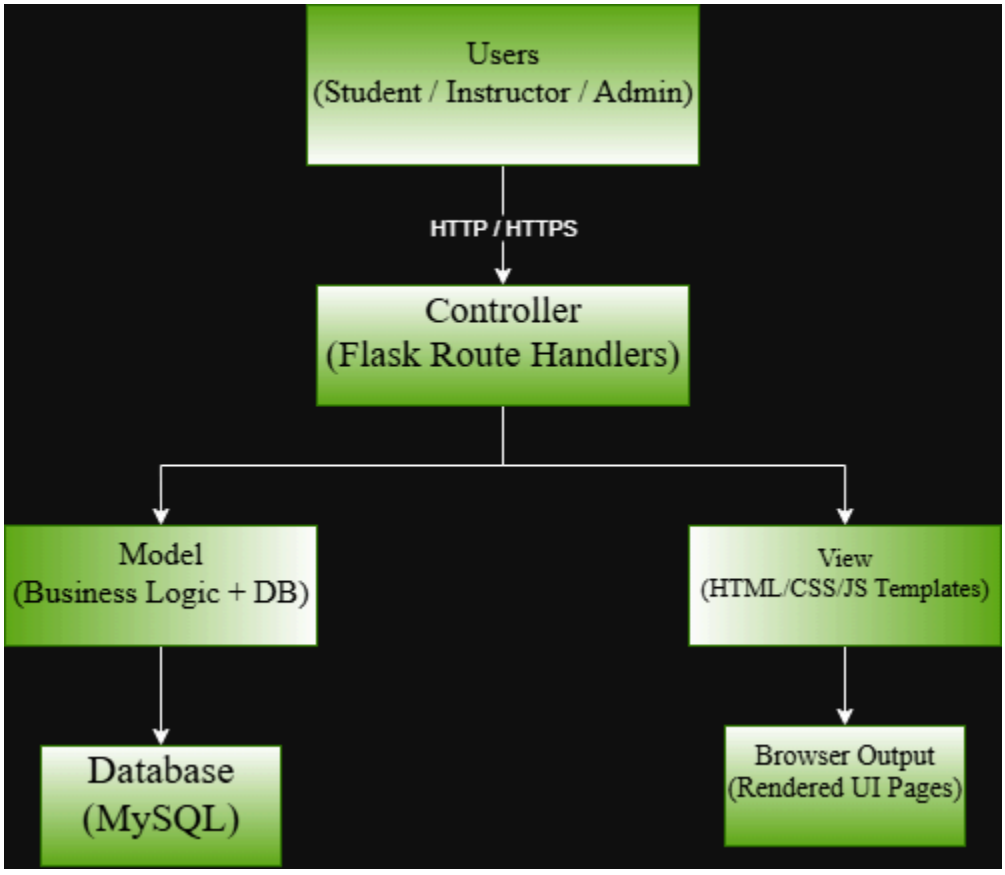
Version Control: Git & GitHub

IDE: [e.g., Visual Studio Code]

Testing: [e.g., PyTest for backend, Jest for frontend]

4. Detailed Design

4.1 Model–View–Controller (MVC) Design Pattern



4.1.1 Description of MVC Pattern

The Model-View-Controller (MVC) pattern is a software architectural pattern that separates an application into three interconnected components: the Model, the View, and the Controller. This separation of concerns is crucial for building scalable and maintainable applications. The Model handles the data and business logic, the View presents the data to the user, and the Controller acts as an intermediary to process user input and coordinate the Model and View.

4.1.3 Responsibilities of Model, View, and Controller

Model Responsibilities:

Manage application state and data (e.g., user profiles, event details, registration records).

Implement business logic and data validation rules.

Handle all database interactions (CRUD operations: Create, Read, Update, Delete).

Communicate with the database for login verification, event creation, registration management, etc.

View Responsibilities:

Generate the user interface (UI) based on the data provided by the Controller.

Display data to the user in a readable format (e.g., event lists, registration forms, dashboards).

Send user actions (e.g., button clicks, form submissions) to the Controller.

Contain no business logic.

Controller Responsibilities:

Receive and interpret user input from the View (e.g., login credentials, event registration form).

Interact with the Model to perform operations (e.g., UserModel.verifyCredentials(), EventModel.createEvent()).

Select the appropriate View to render the response (e.g., redirect to Dashboard on success, show error on failure).

Act as an intermediary between the Model and the View, ensuring they remain decoupled.

4.1.4 Interaction Between Components

The typical flow of interaction is illustrated by the User Login Sequence:

The User submits login credentials through the View (Login Page).

