PES UNIVERSITY, Bengaluru
Department of Computer Science and Engineering
B. Tech (CSE) – 5th Semester – Aug-Dec 2023

UE21CS341A – Software Engineering

PROJECT REPORT
on

Campus Compass : Navigating College Life Together

Submitted by : Team T12

PES1UG21CS564	Shreeja Rajesh	PES1UG21CS574	Shreya Mishra
PES1UG21CS568	Shreya Tippireddy	PES1UG21CS599	Siri Gowri H

Class of Prof. Raghu B. A.

5th Sem. C/J Sec.

Table of Contents

1. Project Proposal / Synopsis

Proposed Project Description: CampusCompass - Navigating College Life Together

In the fast-changing landscape of higher education, college life can often feel overwhelming. Students encounter a multitude of challenges, ranging from academic coursework to personal growth. The project is initiated with the intent of addressing the challenges that college students face in finding and sharing educational resources, connecting with peers, accessing mentorship, and staying updated with important announcements.

Introducing **Campus Compass: Navigating College Life Together** — a vibrant and exclusive peer to peer online community meticulously crafted for college students within a specific institution.

This innovative platform is dedicated to fostering knowledge sharing, mentorship, and community engagement among students, all within the digital realm of their academic journey. It serves as a gathering place for college students to unite, exchange materials and books, offer mentorship, seek clarification for doubts, and access vital announcements within their college community.

Project Objectives

Facilitate resource sharing among college students.
Create a mentorship network for students seeking guidance.
Provide a platform for students to clarify doubts and discuss coursework.
Enable colleges and universities to share announcements and events efficiently.

Users

College Students: The primary users of the platform, including undergraduate and graduate students from various disciplines.
Mentors: Experienced students or alumni willing to provide mentorship and guidance.
Admin: Colleges and universities interested in using the platform to communicate with students and share important information.

Functional Features

1. **Knowledge Sharing:** Campus Compass provides a sophisticated platform for students to exchange educational resources, textbooks, and study materials, easing the financial burden often associated with higher education.
2. **Community Engagement:** The platform serves as a hub for academic discussion, allowing students to seek clarification on doubts, engage in meaningful dialogues, and access essential announcements within their college community.
3. **Mentorship and Guidance:** Connect with experienced seniors who willingly serve as mentors. Seek insightful guidance, career advice, and unwavering support for academic endeavors. Engage in mentorship sessions and Q&A forums to enhance your learning experience.
4. **Doubt Resolution:** Quickly resolve academic uncertainties by posting questions and receiving responses from peers and mentors. Cultivate a cooperative atmosphere that encourages the sharing of knowledge.
5. **Exclusive Resources Repository:** Gain access to an exclusive collection of resources and presentations generously shared by community members. Delve into a comprehensive archive of lecture slides and study aids.
6. **Vital Announcements:** Stay well-informed about essential college announcements, academic deadlines, and upcoming events. Receive timely updates on community initiatives and activities.

Plan of Work

Week 1: Project Proposal and Planning

During the first two weeks, we will convene to discuss the project's objectives and scope, form project teams, and gather initial user requirements. Additionally, we will create a comprehensive project plan and schedule to guide our activities.

Week 2-3: Software Requirements Specification (SRS)

Entering weeks three and four, our attention turns to crafting the Software Requirements Specification (SRS). This crucial phase involves detailed documentation of the functional and non-functional requirements of the "Campus Compass". We will meticulously outline features, user interactions, and system behavior to serve as a comprehensive guide throughout the development process.

Week 4-5: Architectural Design and UML Diagrams

During weeks five and six, our team will define the system architecture, including components like user profiles, resource sharing, and mentorship. Additionally, we will create UML diagrams to illustrate the structure and behavior of the system.

Week 6-7: Development and Coding

Weeks seven and eight will mark the beginning of the development phase. We will commence coding activities, starting with the front-end, back-end, and database components of the community hub.

Week 7-8: Testing and Documentation Kickoff

In the ninth week, we will initiate testing procedures to ensure the functionality, usability, and performance of our community hub. Additionally, we will commence drafting project documentation.

Week 9: Testing, Finalization, and Presentation

The final week of our project will be dedicated to rigorous testing and issue resolution. Simultaneously, we will finalize project documentation. Lastly, we will prepare for the project presentation, where we will showcase the fully functional "Campus Compass".

Product Ownership

Shreeja Rajesh (Frontend and Backend Development)

Weeks 1-2: Requirement gathering and initial team formation.
Weeks 6-7: Lead frontend development activities.
Weeks 9: Contribute to documentation and the final presentation.

Shreya Mishra (Frontend and Backend Development)

Weeks 1-2: Requirement gathering and initial team formation.
Weeks 6-7: Lead backend development activities.

Weeks 9: Contribute to documentation and the final presentation.

Shreya Tippireddy (Testing and Usability)

Weeks 3-4: Focus on design planning and user interface design.
Weeks 7-8: Engage in usability testing and design improvements.
Weeks 9: Contribute to testing, documentation, and the final presentation.

Siri Gowri H (Testing and Documentation)

Weeks 3-4: Focus on design planning and user interface design.
Weeks 7-8: Assist with functional and user acceptance testing.
Weeks 9: Contribute to documentation and the final presentation.

---

2. Software Requirements Specification

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document outlines the software requirements for the "Campus Compass: Navigating College Life Together" project. The software to be specified in this document is a comprehensive online platform designed to address the challenges faced by college students in various aspects of their academic journey and provide them with a peer to peer network between college students.

1.2 Intended Audience

This document is intended for various stakeholders involved in the "Campus Compass" project, including:

Development Team: Developers, designers, and testers responsible for building and validating the system.
Project Managers: Individuals overseeing the project's progress, ensuring it aligns with requirements and timelines.

Quality Assurance Team: Individuals responsible for ensuring the quality and compliance of the final product.
College Students: The primary users of the system, including undergraduate and graduate students from diverse disciplines.
Mentors: Experienced students or alumni willing to provide mentorship and guidance.
Admin: College administrator for the specific college

The rest of this SRS document is organized as follows:

Section 2 provides an overall description of the software, including its features, user classes, and operating environment.
Section 3 specifies the detailed software requirements, including functional and non-functional requirements.
Section 4 presets system models, including use case and entity-relationship diagrams.
Section 5 details external interface requirements, including user interfaces, software interfaces, and communication interfaces.
Section 6 covers other non-functional requirements such as security, performance, usability, and scalability.
Section 7 includes any appendices, glossaries, or revision history as needed.

1.3 Product Scope

The "Campus Compass: Navigating College Life Together" is a purpose-built online platform that aims to enhance the college experience for students within a specific institution. Its primary objectives and goals include:

Resource Sharing: Facilitating the exchange of educational resources, textbooks, and study materials among students, alleviating financial burdens associated with higher education.
Mentorship and Guidance: Creating a mentorship network connecting students with experienced peers who offer academic and career guidance.
Doubt Resolution: Providing a platform for students to post questions, receive responses from peers and mentors, and foster a cooperative learning environment.
Community Engagement: Serving as a hub for academic discussions, enabling students to seek clarification on doubts and access essential announcements.
Vital Announcements: Allow the institution and clubs to efficiently share important college announcements, academic deadlines, and events.

2. Overall Description

2.1 Product Perspective

The "Campus Compass: Navigating College Life Together" project is a self-contained online platform developed to address the challenges and needs of college students within a specific educational institution. It does not directly replace any existing systems but serves as a standalone solution dedicated to enhancing the college experience. The system operates within the context of a single educational institution and is not intended to be part of a larger product family.

2.2 Product Functions

The major functions of the Campus Compass system include:

User Registration and Authentication
User Profile Management
Resource Sharing
Mentorship and Guidance
Doubt Resolution
Community Engagement
Vital Announcements
Admin Management
Mentor Section for Mentors
Mentor Approval System

Detailed specifications for each function are provided in Section 5.

