CampusHub
University Activity & Event Management System

▪ Introduction

1. Purpose:

- This document lays out the software needs for CampusHub. It is a system meant to handle university events and activities. The project comes from the CSAI203 course.
- The main goal here is to spell out the functional and non-functional requirements in detail. These will help shape the design work, the actual building of the system, and the testing that follows.
- CampusHub works to make things easier for students, instructors, and administrators. They can manage events, post updates, and keep track of university announcements and communications.

2. Scope:

- CampusHub serves as a web-based application. It relies on Python Flask for the backend. The frontend uses HTML, CSS, and JavaScript.
- This setup offers a central platform for various users. Students can register for events there. They receive announcements and communicate with instructors.
- Instructors handle creating events. They also manage announcements and keep track of them.
- Administrators take care of user management. They monitor overall activity. They make sure the system stays secure and intact.
- The application sticks to event and activity management. It leaves out academic grading entirely. Payments do not fall under its scope. Library operations stay outside the project too.

3. Definitions, Acronyms, and Abbreviations:

Acronym	**Definition**
SRS	Software Requirements Specification
UI	User Interface
UX	User Experience
DB	Database
CRUD	Create, Read, Update, Delete
Flask	Python web framework used for backend development
SQL	Structured Query Language
MVC	Model-View-Controller (software pattern)

4. Overview:

- The CampusHub System requirements are described in this document along with the system's goals, general overview, particular functional and non-functional requirements, and external interface specifications. Use cases and domain structure are shown through the use of diagrams.

Overall Description

1. Product Perspective:

- The web-based platform used by the university to manage announcements, events, and activities is called CampusHub. The system uses a database connection, a login process for validation, and SMTP for email sending.

2. Product Functions:

- User registration and authentication
- Access-based roles (students, instructors, and administrators)
- Event Management (creation, edits)
- Event Registration/Cancellation
- Email-based notifications system
- Event Search capabilities
- Account Manager
- Admin control over events

3. User classes:

- Students (the main users) - they will have access to different functionalities like browsing, signing up for events, viewing events, announcements, and profiles, and will also get notifications.
- Instructors (the next in line) - will be able to carry out the following tasks like creating, modifying, or removing events, posting notices, and monitoring attendees.
- Admins (the last but not the least) - will be responsible for the overseeing of the users, instructors, and events (cancellation of events).

4. Operating Environment:

- Web browsers: Chrome, Firefox, and Edge
- Server: Windows 11
- Database: MySQL

5. Constraints:

- Must be implemented using Flask, HTML, CSS, and JavaScript.
- Local deployment is a must.
- MVC architecture must be implemented.

6. User Documentation:

- User manual to help with event creation, login, and registration.
- Tooltips when hovering over some options.

7. Assumptions:

- Users have university emails.
- Users have basic computer literacy and internet access.
- Authentication system is operational.
- Notifications are configured using SMTP.

Specific Requirements:

1. User Requirements (what users need):

- Users need a simple and easy-to-navigate frontend
- Students need the ability to view the website from multiple device types.
- Students need the ability to see event status such as: full, open, or cancelled.
- Students need email confirmations on changes done to their accounts and/or posting events.
- Instructors need to be able to track user attendance for feedback.
- Instructors need email confirmations after posting or modifying events.
- Admins need the ability to track user activity.
- Users need the website to be operational at all times.