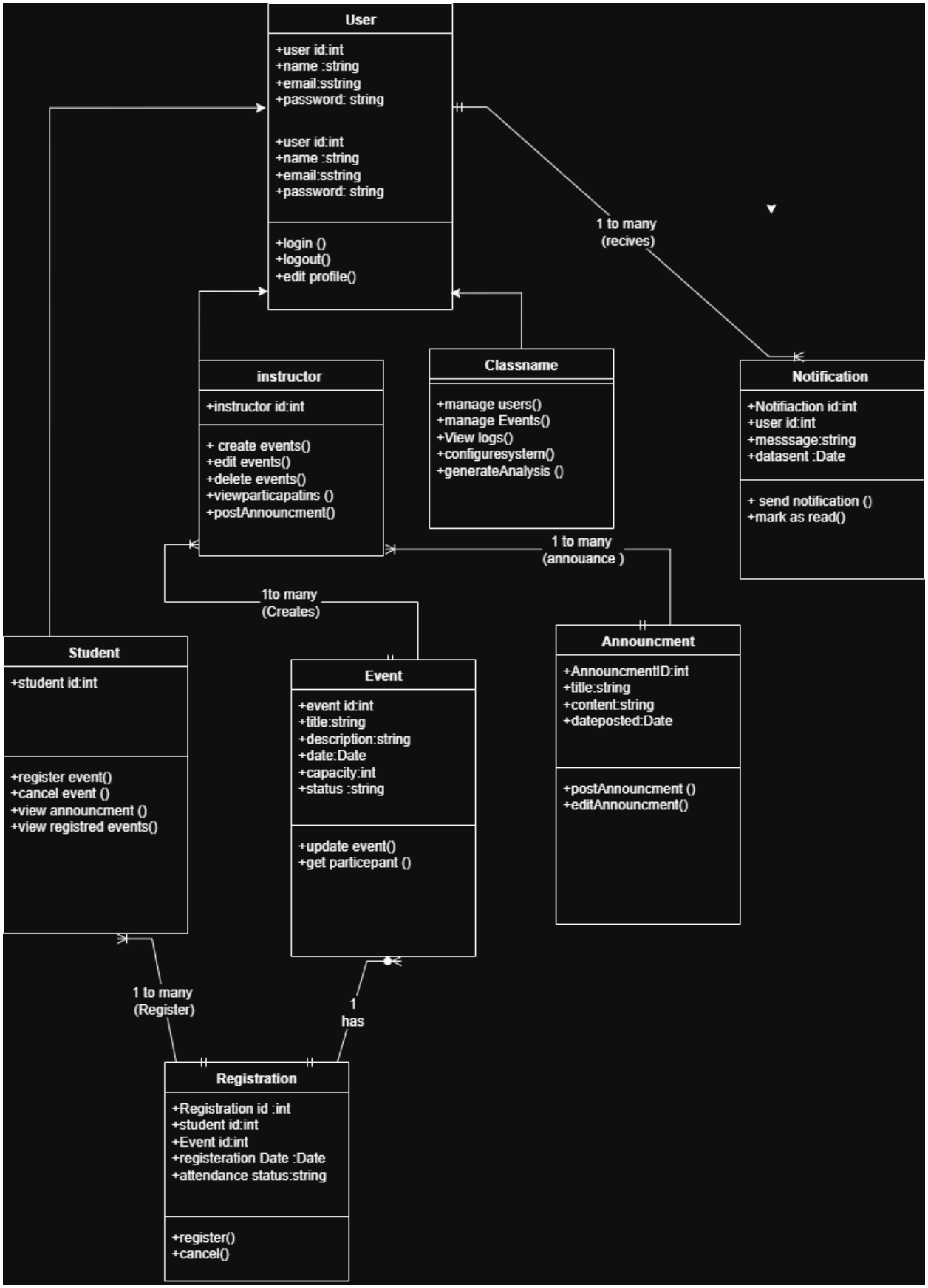
The Controller (AuthController) receives the request, validates the input, and calls the Model (UserModel) to check the database for user credentials.

The Model (UserModel) interacts with the Database to verify credentials.

Based on the response, the Controller either redirects the user to the Dashboard View (on success) or shows an error message in the same View (on failure).

4.2 UML Diagrams

4.2.2 Detailed Class Diagram


User
+user id:int
+name :string
+email:sstring
+password: string
+user id:int
+name :string
+email:sstring
+password: string
+login ()
+logout()
+edit profile()
1 to many
(recives)
instructor
+instructor id:int
+ create events()
+edit events()
+delete events()
+viewparticapatins ()
+postAnnouncment()
Classname
+manage users()
+manage Events()
+View logs()
+configuresystem()
+generateAnalysis ()
Notification
+Notifiaction id:int
+user id:int
+messsage:string
+datasent :Date
+ send notification ()
+mark as read()
1 to many
(annouance)
1to many
(Creates)
Student
+student id:int
+register event()
+cancel event ()
+view announcment ()
+view registred events()
Event
+event id:int
+title:string
+description:string
+date:Date
+capacity:int
+status :string
+update event()
+get particepant ()
Announcment
+AnnouncmentID:int
+title:string
+content:string
+dateposted:Date
+postAnnouncment ()
+editAnnouncment()
1 to many
(Register)
1
has
Registration
+Registration id :int
+student id:int
+Event id:int
+registeration Date :Date
+attendance status:string
+register()
+cancel()

The class diagram is structured around a core User hierarchy and event management entities.