2.3 User Classes and Characteristics

The following user classes are anticipated to use the Campus Compass system:

College Students

Frequency of Use: Regular
Characteristics: Undergraduate and graduate students from various disciplines seeking academic resources, mentorship, and community engagement.

Mentors:

Frequency of Use: Regular
Characteristics: Experienced students or alumni willing to provide mentorship and guidance to their peers.

Admin:

Frequency of Use: Regular
Characteristics: College using the platform to communicate with students and share important information.
User classes are differentiated based on their roles and interactions with the system.

Further details about user classes and their specific requirements are provided in Section 5.

2.4 Operating Environment

The Campus Compass system is designed to operate in the following environment:

Hardware Platform: The system is expected to run on standard hardware configurations commonly used by college students, including personal computers and laptops.

Operating System: The system should be compatible with major operating systems, including but not limited to Windows, macOS and Linux.

Software Components: Campus Compass will be a web-based application and should be compatible with standard web browsers such as Chrome, Safari, and Edge.

Coexistence: The system should coexist peacefully with common software applications installed on users' devices, and it should not conflict with any existing software.

2.5 Design and Implementation Constraints

The development of the Campus Compass system is subject to the following constraints:

Corporate and Regulatory Policies: The project must adhere to the corporate policies and

regulatory requirements of the educational institution.

Hardware Limitations: The system should be designed to operate within typical hardware limitations, including timing and memory constraints.

Interfaces: It must integrate seamlessly with other applications and services, including database systems.

Security Considerations: Security measures must be implemented to protect user data and ensure data privacy.

Programming Standards: The system development should adhere to programming standards and conventions defined by the development team and the educational institution.

2.6 Assumptions and Dependencies

Assumptions:

It is assumed that users will have access to standard computing devices (computers and laptops) and internet connectivity to use the Campus Compass system.

The development team has access to the necessary hardware and software resources for development and testing.

Dependencies:

The project may depend on third-party components or services for certain functionalities (e.g., email services for user authentication).

External dependencies may include the availability of specific software libraries or APIs required for integration.

3. External Interface Requirements

3.1 User Interfaces

The user interface of the Campus Compass system is designed to be intuitive, user-friendly, and

responsive to the needs of college students and mentors. The software product will utilize the Bootstrap 5.1.3 framework for design and styling elements. Bootstrap provides a set of standard design components and conventions that enhance user experience and consistency.

In order to maintain a consistent and user-friendly user interface across the software product, the following UI standards are to be followed:

3.1.1 GUI Standards

The user interface design standards for containers apply to all relevant software components requiring a user interface, ensuring a consistent and visually appealing design which is easy to navigate, including but not limited to:

Content sections
Forms and input screens
Data presentation
Dashboard elements
Dialog boxes and modal windows
Any other user-facing modules

3.1.2 Screen Layout

Screens will be organized to maximize usability and provide a clear flow of information. Key considerations include:

Content Hierarchy: Prioritize the arrangement of content and elements to reflect their importance and relevance to the user.
Readability: Text and visual elements should be presented in a manner that is easy to read, with attention to typography, spacing, and contrast.
Navigation: Navigation menus and elements should maintain a consistent location and behavior across screens.
Styling: The visual styling, including color schemes, typography, and button designs, should be consistent throughout the application.
Button Placement: Ensure that action buttons are appropriately positioned and labeled for clarity.
Form Design: Forms and input fields should be logically organized, with clear labels and validation feedback.

Confirmation Messages: When users complete actions or transactions, they should receive clear and timely confirmation messages.
Error Messages: Error messages should follow a standard format and provide guidance on how to address the issue.

3.1.3 Standard Buttons and Functions

Common interface elements such as navigation menus, buttons for posting posts, sharing resources, searching relevant information and accessing announcements will be present on relevant screens.
The color palette of the software product will adhere to Bootstrap's predefined color classes and other defined colors, ensuring a cohesive and visually appealing design.
Primary and success colors will be used for call-to-action buttons, links, and other key elements as specified in Bootstrap's standards.
Standard buttons within the software product will utilize Bootstrap's predefined button styles. These styles include primary, secondary, success, danger, and other button classes to indicate different actions.
The navigation bar of the software product will follow Bootstrap's navigation bar guidelines. This includes a responsive navigation menu with dropdowns .

3.1.4 Keyboard Shortcuts

Keyboard shortcuts enhance user productivity and are essential for users who prefer keyboard shortcuts for making posts. The following keyboard shortcuts will be implemented:

Text Styling
Bold: The user can apply bold formatting to selected text using the keyboard shortcut Ctrl + B (Windows).
Italic: The user can apply italic formatting to selected text using the keyboard shortcut Ctrl + I (Windows).
Underline: The user can underline selected text using the keyboard shortcut Ctrl + U (Windows).
Strikethrough: The user can apply strikethrough to selected text using the keyboard shortcut Ctrl + Shift + S (Windows).

Text Alignment
Left Alignment: The user can align text to the left using the keyboard shortcut Ctrl + L

(Windows).
Center Alignment: The user can center-align text using the keyboard shortcut Ctrl + E (Windows).
Right Alignment: The user can right-align text using the keyboard shortcut Ctrl + R (Windows).
Justify Alignment: The user can justify-align text using the keyboard shortcut Ctrl + J (Windows).

Undo and Redo
Undo: The user can undo the previous action using the keyboard shortcut Ctrl + Z (Windows).
Redo: The user can redo the previously undone action using the keyboard shortcut Ctrl + Shift + Z (Windows).

Other Operations
Cut: The user can cut selected text or content using the keyboard shortcut Ctrl + X (Windows).
Copy: The user can copy selected text or content using the keyboard shortcut Ctrl + C (Windows).
Paste: The user can paste copied or cut content using the keyboard shortcut Ctrl + V (Windows).
Move Cursor to Start: The user can move the cursor to the beginning of the document or line using the keyboard shortcut Ctrl + Home (Windows).
Move Cursor to End: The user can move the cursor to the end of the document or line using the keyboard shortcut Ctrl + End (Windows).
Select All: The user can select all content within the rich text editor using the keyboard shortcut Ctrl + A (Windows).

3.1.5 Error Message Display Standards

Effective error messages are essential for providing feedback to users when issues or unexpected situations arise. The following error message display standards are to be followed in the software:
Error messages, warnings, and notifications will use Bootstrap's alert component. These alerts will adhere to Bootstrap's styling, including background color and text formatting.

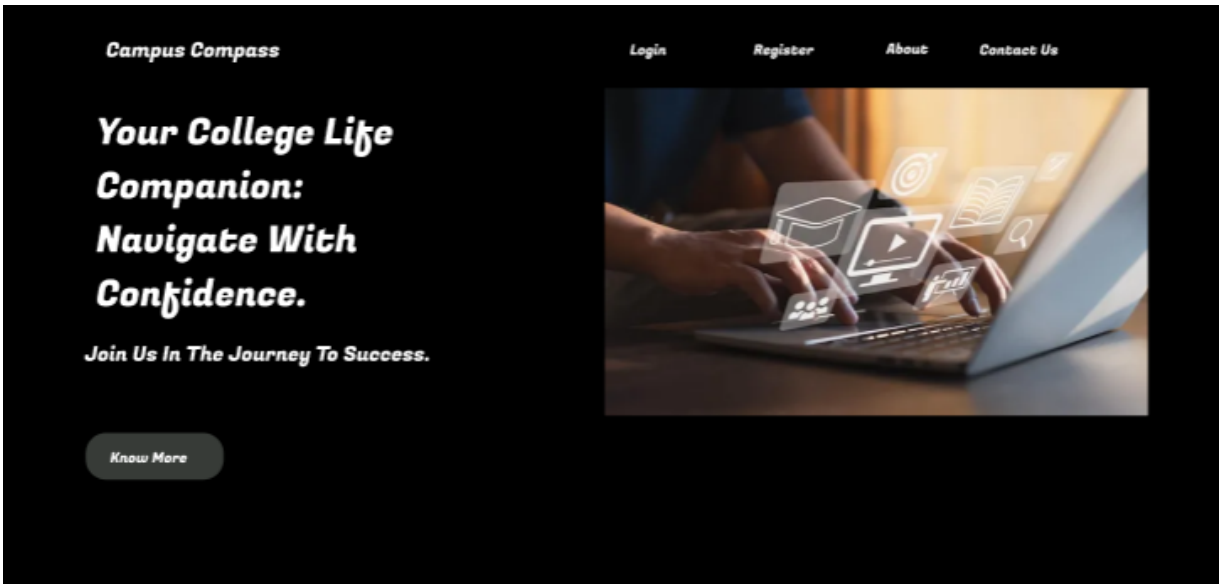
Message Format: Error messages will follow a consistent format, including a clear and descriptive error message text and an icon or visual indicator to draw attention.
Positioning: Error messages will be positioned prominently within the user interface, typically