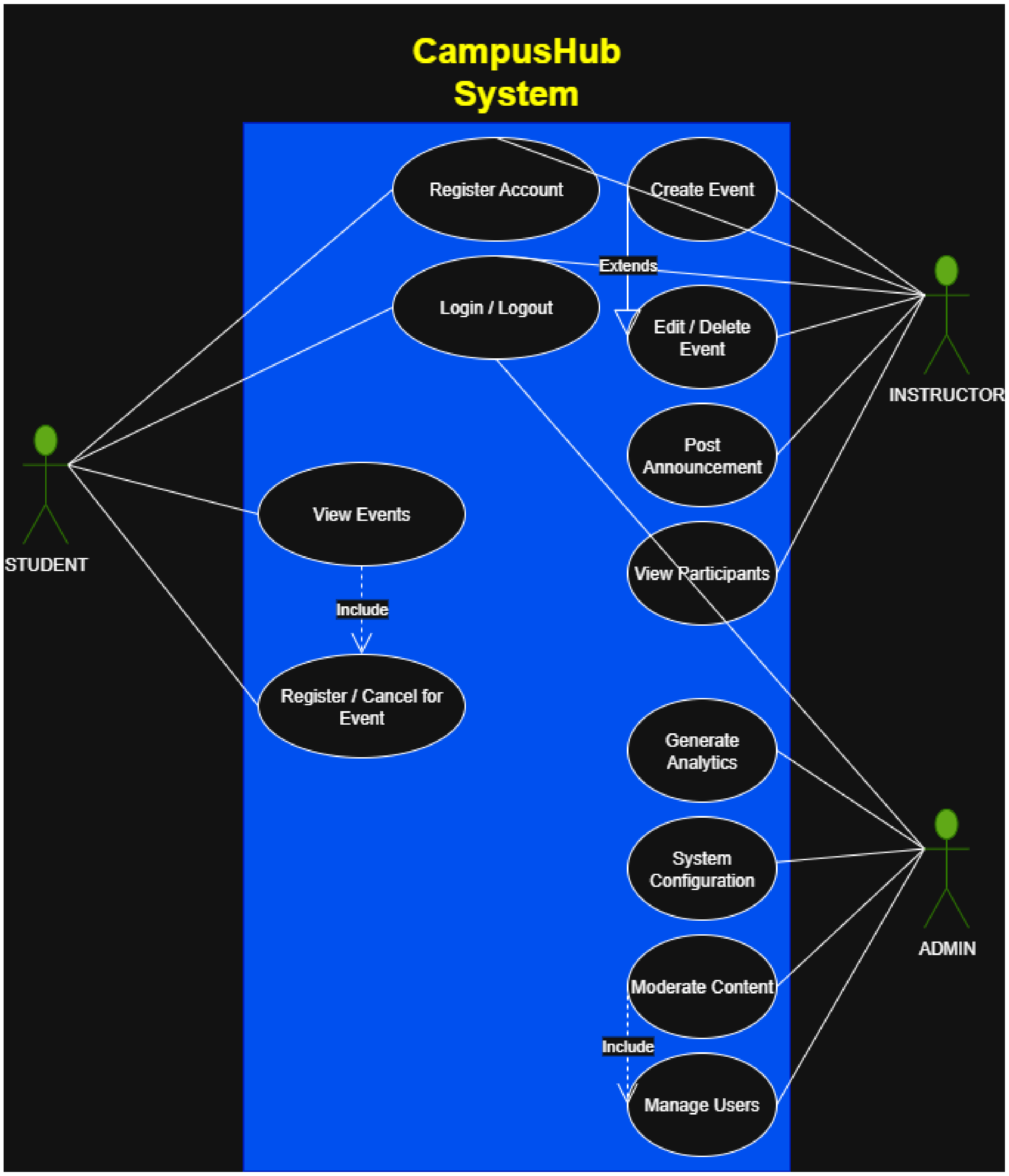
2. System Requirements (what the system must do):

- The system shall generate event participation statistics.
- The system shall enforce user roles.
- The System shall log users out after 5 minutes of inactivity.
- The system encrypts passwords using SHA-256 or better.
- The system shall validate the user input to make sure it is a valid entry.
- The system shall allow multiple users to update their data at the same time.
- The system shall record activity logs.
- The System will send automated emails using SMTP for any changes made.

3. Functional Requirements (what features the system provides):

- FR1: Users shall be allowed to create accounts using university email.
- FR2: Users shall be able to login using their credentials creating their own session.
- FR3: Instructors shall be able to create and/or manage event details.
- FR4: Students shall be able to register or unregister from events.
- FR5: Instructors shall be able to post announcements.
- FR6: Users shall be able to receive optional notifications via email.
- FR7: Users shall be able to search for events.
- FR8: Users shall be able to manage their own profile. (changing passwords, etc.)
- FR9: Admins shall be able to monitor users, instructors, and system logs. They can remove content.