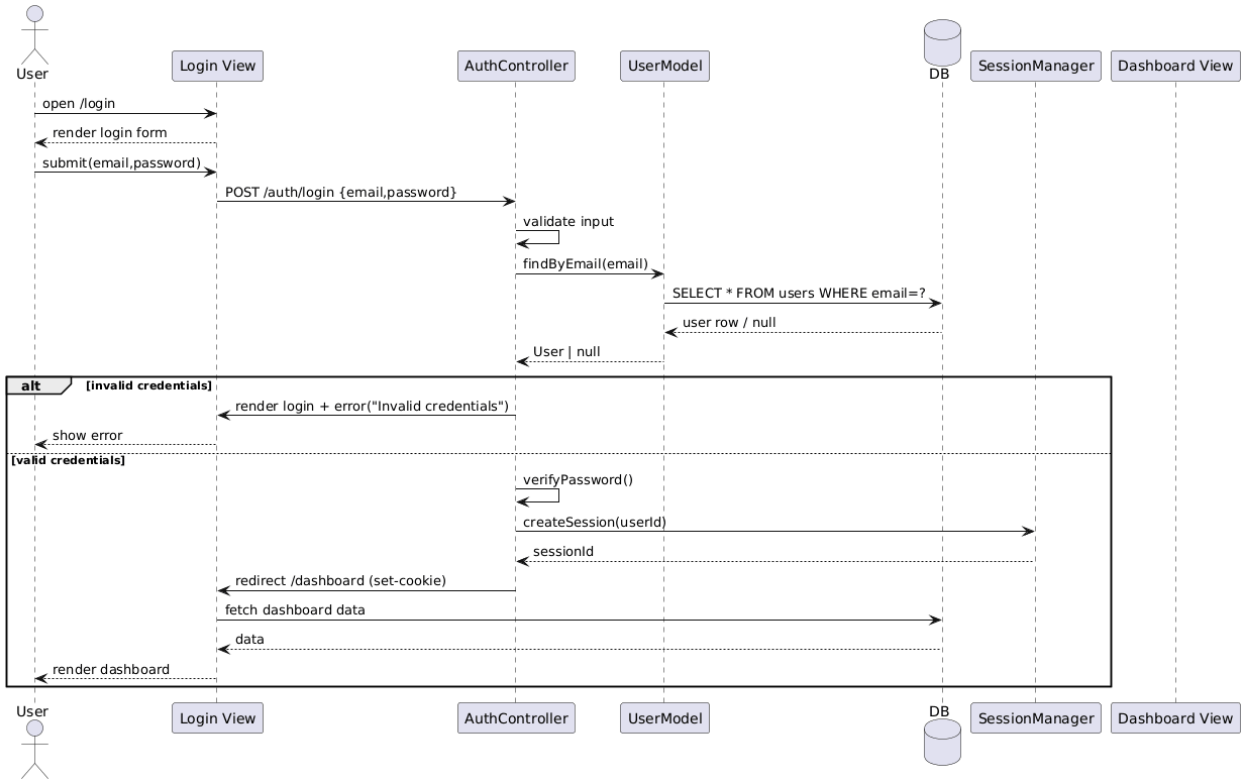
User Management: A base User class contains common attributes (userID, email, passwordHash) and methods (login(), logout(), editProfile()). Three classes inherit from User: Student, Instructor, and Admin, each with role-specific methods (e.g., Student.registerForEvent(), Instructor.createEvent(), Admin.manageUsers()).

Event Management: The Event class represents events with attributes like eventID, title, date, and capacity. The Registration class acts as a bridge to resolve the many-to-many relationship between Student and Event, with a composite key (studentID, eventID).

Communication: The Announcement class is used by Instructor to post messages, linked by a one-to-many relationship. The Notification class handles system-generated alerts for users.

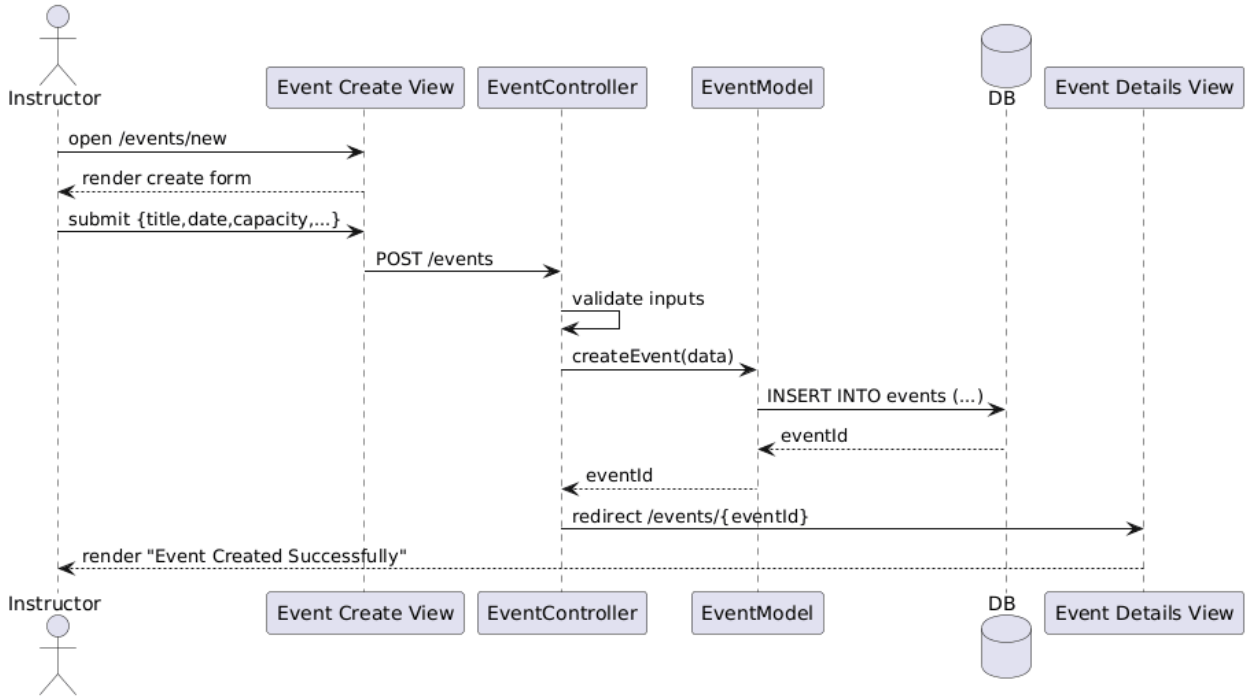
4.2.3 Sequence Diagrams

[Figure 3: Sequence Diagram for User Login (FR2)]