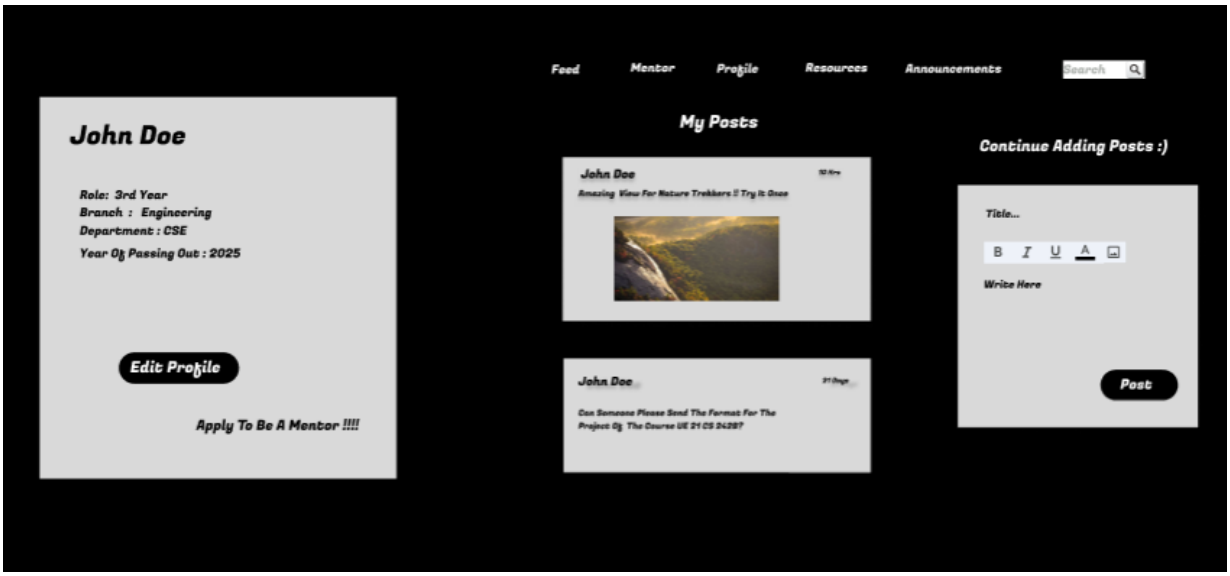
near the point of action or input that triggered the error. They will be clearly visible and stand out from other content.
Plain Language: Error messages will be presented in plain and user-friendly language, avoiding technical jargon or complex terminology.
Specificity: Messages will provide specific information about the nature of the error, what caused it, if applicable.
Color Coding: Different colors may be used to indicate the severity of errors, with red typically denoting critical errors and yellow for less critical issues.

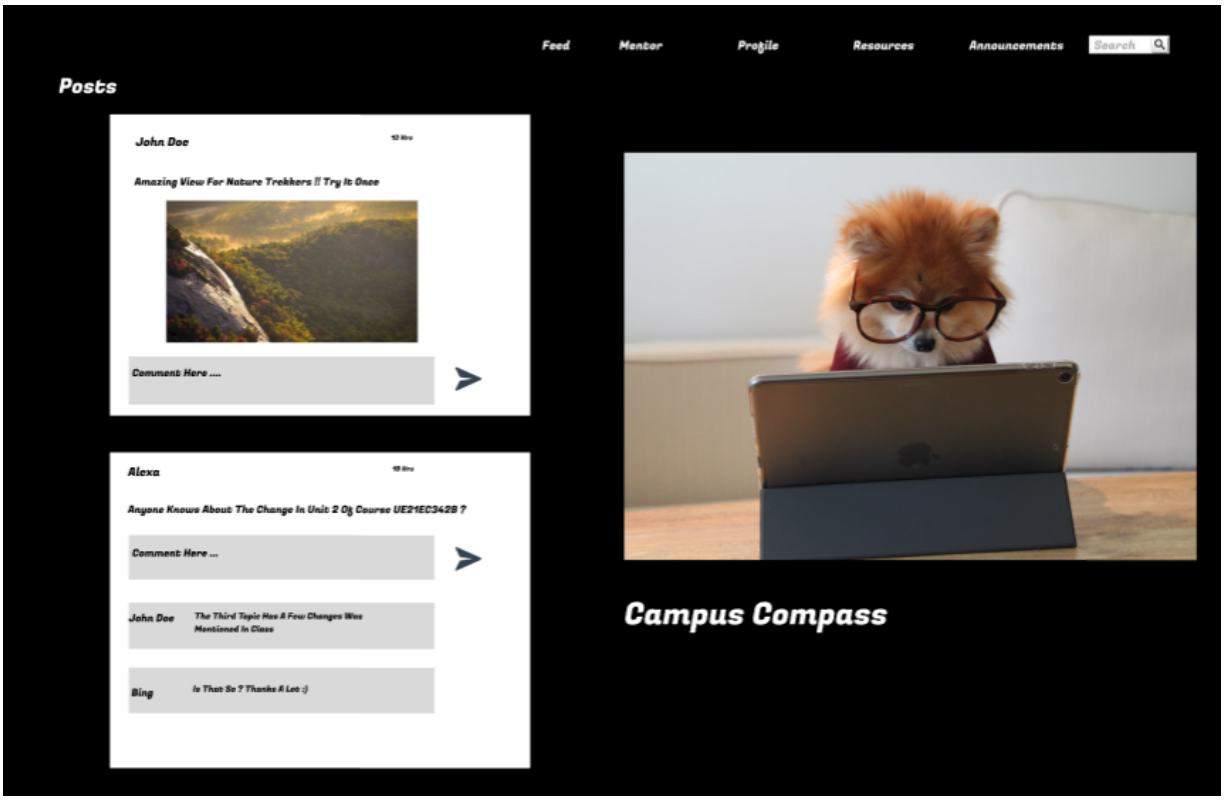
3.1.5.1 Error Categories

Error messages should be categorized based on the type of error. Categories may include:
Validation Errors: Messages that appear when a user submits a form with invalid or missing information.
System Errors: Messages related to technical issues, such as server errors or connectivity problems.
User Authentication Errors: Messages related to login or access permissions, such as incorrect credentials or access denial.
Data Entry Errors: Messages that guide users when entering data, such as formatting or character restrictions.

3.1.6 Sample Screen Images







3.2 Software Interfaces

The Campus Compass system will interact with various software components and external systems. The key software interfaces are as follows:

Database System: The system will communicate with a MySQL database to store and retrieve user data, resources, and system-related information. Specific SQL queries and data transfer protocols will be defined in the database documentation.

Web Browsers: Users will access the system through standard web browsers, and the system will be designed to be compatible with common browsers such as Chrome and Edge.

External APIs: External application programming interfaces (APIs) may be used for features such as email notifications or integrations with third-party services. API documentation will be referenced for integration.

Libraries and Frameworks: The development of the system may depend on various libraries and frameworks, such as Django for backend development and JavaScript libraries and Bootstrap framework for frontend functionality.