4. Use Case Model:



5. Use Case Descriptions:

❖ *Actors and their rules:*

1. Student:
    - Users can register an account and then log in or log out of the system at any time. They have access to view all the events that are available within the platform right now. They can also register for those events or cancel the registration if needed. This comes as part of the viewing function overall.
2. Instructor:
    - This role lets users log in or log out of the system whenever they need to.
    - They can set up new events and make changes to them or even delete ones that already exist. This feature builds right on top of the basic login access.

- Users also have the option to share announcements. These help keep students in the loop about any updates or new opportunities that come along.
- On top of that, they get to check out the list of participants who signed up for events they organized.

3. Admin:
   - Oversees system operations and settings by managing configurations and moderating content.
   - Can produce analytics to assess event attendance and user engagement.
   - Holds administrative authority to manage users, which is part of the content moderation process.

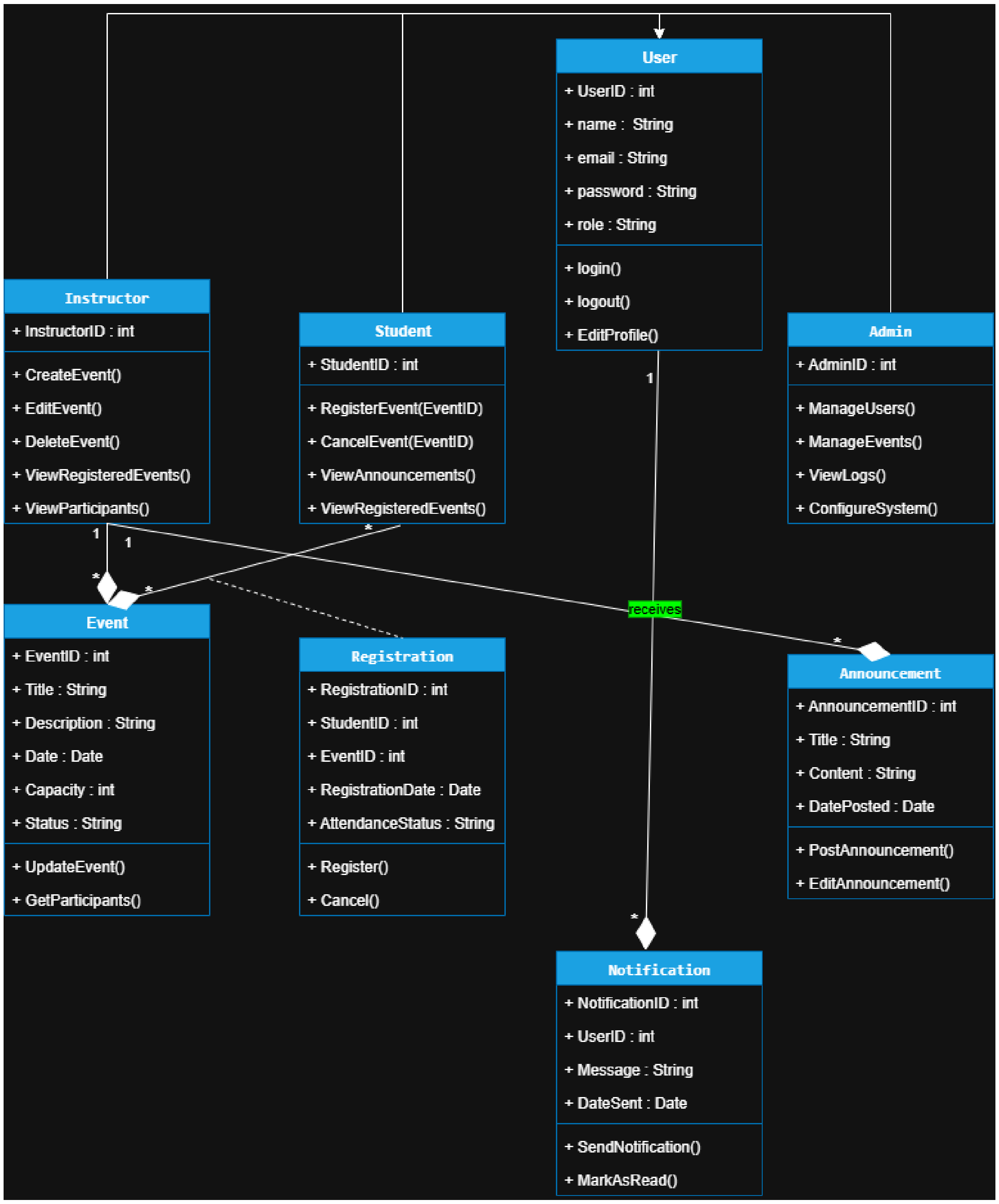
❖ *Use Case Relationships:*

- The "Include" relationship is found between View Events and Register/Cancel for Event, meaning that in order to register for an event, users must first view the events.
- The "Extends" relationship links Login/Logout with Edit/Delete Event, suggesting that users must be authenticated to edit or delete events.

❖ *Overall System Goal:*

- The goal of the CampusHub System is to consolidate event management, enhance communication between students and instructors, and enable administrators to effectively monitor system performance and user activities.

6. Domain Model:



7. Class Descriptions:

❖ *User Hierarchy:*

- At the foundation of the system is the User superclass, which outlines the attributes and methods common to all users. The attributes include UserID, name, email, password, and role, while the methods consist of login(), logout(), and editProfile(). There are three specialized subclasses that derive from User: Student, Instructor, and Admin, each equipped with unique features tailored to their specific roles.

❖ *Student:*

- Handles event participation with methods such as registerEvent(), cancelEvent(), viewAnnouncements(), and viewRegisteredEvents().

- *Instructor:*
  - Concentrates on event management, utilizing methods like createEvent(), editEvent(), deleteEvent(), and viewParticipants().
- *Admin:*
  - Manages the entire system with methods for manageUsers(), manageEvents(), viewLogs(), and configureSystem().
  - This hierarchical structure promotes code reusability, clarity, and a clear division of responsibilities among different user types.
- *Event and Registration Management:*