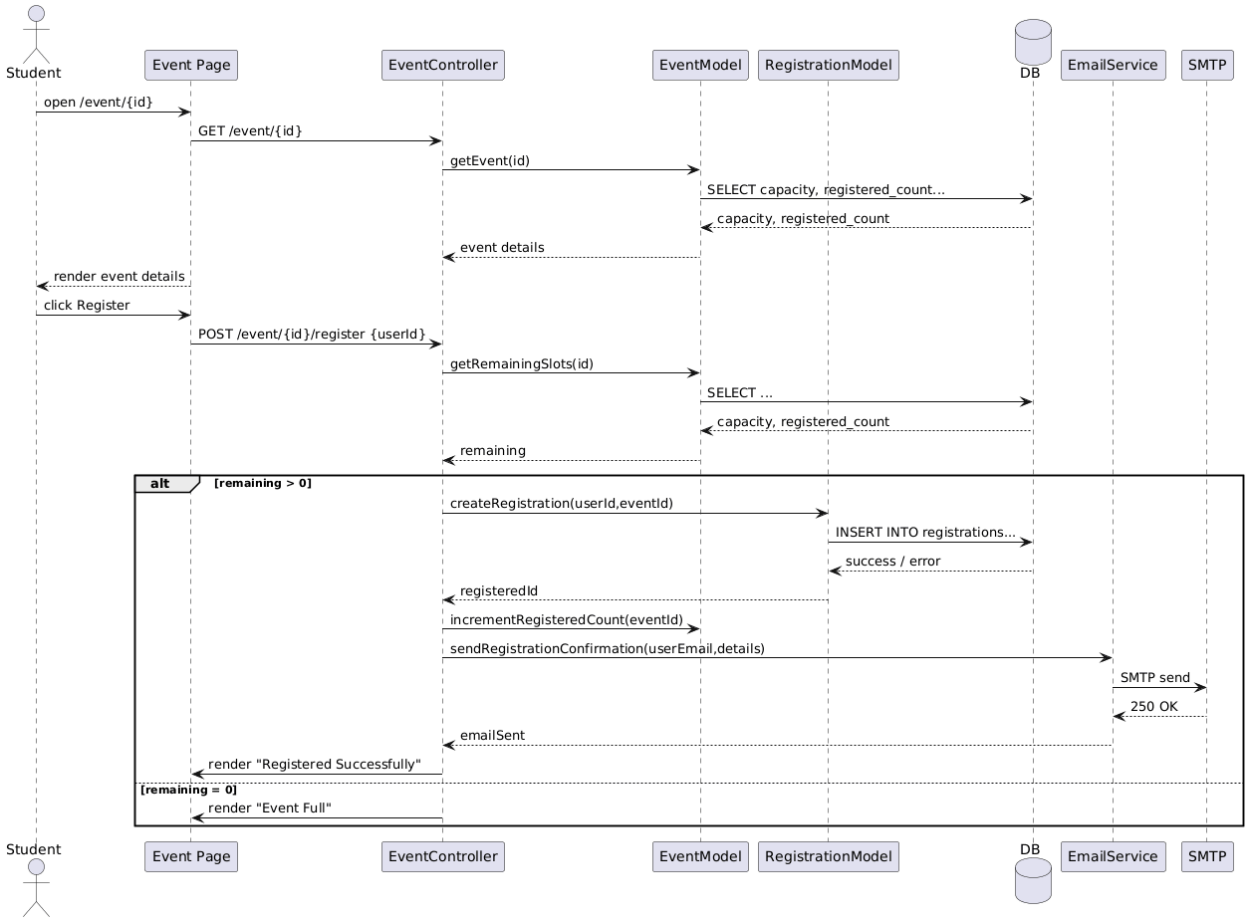
This diagram illustrates the user authentication process. The Login View sends user credentials to the AuthController. The controller delegates verification to the UserModel, which queries the DB. Based on the result, the controller either creates a session via SessionManager and redirects to the Dashboard View (valid credentials) or re-renders the Login View with an error message (invalid credentials).

[Figure 4: Sequence Diagram for Instructor Creates Event (FR3)]



This diagram shows the event creation workflow. The Instructor submits a form via the Event Create View to the EventController. The controller validates the input and calls the EventModel to insert the new event into the DB. Upon success, the controller redirects the instructor to the Event Details View with a success message.