External Components: External components, if any, will be identified, and the data exchange mechanisms will be defined in accordance with the requirements of those components.

3.3 Communications Interfaces

The communication interfaces are as follows:

Data Exchange and Flow:
User requests will be processed by the Django framework, which will communicate with the MySQL database to retrieve or store data.
Bootstrap will ensure that the data is presented to users in a visually appealing and responsive manner.
Django's ORM system will be used to manage data and communicate with the database.

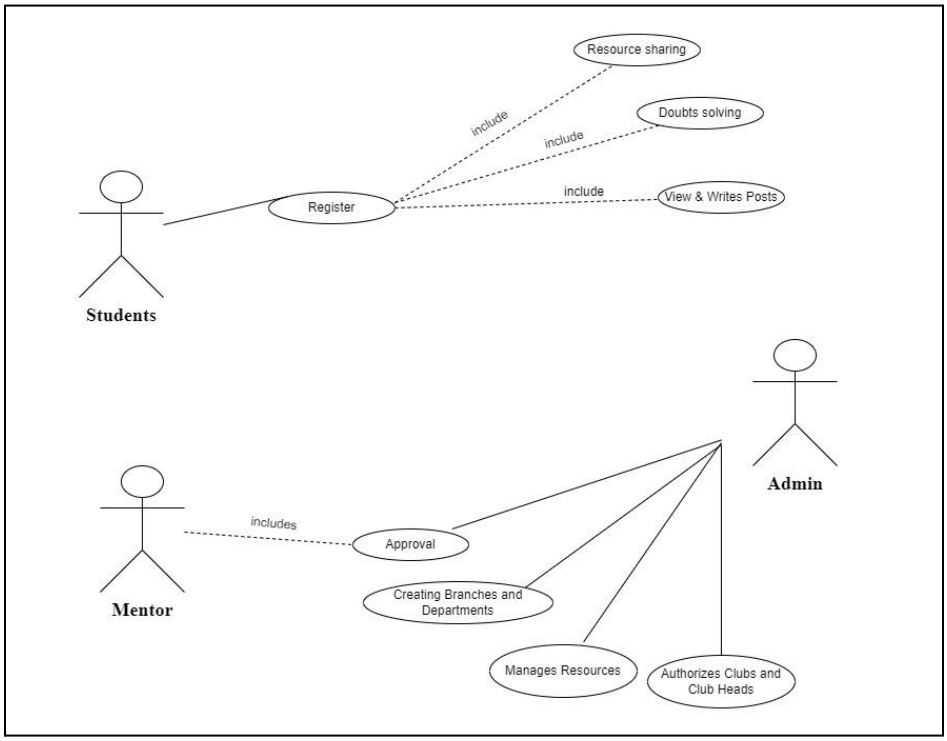
Email Communication: The system will use email for account registration and password reset. SMTP (Simple Mail Transfer Protocol) or email APIs will be used for email communication.

Web Server Protocols: The system will communicate with web servers using standard HTTP (Hypertext Transfer Protocol).

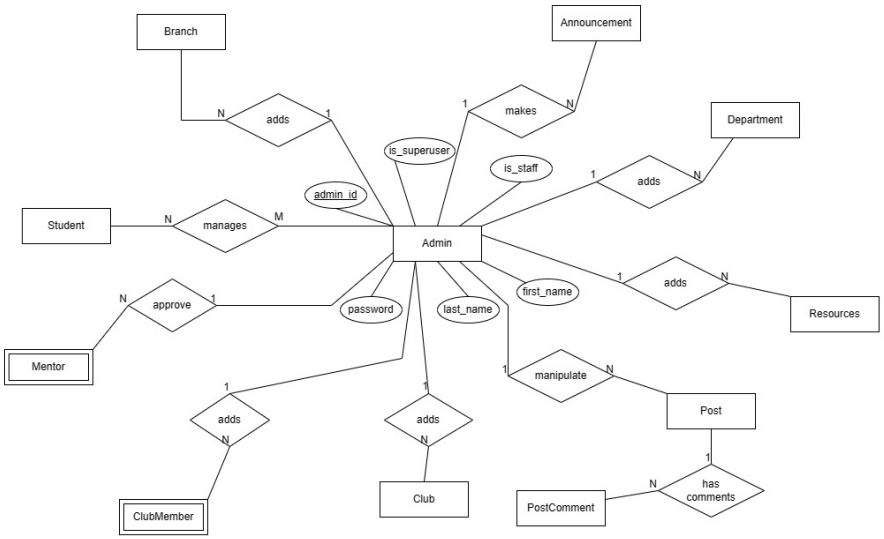
Data Transfer: Data transfer rates and synchronization mechanisms will be optimized to ensure efficient communication, especially for resource sharing and doubt resolution features.

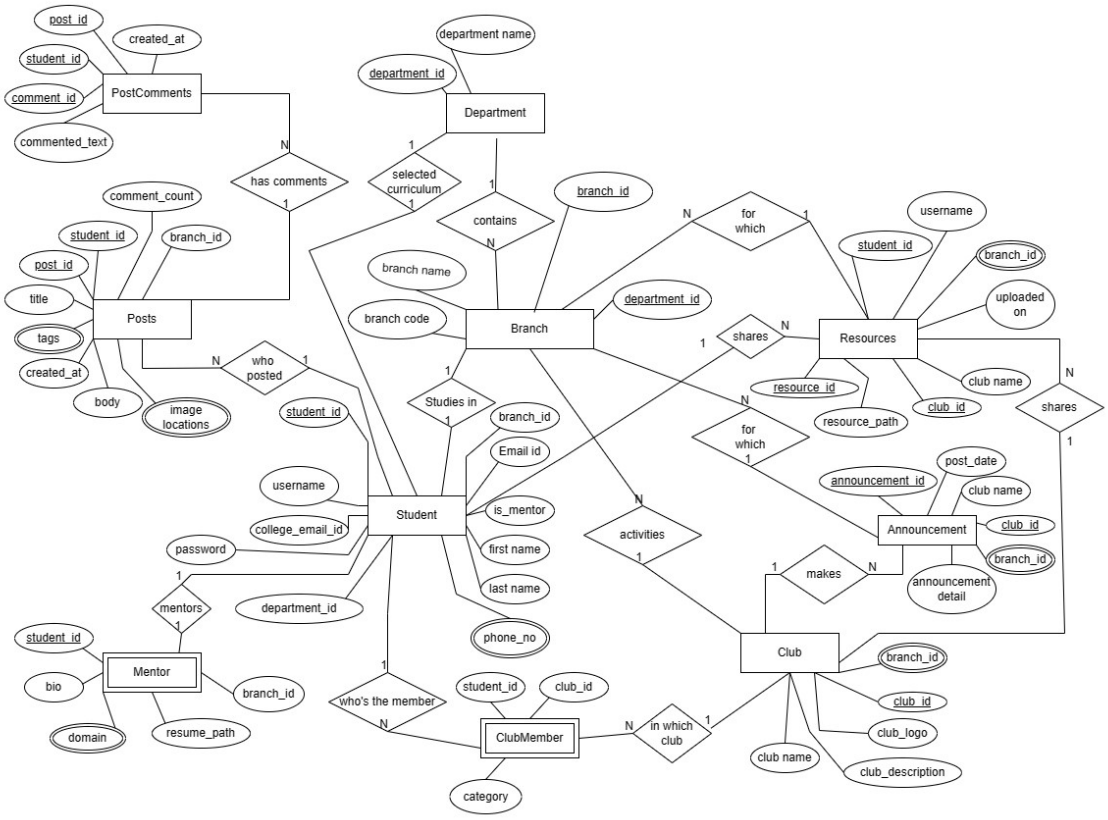
4. Analysis Models

Use Case Diagram



ER Diagram





5. System Features

5.1 *User Registration and Authentication*

5.1.1 *Description and Priority*

This feature involves the registration of new users and the authentication of existing users to access the Campus Compass system. User registration is of high priority as it is the first step in gaining access to the system.