  - The Event class serves as the central component of the system, responsible for creating and managing events. Its attributes include EventID, title, description, date, capacity, and status, while its methods are updateEvent() and getParticipants(). Each event is initiated by an instructor and attended by multiple students.
  - The Registration class functions as a connector between Student and Event to represent student participation.
  - Its attributes consist of RegistrationID, StudentID, EventID, RegistrationDate, and AttendanceStatus, with methods for register() and cancel().
  - This design facilitates the tracking of attendance and registration information for each student at every event, ensuring accurate participation records.
- *Communication and System Control:*
  - In the CampusHub System, effective communication is managed through the Announcement and Notification classes.
- *Announcement:*
  - Allows instructors to communicate updates to students.
  - Its attributes include AnnouncementID, title, content, and datePosted, with methods for postAnnouncement() and editAnnouncement().
- *Notification:*
  - Keeps users updated on system events and changes, with attributes like NotificationID, UserID, message, and dateSent, and methods for sendNotification() and markAsRead().
  - The Admin class plays a crucial role in maintaining system functionality by managing user accounts, configuring settings, and overseeing system activities through logs and analytics.
- *Relationships and Overall Design:*
  - *Inheritance:*

- Students, instructors, and admins inherit shared characteristics from the User class.

  - *Association:*

- The Event class is linked to both Instructor and Registration, while Registration connects Student to specific Event entries.
- User is associated with Notification, ensuring that all users can receive multiple alerts.

        - *Multiplicity:*

- An instructor can create numerous events, a student can register for various events, and each event can produce multiple announcements and notifications.

❖ *Conclusion:*

  - The CampusHub Class Diagram offers a clear and organized depiction of how users, events, and communications interact within the system.
  - It emphasizes a strong inheritance hierarchy for user roles.
  - Significant associations between users, events, and notifications.
  - Logical modularity that enhances scalability and maintenance.
  - This model guarantees that the system remains organized, adaptable, and capable of efficiently managing complex university activities.

8. Non-Functional Requirements:

Usability:

- **Description:**

The system interface should be intuitive, user-friendly, and easily navigable by both students and instructors.

- **Test Plan:**

Conduct a usability test with 10 participants (5 students, 5 instructors) to perform core tasks such as login, event registration, and messaging.