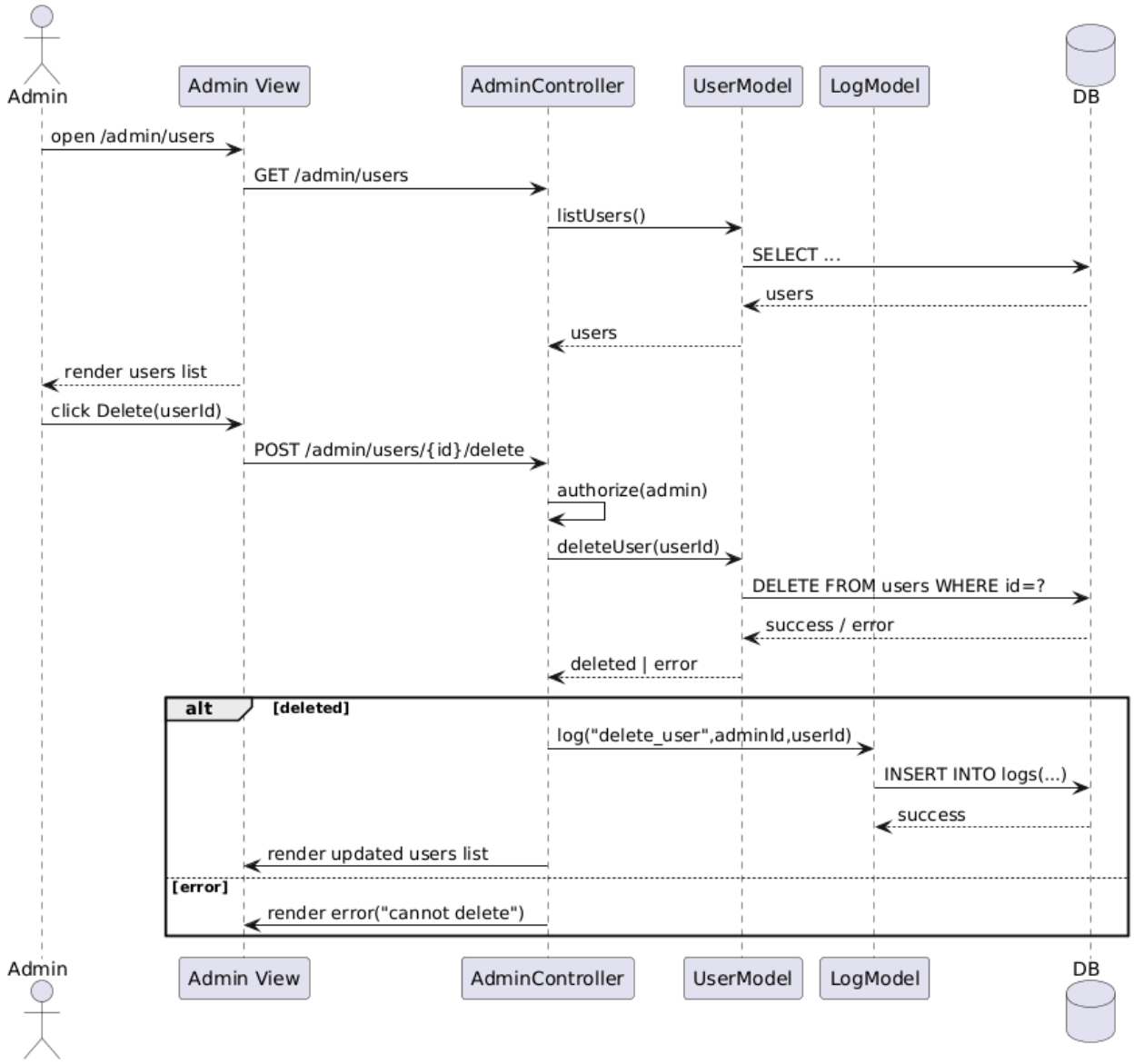
[Figure 5: Sequence Diagram for Student Registers for Event (FR4)]



This diagram details the event registration process. The Student initiates registration from the Event Page, which sends a request to the EventController. The controller uses the EventModel to check event capacity. If space is available, it calls the RegistrationModel to create a record and updates the event count. It then triggers the EmailService to send a confirmation via SMTP and renders a success message. If the event is full, it displays an "Event Full" message.

[Figure 6: Sequence Diagram for Admin Deletes User (FR9)]