5.1.2 *Stimulus/Response Sequences*

Stimulus:

A new user accesses the Campus Compass website.
The user selects the "Register" option.

Response:

The registration form is displayed.
The user enters their information, including name, email, password, and other required details.
The user submits the registration form.
The system validates the user's information.
If the information is valid, the user is registered, and their profile is created.
If there are validation errors, the system displays error messages and prompts the user to correct the information.
A confirmation link is sent to the user’s college email for verification of the account.

5.1.3 *Functional Requirements*

REQ-1: The system shall provide a user registration form that includes fields for the user's name, email address, password, and other required details provided in appendix B.

REQ-2: The system shall validate user-provided information, including email format, password strength, and unique email addresses.

REQ-3: Upon successful registration, the system shall create a user profile with the provided information.

REQ-4: The system shall securely store user credentials, including passwords, using encryption and hashing techniques to ensure data security.

REQ-5: Users shall be able to log in to the system using their registered email and password.

REQ-6: The system shall provide password recovery options, allowing users to reset their passwords in case of forgotten credentials.

REQ-7: For authentication, the system shall send a confirmation link in the college email id and user access to the system is granted only upon confirmation via a college email link.

REQ-8: Data should be protected against CSRF attacks.

5.2 *User Profile Management*

5.2.1 *Description and Priority*

User Profile Management allows users to edit and manage their profiles. This feature is of medium priority as it provides essential customization options for users.

5.2.2 *Stimulus/Response Sequences*

Stimulus:

A registered user logs in to the Campus Compass system.
The user accesses their profile settings.

Response:

The user is presented with their profile information.
The user can edit and update profile details, including their name, contact information, and profile picture.
After making changes, the user saves their updated profile.
The system validates and stores the updated profile information.
If there are validation errors, the system displays error messages and prompts the user to correct the information.

5.2.3 *Functional Requirements*

REQ-1: The system shall provide users with the ability to view and edit their profile information, including name, email, contact information, and profile picture.

REQ-2: Users shall be able to update their profile information and save the changes.

REQ-3: The system shall validate user-provided information during profile updates and display error messages for any validation errors.

REQ-4: Users shall have the option to upload, change, or remove their profile picture.

REQ-5: User profiles shall display their year or level of study, allowing other users to identify their status.

REQ-6: The system shall ensure data security by securely storing and managing user profile information.

REQ-7: Users will be able to control the privacy settings of their profile, including who can view their profile information.

5.3 *Resource Sharing*

5.3.1 *Description and Priority*
Resource Sharing allows users to upload, share, and access educational resources within the Campus Compass system. This feature is of high priority as it aligns with the core purpose of the system.

5.3.2 *Stimulus/Response Sequences*

Stimulus:

A user logs in to the Campus Compass system.
The user accesses the "Resource Sharing" section.

The user uploads an educational resource (e.g., a document or presentation).

Response:

The user is presented with the resource sharing interface.
The user can upload a resource by providing a title, description, and the resource file.
The system validates and stores the resource in the database.
Other users can browse and search for shared resources.
Users can view details, download, and rate shared resources.
Users can also report inappropriate or misleading resources.

5.3.3 *Functional Requirements*

REQ-1: The system shall provide an interface for users to upload educational resources, including titles and resource files (e.g., PDF).

REQ-2: Users shall have the ability to view, search, and filter shared resources by title, branch category, and rating.

REQ-3: Users shall be able to view resource details, including the uploader's name and title.

REQ-4: Users shall have the option to download shared resources.

5.4 *Mentorship and Guidance*

5.4.1 *Description and Priority*

Mentorship and Guidance is a feature that connects students with experienced mentors for academic and career support. This feature is of high priority as it contributes significantly to the system's educational objectives.

5.4.2 *Stimulus/Response Sequences*

Stimulus:

A user logs in to the Campus Compass system.
The user accesses the "Mentorship" section.
The user can request mentorship or browse available mentors.

Response:

The user is presented with the mentorship interface.
Users can browse a list of available mentors, including their profiles and areas of expertise.
Users can request mentorship from a specific mentor by connecting through whatsapp.
The system notifies the admin about the request and awaits their response.
Admins can accept or decline mentorship requests.

5.4.3 *Functional Requirements*

REQ-1: The system shall provide a directory of available mentors, including their names, profiles and areas of expertise.

REQ-2: The system should provide a mentor's whatsapp link for the users to connect and form mentor-mentee pairs.

REQ-3: The admin is notified of new mentor requests in the system.

5.5 *Doubt Resolution*

5.5.1 *Description and Priority*

Doubt Resolution is a feature that allows users to post academic questions or doubts and receive responses from peers and mentors. This feature is of high priority as it directly supports the educational objectives of the system.

5.5.2 *Stimulus/Response Sequences*

Stimulus:

A user logs in to the Campus Compass system.
The user accesses the "Posts" section.
The user posts an academic question or doubt.

Response:

The user is presented with the post interface.
Users can post academic questions or doubts, including a description and relevant context.
Other users, including peers and mentors, can view and respond to posted doubts.
Users can engage in discussions to resolve doubts by providing answers, explanations, and additional resources.

5.5.3 *Functional Requirements*

REQ-1: The system shall provide an interface for users to post academic questions or doubts, including a description, relevant details, and categorization.

REQ-2: Users shall be able to browse and search for posted doubts, categorized by branch.

REQ-3: Users, including peers and mentors, shall be able to view and respond to posted doubts with answers, explanations, and additional resources.

REQ-4: The system shall ensure the security and privacy of user interactions within the doubt resolution feature.

5.6 *Community Engagement*

5.6.1 *Description and Priority*

Community Engagement is a feature that fosters academic discussion and participation within the Campus Compass system. This feature is of high priority as it promotes a learning environment.

5.6.2 *Stimulus/Response Sequences*

Stimulus:

A user logs in to the Campus Compass system.
The user accesses the "Posts" section.
The user can participate in academic discussions, ask questions, and access vital announcements.
The user can access vital announcements and resources.

Response:

The user is presented with the posts interface.
Users can browse and search for academic discussions and questions, categorized by subjects or topics.
Users can participate in discussions by posting questions, comments, and answers.
Users can access and view vital announcements and updates from the educational institution.
The system provides a user-friendly and responsive interface for efficient engagement and communication.

5.6.3 *Functional Requirements*

REQ-1: The system shall provide an interface for users to participate in academic discussions, post questions, comments, and answers, and engage with the community.

REQ-2: Users shall be able to browse and search for academic discussions and questions, categorized by branch.

REQ-3: Users shall have access to vital announcements and updates from the educational institution.

REQ-4: Users shall be able to comment on discussions and questions to interact.

REQ-5: The system shall ensure the privacy and security of user interactions within the community engagement feature.

5.7 *Vital Announcements*

5.7.1 *Description and Priority*

Vital Announcements is a feature that enables educational institutions to share important announcements, academic deadlines, and upcoming events within the Campus Compass system. This feature is of high priority as it ensures that students are well-informed about essential updates and activities.

5.7.2 *Stimulus/Response Sequences*

Stimulus:

An educational institution, represented by an administrator, logs in to the Campus Compass system.
The administrator accesses the "Announcements" section.
The administrator can create, edit, or delete announcements.

Response:

The administrator is presented with the announcements interface.
Administrators and Club heads can create new announcements, including a title and description.

Announcements are displayed to all users upon logging in to the system.
Users can view and access details of each announcement.

5.7.3 *Functional Requirements*

REQ-1: The system shall provide an interface for educational institution administrators to create, edit, and delete vital announcements.

REQ-2: Announcements shall include a title and description.

REQ-3: Announcements shall be displayed to all users upon logging into the system.

REQ-4: Users shall be able to access details of each announcement, including associated dates.

REQ-5: The system shall ensure that only authorized administrators and club heads can create or edit announcements.

REQ-6: The system shall ensure the privacy and security of user interactions and information within the announcements feature.