- **Success Criteria:**

At least 90% of participants can complete all key tasks within 5 minutes without external assistance.

Performance

- **Description:**

The system must handle multiple concurrent users efficiently without noticeable delays.

- **Test Plan:**

Perform load testing with 100 concurrent users accessing various features simultaneously.

- **Success Criteria:**

The average response time should be less than 3 seconds per request.

Security:

- **Description:**

User data must be protected through encryption and secure authentication methods. Passwords should be hashed and transmitted via secure channels (HTTPS).

- **Test Plan:**

Attempt unauthorized login attempts and data breaches under controlled testing.

- **Success Criteria:**

The system blocks unauthorized access and locks accounts after 5 failed login attempts.

Reliability:

- **Description:**

The system should maintain consistent performance and be available most of the time.

- **Test Plan:**

Simulate a week-long operation test to monitor uptime and stability.

- **Success Criteria:**

System uptime should be at least 99% with minimal downtime.

Maintainability:

- **Description:**

The system should be modular, well-documented, and easy to update or modify by future developers.

- **Test Plan:**

A different developer should be able to modify a system feature (e.g., event module) without breaking functionality.

- **Success Criteria:**

The modification is completed successfully in under one hour without introducing errors.

Portability:

- **Description:**

The system should operate correctly across various devices (mobile, tablet, desktop) and browsers (Chrome, Firefox, Edge).

- **Test Plan:**

Test the application on at least three browsers and three device types.

- **Success Criteria:**

The layout and functionality remain consistent and error-free across all platforms.

Scalability:

- **Description:**

The system must support future expansion in the number of users and data volume without major changes.

- **Test Plan:**

Simulate doubled database size and user count.

- **Success Criteria:**

The system's response time degradation does not exceed 10%.

9. External Interface Requirements

User Interface:

- The interface shall be **responsive** and adaptable to different screen sizes (desktop, tablet, mobile).
- The system will support both **light** and **dark mode** for better user experience.
- The interface will follow modern, clean design principles using **HTML5, CSS3, JavaScript**, and frameworks like **Bootstrap or Tailwind CSS**.
- Key pages include:
    - Login and Registration
    - Dashboard
    - Events Management
    - Messaging (Chat)
    - Tasks and Deadlines
    - User Profile
    - Admin Control Panel

Hardware Interface:

- **Server Requirements:**
    - Processor: Intel i5 or higher
    - Memory: Minimum 8 GB RAM
    - Storage: At least 100 GB
- **Client Requirements:**
    - Any internet-connected device (laptop, tablet, smartphone)
    - Modern web browser supporting HTML5

Software Interface:

- **Operating System:** Linux or Windows Server
- **Backend Framework:** Python Flask
- **Database:** MySQL or PostgreSQL

- **Authentication:** JWT (JSON Web Token)
- **Email Service:** SMTP for notifications
- **External APIs:** Integration with Google Calendar API (optional)

Communication Interface:

- The client and server will communicate via **HTTP/HTTPS protocols**.
- Data exchange between frontend and backend will use **RESTful APIs**.
- Real-time notifications will be handled using **WebSocket or Firebase Cloud Messaging (FCM)**.
- All transmitted data must be encrypted using **SSL/TLS** for security.

10.Appendices

Appendix A: Data Dictionary:

The data dictionary describes the main entities of the system, including users, events, messages, tasks, and feedback, along with their attributes and relationships. Each entity contains unique identifiers, descriptive attributes, and foreign key relationships connecting them to other parts of the system.

Appendix B: Glossary:

- **CampusHub:** The proposed software system designed to centralize university life management.
- **Event:** A scheduled academic or extracurricular activity, such as a seminar or workshop.
- **Dashboard:** The main interface where users can access summarized data and key actions.
- **Feedback:** Comments or complaints submitted by users regarding their experience.
- **Use Case:** A specific scenario describing how a user interacts with the system.
- **Actor:** Any entity (person or external system) that interacts with CampusHub.
- **Admin:** The user role with full access and management privileges.
- **Responsive Design:** A web design approach that adapts to different screen sizes automatically.
- **Flask:** A Python-based web development framework used to build the backend.

- **JWT (JSON Web Token):** A secure token format used for authentication and maintaining active sessions.