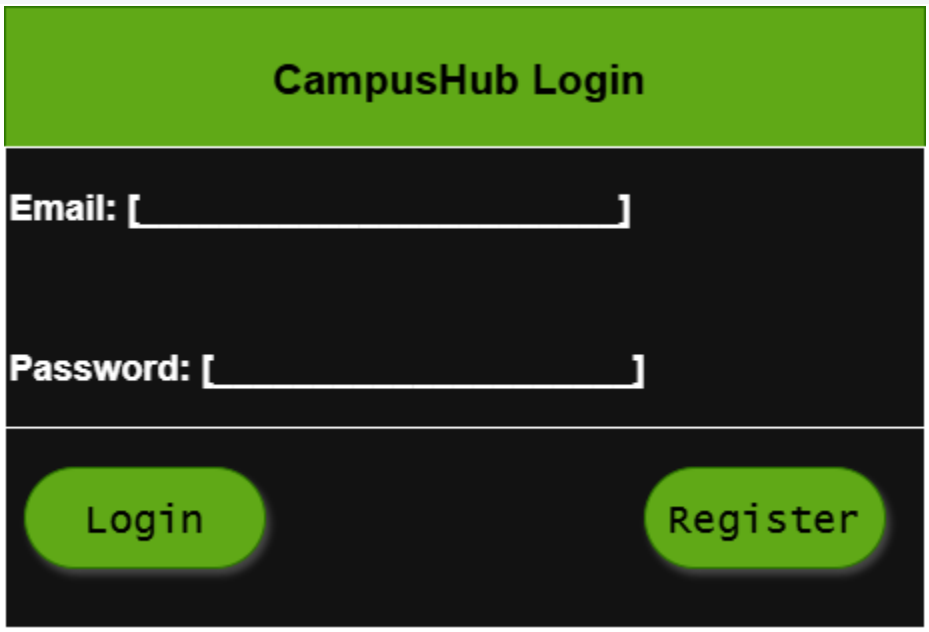
This dia

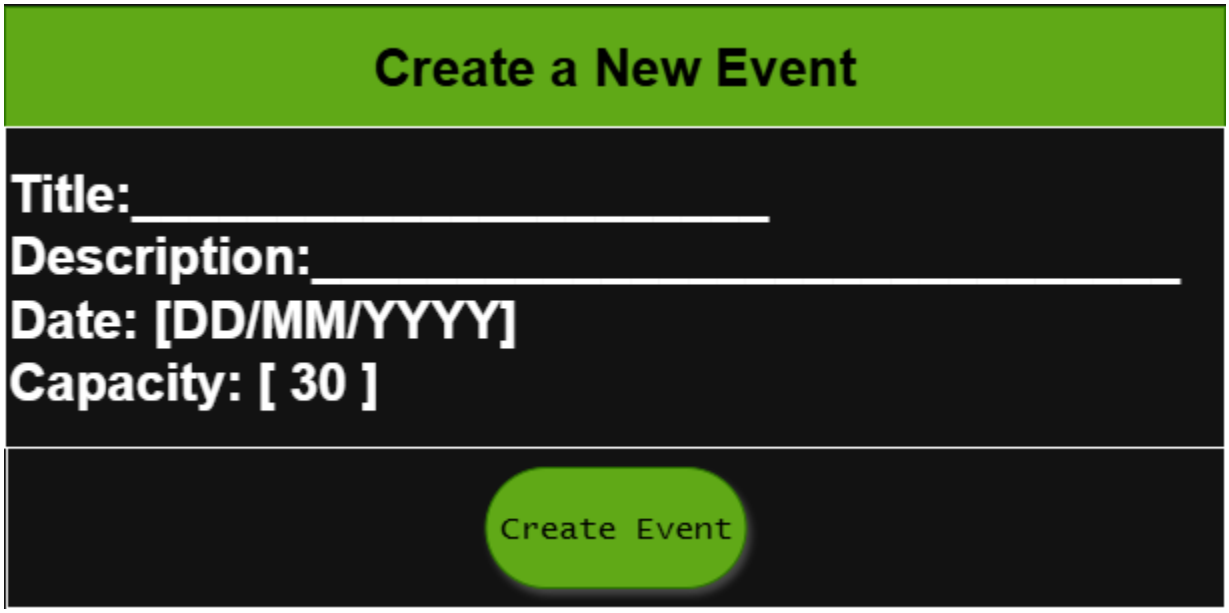


gram outlines the user deletion process by an admin. The Admin View sends a POST request to the AdminController. After authorization, the controller calls the UserModel to execute a DELETE operation on the DB. On success, the LogModel is invoked to record the action, and the view is updated. If the deletion fails, an error message is displayed.

4.3 UI/UX Design

4.3.1 Wireframes / Mockups





Event: AI Workshop

. Date: 25/11/2025
. Capacity: 30 Students
. Description: Introduction to AI concepts