5.8 *Admin Management*

5.8.1 *Description and Priority*
Admin has many features namely to add new branches and departments to the system , to manage user roles and permissions, to approve or reject mentor applications , to create and publish announcements , to manage user accounts, including creating, modifying, or deactivating accounts. It is of high priority as it controls the overall system administration.

5.8.2 *Stimulus/Response Sequences*

Stimulus:

Admin logins from the Admin dashboard.
Admin can access the following sub-features:
Add Branches and Departments
User Role Management
Mentor Approval
Make Announcements
Manage User Accounts
The system responds to the admin's selection of each sub-feature accordingly.

Response:

For the following sub-features, Admin gets the following responses:
Add Branches and Departments

Admin provides branch/department details and the system adds the new branch/department to the database.

User Role Management

Admin selects a user and modifies their role and permissions and the system updates the user's role and permissions.

Mentor Approval

Admin reviews mentor applications and approves or rejects mentor applications and the system updates the status of mentor applications in the database.

Make Announcements

Admin enters announcement details and publishes the announcement the system displays the announcement to all users.

Manage User Accounts

Admin selects a user account to edit and can modify user account details or deactivate the account and the system updates the user account as per admin's actions.

5.8.3 *Functional Requirements*

Add Branches and Departments

REQ-1: The system should provide a form for entering branch/department details.

REQ-2: The system should validate the information provided by the admin.

REQ-3: If the information is valid, the system should add the new branch/department to the database.

REQ-4: If the information is invalid, the system should display an error message .

User Role Management

REQ-1: The system should provide a user role management interface.

REQ-2: The system should allow the admin to select a user and modify their role and permissions.

REQ-3:The system should update the user's role and permissions in the database.

Mentor Approval

REQ-1: The system should provide a list of mentor applications.

REQ-2: The system should allow the admin to review and approve/reject applications.

REQ-3: The system should update the status of mentor applications in the database.

Make Announcements

REQ-1: The system should provide an announcement creation interface.

REQ-2: The system should allow the admin to enter announcement details.

REQ-3: The system should publish the announcement to all users.

Manage User Accounts

REQ-1: The system should provide a user account management interface.

REQ-2: The system should allow the admin to select a user account and edit their details.

REQ-3: The system should update the user account in the database based on the admin's actions.

5.9 *Mentor Section for Mentors*

5.9.1 *Description and Priority*

The Mentor Section feature allows mentors within the Campus Compass system to create individual mentor profiles that include their resume, domain of expertise, and contact information. This feature is of medium priority as it enhances the mentorship experience within the system.

5.9.2 *Stimulus/Response Sequences*

Stimulus:

A mentor logs in to the Campus Compass system.
The mentor accesses their profile settings.
The mentor enters or updates their resume, domain of expertise, and contact information.

Response:

The mentor is presented with their profile settings interface.
The mentor can enter and update their resume, including educational background, work experience, and achievements.
The mentor specifies their domain of expertise or specialization.
The mentor can enter and manage their contact information, such as email or phone number.

5.9.3 *Functional Requirements*

REQ-1: Mentors shall have the ability to manage their individual profiles within the system.

REQ-2: Mentor profiles shall include a section for resumes, allowing mentors to provide details of their educational and professional background.

REQ-3: Mentor profiles shall include a field for specifying their domain of expertise or specialization.

REQ-4: Mentor profiles shall include contact information fields, allowing mentors to share their email address or other relevant contact details.

REQ-5: Mentors shall be able to update and maintain their profiles over time.

REQ-6: The system shall ensure the privacy and security of mentor contact information.

REQ-7: Users seeking mentorship shall be able to view mentor profiles, including resumes, domains of expertise, and contact information.

REQ-8: The system shall allow users to initiate contact with mentors through the provided contact information.

5.10 *Mentor Approval System*

5.10.1 *Description and Priority*

The Mentor Approval System is a feature designed to manage the approval process for individuals seeking mentorship roles within the Campus Compass system. This feature is of high priority as it ensures that mentors meet the required criteria before they can provide guidance and support to other students.

5.10.2 *Stimulus/Response Sequences*

Stimulus:

A user applies to become a mentor within the system.
The system identifies mentorship applications and reviews them.
The system approves or rejects mentorship applications.

Response:

The system provides a user-friendly mentorship application interface.
Mentorship applications are collected and reviewed by administrators or designated mentors.
The system checks for the fulfillment of mentorship criteria, such as academic standing or experience.
Approved mentorship applications are granted mentor status, while rejected applications are communicated to applicants with reasons.

5.10.3 *Functional Requirements*

REQ-1: The system shall provide a mentorship application interface for users interested in becoming mentors.

REQ-2: Mentorship applications shall collect information such as resume, experience, and domain of expertise.

REQ-3: The system shall include an approval process to review and evaluate mentorship applications.

REQ-4: The system shall check for fulfillment of mentorship criteria, which may include academic standing, experience, or other relevant factors.

REQ-5: Approved mentors shall be granted mentor status, allowing them to provide guidance and support to other students.

REQ-6: The system shall ensure the privacy and security of mentorship application data.

6. Other Nonfunctional Requirements

6.1 Performance Requirements

Performance requirements for the Campus Compass system according to the functional features are as follows:

User Registration and Authentication

Performance Requirement

1. The registration process should be completed within 5 minutes or less, even during peak usage.
2. User authentication should be completed in under 24hrs.

Rationale: These requirements ensure that user registration and authentication processes are quick and responsive, contributing to a positive user experience.

User Profile Management

Performance Requirement

1. User profile updates, including personal information and preferences, should be processed within 120 seconds.

Rationale: This requirement ensures that users can efficiently manage their profiles.

Resource Sharing

Performance Requirement

1. Resource uploads and downloads should be completed within 120 seconds for files up to 10 MB.
2. The system should support concurrent resource sharing by at least 100 users without performance degradation.

Rationale: These requirements ensure efficient resource sharing and accommodate concurrent user activities.

Mentorship and Guidance

Performance Requirement

1. Mentorship requests should be processed within a week.

Rationale: These requirements ensure timely mentorship interactions.

Doubt Resolution

Performance Requirement

1. Doubt resolution queries should be answered within 24 hours or more.

2. The system should support concurrent doubt resolution sessions for at least 200 users without performance degradation.

Rationale: These requirements ensure efficient doubt resolution and support for multiple users.

Community Engagement

Performance Requirement

1. Real-time community discussions and forum posts should have a response time of 120 seconds or less.
2. The system should support at least 200 concurrent users in community discussions without performance degradation.

Rationale: These requirements ensure responsive community engagement and discussions.

Year-Based Roles (Student Hierarchy)

Performance Requirement

1. User role assignments based on year or level of study should be updated within 120 seconds.

Rationale: This requirement ensures efficient management of year-based user roles.

Vital Announcements

Performance Requirement

1. Vital announcements should be delivered to users in real-time, with a maximum delay of 120 seconds.

Rationale: This requirement ensures timely delivery of important announcements.

Admin Management

Performance Requirement

1. The system should respond to admin actions within 120 seconds for actions like user role modification, mentor approval, and user account management.

2. Announcement creation and publication should take no more than 120 seconds.
3. The system should be capable of handling a growing number of branches and departments.
4. The system should support a minimum of 10 concurrent admin users performing different admin tasks without significant performance degradation.

Rationale: These requirements ensure efficient admin management.

Mentor Section for Mentors

1. Mentors' profile updates should be processed within 120 seconds.

Rationale: This requirement ensures efficient management of mentors' profiles.

Mentor Approval System

Performance Requirement

1. Mentorship applications should be reviewed and processed within 24 hrs.

Rationale: This requirement ensures a timely review and approval process for mentors.

6.2 Safety Requirements

Safety requirements for the Campus Compass system:

SAFETY-1: The system should protect user data and privacy, adhering to data protection regulations and policies.
SAFETY-2: All user data, including personal and payment information, must be stored securely and encrypted to prevent unauthorized access like CSRF attack.

Rationale: Safety requirements emphasize the importance of data security and user protection within the system.