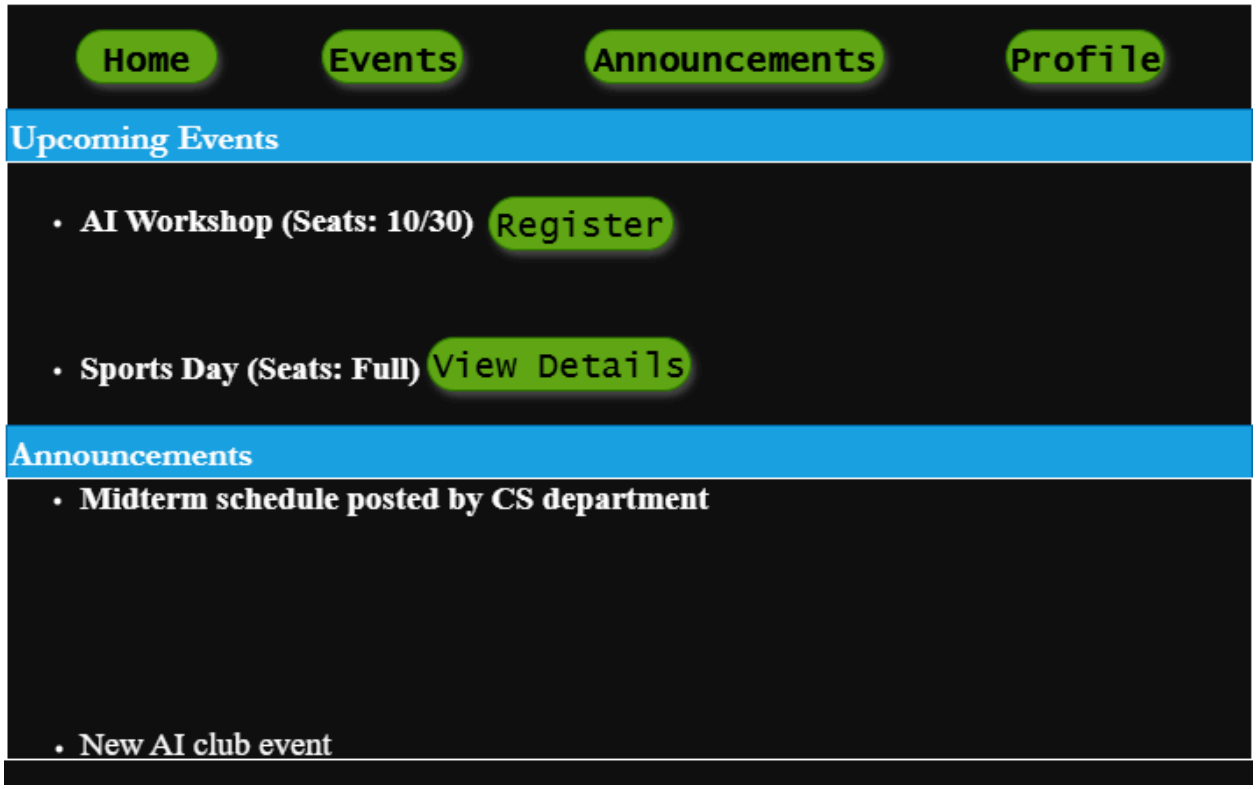
Register Cancel Registration

Admin Dashboard

Manage Users Manage Events View Logs

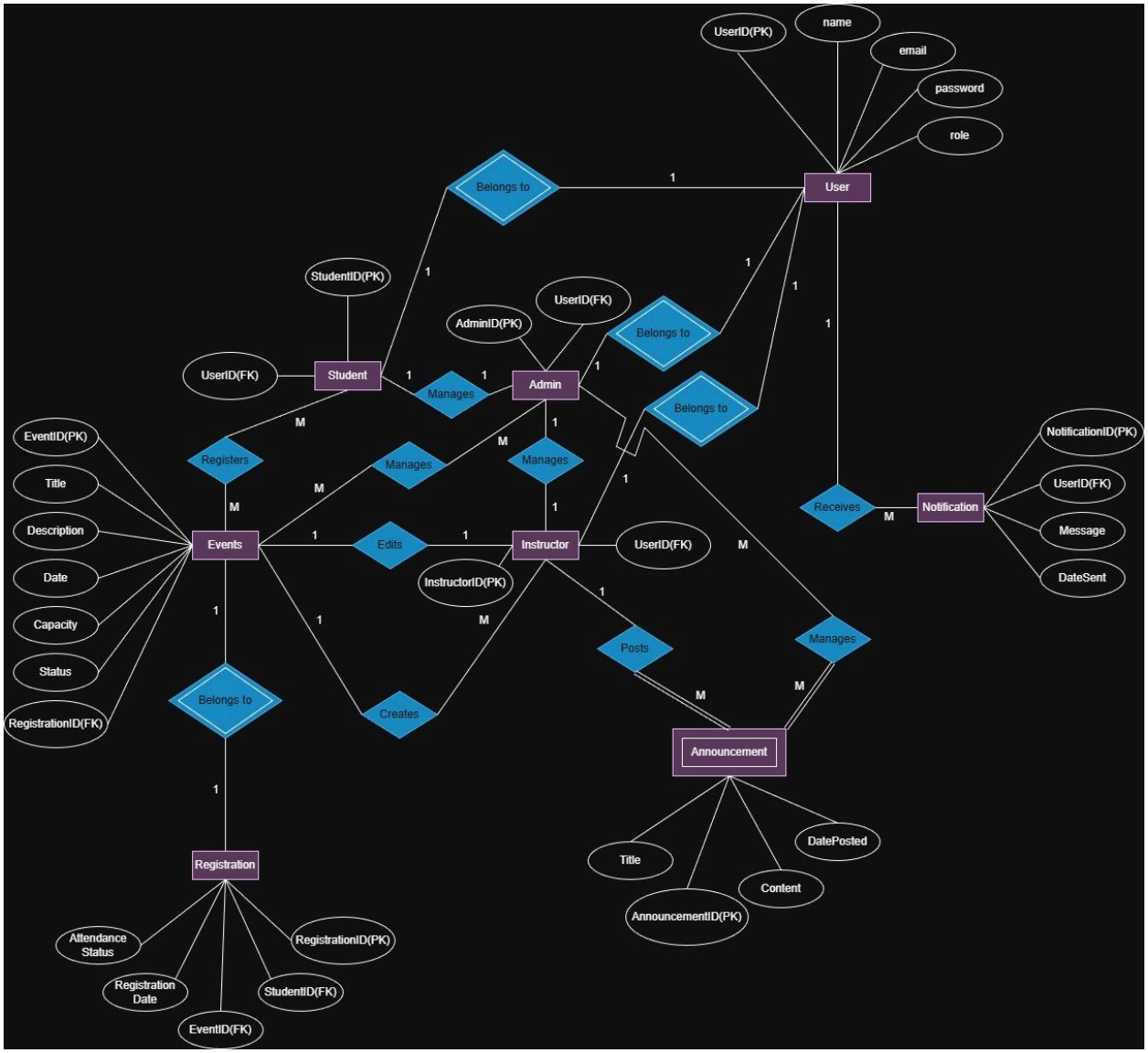
User Activity Logs:

1. **Student Ahmed registered for AI Workshop**
2. **Instructor Omar created "Data Science Meetup"**



4.4 ERD Design

4.4.1 ERD



4.4.2 Data Dictionary

Data Dictionary

Table: users

Column Name	Data Type	Constraints	Description

user_id	INT	Primary Key, Auto Increment	Unique identifier for each user.
email	VARCHAR(255)	Not Null, Unique	University email address.
password_hash	VARCHAR(255)	Not Null	Hashed and salted password.
role	ENUM('student','instructor','admin')	Not Null	Defines user permissions.
name	VARCHAR(200)	Not Null	User's name.

Table: events

Column Name	Data Type	Constraints	Description
event_id	INT	Primary Key, Auto Increment	Unique identifier for each event.
registration_id	INT	Foreign Key (Registration registration_id)	ID of the registration entries.
title	VARCHAR(255)	Not Null	Title of the event.
description	TEXT		Event description.
date	DATETIME	Not Null	Event date and time.
capacity	INT(10)		Maximum attendees.

Table: registrations

Column Name	Data Type	Constraints	Description
registration_id	INT	Primary Key, Auto Increment	Unique registration ID.
event_id	INT	Foreign Key (events.event_id)	Event the student registered for.
student_id	INT	Foreign Key (users.user_id)	Student who registered.
Registration date	TIMESTAMP	Default CURRENT_TIMESTAMP	Registration timestamp.
Attendance Status	VARCHAR(30)		Lists if the student attended or not.

Table: announcements

Column Name	Data Type	Constraints	Description
announcement_id	INT	Primary Key, Auto Increment	Unique announcement ID.
title	VARCHAR(255)	Not Null	Announcement title.
content	TEXT	Not Null	Announcement body.
date_posted	TIMESTAMP	Default CURRENT_TIMESTAMP	Posting date.

Table: Notification

Column Name	Data Type	Constraints	Description
notification_id	INT	Primary Key, Auto Increment	Unique notificationID.

user_id	INT	Foreign Key (users.user_id)	User who is notified.
message	TEXT	Not Null	Notification body.
datesent	TIMESTAMP	Default CURRENT_TIMESTAMP	Sending date.