6.3 Security Requirements

Security requirements for the Campus Compass system:

SEC-1: User authentication shall be secure and use standard encryption methods.
SEC-2: User data, including personal information, shall be encrypted when stored in the system.
SEC-3: The system shall implement role-based access control to ensure data privacy.
SEC-4: Security audits and regular vulnerability assessments shall be conducted to identify and mitigate potential security risks.

Rationale: Security requirements are essential to protect user data and maintain the integrity of the system.

6.4 Software Quality Attributes

Quality attributes for the Campus Compass system:
QUAL-1: Usability: The system should be intuitive and user-friendly, with a focus on ease of use to encourage adoption and engagement.
QUAL-2: Reliability: The system should operate consistently and reliably, minimizing downtime and errors.
QUAL-3: Maintainability: The system should be designed for ease of maintenance and updates to accommodate future changes and improvements.

Rationale: Software quality attributes are essential to ensure a positive user experience and the long-term success of the system.

6.5 Business Rules

Business rules for the Campus Compass system:

Users in the role of administrators have the authority to manage user accounts, including role assignments.
Mentorship applications must be reviewed and approved by designated administrators before users can become mentors.
Rationale: Business rules define the operational principles of the system and the roles and responsibilities of different user classes.

Also include Domain requirements here.

6.6 Domain Requirements

Data Protection Regulations
Domain Requirement (D1): The system must comply with data protection regulations, including but not limited to the General Data Protection Regulation (GDPR) for the protection of user data and privacy.
Educational Best Practices
Domain Requirement (D2): The system should follow educational best practices, including adherence to principles of academic honesty and integrity. It should encourage students to use resources responsibly and in accordance with academic standards.
Community Engagement Guidelines
Domain Requirement (D3): The system should adhere to community engagement guidelines that promote respectful and constructive interactions among users. It should include mechanisms for reporting and moderating inappropriate content.

7. Other Requirements

Database Requirements

1. The system shall use a relational database management system (RDBMS) for data storage and retrieval.
2. The database shall be designed to efficiently handle a large volume of user-generated content, including text, images, and documents.

Internationalization Requirements

1. The user interface of the system should be designed to support multiple languages, allowing users to select their preferred language.
2. Date and time formats should be displayed according to the user's locale and preferences.

Legal Requirements

1. The system must comply with all relevant legal requirements, including copyright laws, intellectual property rights, and accessibility standards.

Reuse Objectives

1. The project team should identify and document components or modules of the system that have the potential for reuse in future projects.

Data Backup and Recovery

1. The system should implement regular data backup procedures to ensure data integrity and provide a mechanism for data recovery in case of system failures.

Reporting and Analytics

1. The system should include reporting and analytics features to allow administrators to monitor system usage and user engagement.

8. Requirement Traceability matrix

Sl. no.	Requirement ID	Brief Description of Requirement	Architecture Reference	Design Reference	Code File Reference	Test Case ID	System Test Case ID
1	REQ-1	User Registration and Authentication - High Priority	AR-001	DR-001	CR-001	TC-001	STC-001
2	REQ-2	User Profile Management - High Priority	AR-002	DR-002	CR-002	TC-002	STC-002
3	REQ-3	Resource Sharing -	AR-003	DR-003	CR-003	TC-003	STC-003

		Medium Priority					
4	REQ-4	Mentorship and Guidance - High Priority	AR-004	DR-004	CR-004	TC-004	STC-004
5	REQ-5	Doubt Resolution - Medium Priority	AR-005	DR-005	CR-005	TC-005	STC-005
6	REQ-6	Community Engagement - Medium Priority	AR-006	DR-006	CR-006	TC-006	STC-006
7	REQ-7	Vital Announcements - High Priority	AR-007	DR-007	CR-007	TC-007	STC-007
8	REQ-8	Admin Management - High Priority	AR-008	DR-008	CR-008	TC-008	STC-008
9	REQ-9	Mentor Page - Medium Priority	AR-009	DR-009	CR-009	TC-009	STC-009
10	REQ-10	Mentor Approval System - High Priority	AR-010	DR-0010	CR-010	TC-010	STC-010

3. Project Plan with Gantt Chart

Life-cycle followed

The Agile Software Development Model using the Scrum framework
We have chosen the Agile model as it allows for flexibility in responding to changing requirements and promotes continuous collaboration with stakeholders, which is crucial for a project aimed at serving college students' evolving needs.

SCRUM model will allow us to do the following :-

Flexibility: Scrum's iterative approach allows for adapting to changing requirements by organizing work into time-boxed sprints for regular feedback and adjustments.
Risk Mitigation: Scrum identifies and addresses risks early by delivering smaller, manageable portions of the project incrementally.
Regular Inspections: Frequent Sprint Reviews and Retrospectives enable continuous improvement and refinement of the product and development processes.
Enhanced Transparency: Scrum emphasizes transparency through artifacts like the Product Backlog and Burndown Charts, ensuring a clear view of project progress and priorities for all stakeholders.
Continuous Collaboration: Scrum promotes transparent communication with stakeholders through ceremonies like Sprint Reviews and Sprint Planning, ensuring alignment with user needs in future.

Tools Used for this Project

1. Project planning and management tools:
Jira: plays a key role in project planning and management, providing a comprehensive task tracking, agile planning, collaboration and reporting platform that enables teams to organize and execute projects effectively.

2. Design Tools:
Figma: A collaborative design tool for creating wireframes, prototypes and design concepts.
Canva: A user experience and interaction design tool for creating interactive prototypes.

3.Version Control:
Git: A distributed version control system for tracking changes to source code.

GitHub: Online platforms for hosting Git repositories and collaborating with your team.

4. Development tools:

Integrated development environment (IDE): Python's flexibility and Django's powerful features allow for the rapid development of the application's backend logic. Campus Compass leverages the power of Visual Studio Code for efficient development, Python with Django for a robust and secure backend, and HTML/CSS/JavaScript for an intuitive and user-friendly frontend.

Database management system : MySQL

5. Bug Tracking:

Jira: A popular issue and project tracker that can be used to track bugs.

6. Testing Tools:

Selenium: For automated testing of web applications.
Pytest: Pytest is a popular Python testing framework that can be used with Django for more advanced and flexible testing.

7. Documentation tools:

Confluence: Collaboration and documentation tool often used in conjunction with Jira.
Google Docs or Microsoft Word: Used to create project documentation, manuals, and reports.

8. Communication and collaboration tools:

Zoom : For virtual meetings and discussions.
Google Workspace: for email, file sharing and collaboration.

9. Code review and collaboration:

GitHub : In addition to version control, these platforms offer code review and collaboration features.

Deliverables classified as reuse/build components

Reusable Components:
User Authentication Module
Resource Sharing Module
Mentorship Module
Community Discussion Module

Build Components:
User Interface (UI) Components
Database Schema
User Role Management System
Announcement and Event Management System

Work Breakdown Structure

Project: Campus Compass - Navigating College Life Together

1. Project Initiation and Planning (Weeks 1-2)
   1.1. Define project objectives and scope
   1.2. Form project teams and assign roles
   1.3. Gather initial user requirements
   1.4. Create a comprehensive project plan and schedule

2. Design and Prototype (Weeks 3-4)
   2.1. Design user interface concepts
   2.2. Create a functional prototype demonstrating core functionalities

3. Architectural Design and UML Diagrams (Weeks 5-6)
   3.1. Define system architecture
   3.2. Create UML diagrams illustrating system structure and behavior

4. Development and Coding (Weeks 7-8)
   4.1. Front-end Development
      4.1.1. Implement user profiles, registration and login
      4.1.2. Develop resource sharing functionality
      4.1.3. Create mentorship features
      4.1.4. Develop announcement functionality
   4.2. Back-end Development
      4.2.1. Build database components
      4.2.2. Implement user authentication and authorization
      4.2.3. Develop algorithms for doubt resolution
   4.3. Integration of Front-end and Back-end