Table: Student

Column Name	Data Type	Constraints	Description
student_id	INT	Primary Key, Auto Increment	Unique StudentID.
user_id	INT	Foreign Key (users.user_id)	User_id of student.

Table: Admin

Column Name	Data Type	Constraints	Description
admin_id	INT	Primary Key, Auto Increment	Unique AdminID.
user_id	INT	Foreign Key (users.user_id)	User_id of admin.

Table: Instructor

Column Name	Data Type	Constraints	Description
instructor_id	INT	Primary Key, Auto Increment	Unique instructorID.
user_id	INT	Foreign Key (users.user_id)	User_id of instructor.

4.4.3 Requirement-to-design mapping

No.	Function	Description
1.	Authenticate user	User login using university email address and password
2.	Manage users	Add, remove, or update users
3.	Manage events	Create, cancel, or update events
4.	Manage registration	Students register for events
5.	Manage attendance	Recording of student attendance at events
6.	Manage announcements	Instructors post, delete, or update announcements
7.	Receive notifications	Users receive notifications through email (optional)
8.	Generate statistics	Create attendance and engagement statistics
9.	Search events	Users search for events using filters or name
10.	Manage Profile	Users update personal info, password, and preferences
11.	Log activity	Admins see logged user and system activity
12.	Validate input	Make sure users input valid and safe for work (SFW) text
13.	Encrypt passwords	Passwords are hashed and stored
14.	Handle sessions	Sessions are created upon login and terminated upon inactivity for a certain period

5. Conclusion

5.1 Summary of Design Phase

The design phase for the CampusHub - University Event Management System has established a clear and comprehensive plan for implementation. The adoption of a Monolithic architecture, structured with the Model-View-Controller pattern, provides a solid foundation for modularity and maintainability. The detailed UML diagrams, UI wireframes, and data dictionary ensure that all team members have a shared understanding of the system's structure and behavior. This design will guide the development process to build a robust, scalable, and user-friendly application that meets all specified functional and non-functional requirements.

Tab 2

1.The Class Digaram Description

1.the class diagram overview
the system Class Diagram Description Document

CampusHub University Event Management System

Document Overview

· Project: CampusHub - University Activity & Event Management System

1. CLASS DIAGRAM OVERVIEW

1/ System Architecture

The Class Diagram follows a modular, object-oriented design implementing the MVC architectural pattern. It establishes clear relationships between users, events, and communication entities while maintaining separation of concerns.

2/ Design Principles Applied

1· Inheritance for code reusability
2 Encapsulation for data security
3· Association for entity relationships
4· Modularity for maintainability
5. Scalability for future expansion

2. CORE CLASS DESCRIPTIONS

3/ User Management Hierarchy

Class: User (Base Class)

Purpose: Foundation class containing common attributes and methods for all system users

Methods Specification:

· login(credentials) → Authenticates user and creates session
· logout() → Terminates user session securely

· editProfile(newData) → Updates user information with validation

Class: Student (Inherits from User)

Purpose: Represents student users who participate in events

Methods:

· createEvent(eventDetails) → Creates new event with validation
· editEvent(EventID, updates) → Modifies existing event
· deleteEvent(EventID) → Removes event from system
· viewParticipants(EventID) → Displays registered students
· postAnnouncement(content) → Publishes announcements to students

Class: Admin (Inherits from User)

Purpose: System administrators with full management privileges

Methods:

1 manageUsers(action, userData) → CRUD operations on user accounts
2 manageEvents(action, eventData) → Oversees all system events
1 viewLogs(filters) → Monitors system activities and access
2 configureSystem(settings) → Adjusts system parameters
3 generateAnalytics(parameters) → Creates participation reports

3/Event Management System

Class: Event

Purpose: Core entity representing university events and activities

· update Event(updates) → Modifies event details
· get Participants() → Returns list of registered students

Class: Registration

Purpose: Bridge entity managing student-event relationships

Methods:

· register() → Creates registration record
· cancel() → Removes registration record

5/ Communication System

Class: Announcement

Purpose: Manages official communications from instructors

Attributes:
Annnouncment ID :int
1. Title : string
2. Content:text
3. Date posted:Data Time
4. Posted by:int
5. Target audience :string

Methods:

1· post Announcement () → Publishes to targeted students
2· edit Announcement(updates) → Modifies announcement content

Class: Notification

Purpose: Handles system-generated alerts and messages

Attributes:
1. Notification ID:int
2. User ID:int
3. Message : string
4. Datesent :datetime
5. Is Read :boollean
6. type :string

Methods:

· send Notification() → Delivers to user's dashboard/email
· mark As Read() → Updates read status

3/ RELATIONSHIPS DETAILS

6. Inheritance Relationships

1. User (Base Class)
2. Student (Specialization)
3. Instructor (Specialization)
4. Admin (Specialization)

Implementation Strategy:

1 Table per concrete class with type discriminator
2 Shared attributes in User table
3 Role-specific attributes in subclass tables

7. Association Relationships

One-to-Many Relationships:

1. Instructor (1) → Events (*)
   · An instructor creates multiple events
   · Each event has one creator
2. Instructor (1) → Announcements (*)
   · An instructor posts multiple announcements
   · Each announcement has one author
3. Event (1) → Registrations (*)
   · An event contains multiple registrations
   · Each registration belongs to one event
4. User (1) → Notifications (*)
   · A user receives multiple notifications
   · Each notification targets one user

Many-to-Many Resolution:

Student (*) ←→ Registration ←→ (*) Event

1· Registration class resolves M:N relationship
2· Composite key: (Student ID + Event ID)

3· Ensures data integrity and prevents duplicate registrations

Tab 4

Tab 3

Tab 5