5. Testing and Documentation Kickoff (Week 9)
  5.1. Functional Testing
    5.1.1. Test user registration, login and user profile
    5.1.2. Test resource sharing functionality
    5.1.3. Test mentorship features
    5.1.4. Test announcement functionality
  5.2. Usability Testing
    5.2.1. Gather user feedback on the prototype
    5.2.2. Implement design improvements
  5.3. Performance Testing
    5.3.1. Ensure system scalability and responsiveness
  5.4. Documentation
    5.4.1. Create user manuals and guides
    5.4.2. Prepare technical documentation

6. Testing, Finalization, and Presentation (Week 10)
  6.1. Continued Testing
    6.1.1. Address issues and bugs
  6.2. Finalize Project Documentation
    6.2.1. Review and edit documentation
  6.3. Project Presentation
    6.3.1. Prepare presentation materials
    6.3.2. Showcase the fully functional "Campus Compass"

7. Project Completion (Week 10)
  7.1. Review and finalize all project deliverables
  7.2. Handover the project to relevant stakeholders
  7.3. Post-launch support planning

8. Ongoing Support and Maintenance (Post-Project)
  8.1. Monitor system performance and address issues
  8.2. Update and enhance features based on user feedback
  8.3. Provide ongoing technical support to users

Functionalities within the above tasks include:
- User Registration and Authentication
- User Profiles

- Resource Sharing
- Mentorship and Guidance
- Doubt Resolution
- Year-Based Roles (Student Hierarchy)
- Vital Announcements
- Content Moderation and User Management

Effort Estimation (in person-months)

Project title:
No. of working days= 7 days
Effort estimation=7/21.66 = 0.323 person-months

SRS and WBS:
No. of working days=10 days
Effort estimation= 10/21.66=0.461 person-months

Designing and Prototyping:
No. of working days= 14 days
Effort estimation=14/21.66=0.646 person-month

Development:
No. of working days= 14 days
Effort estimation= 14/26.1= 0.646 person-month

Testing:
No. of working days= 10 days
Effort estimation = 10/21.6= 0.461 person-months

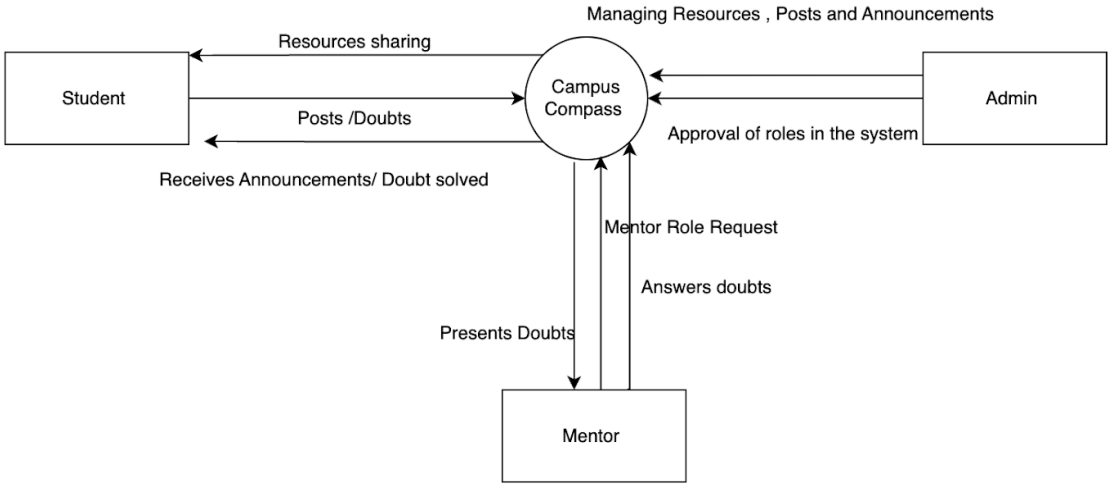
Report and Presentation:
No. of working days= 10 days
Effort estimation = 10/21.6= 0.461 person-months

4. Architecture & Design Choices and Diagrams

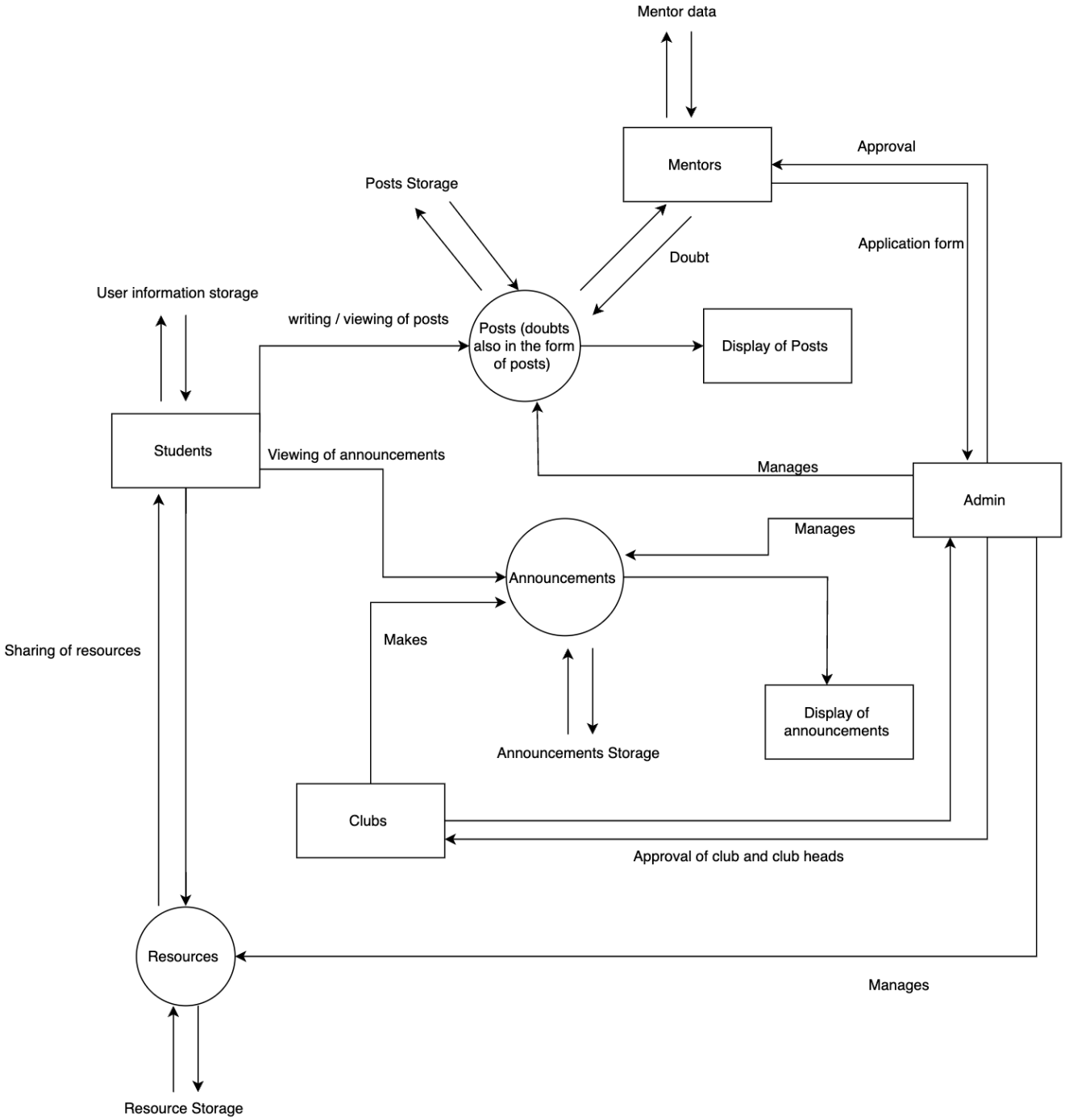
Design Diagrams

Diagrams of Levels of DFD (AR -001)

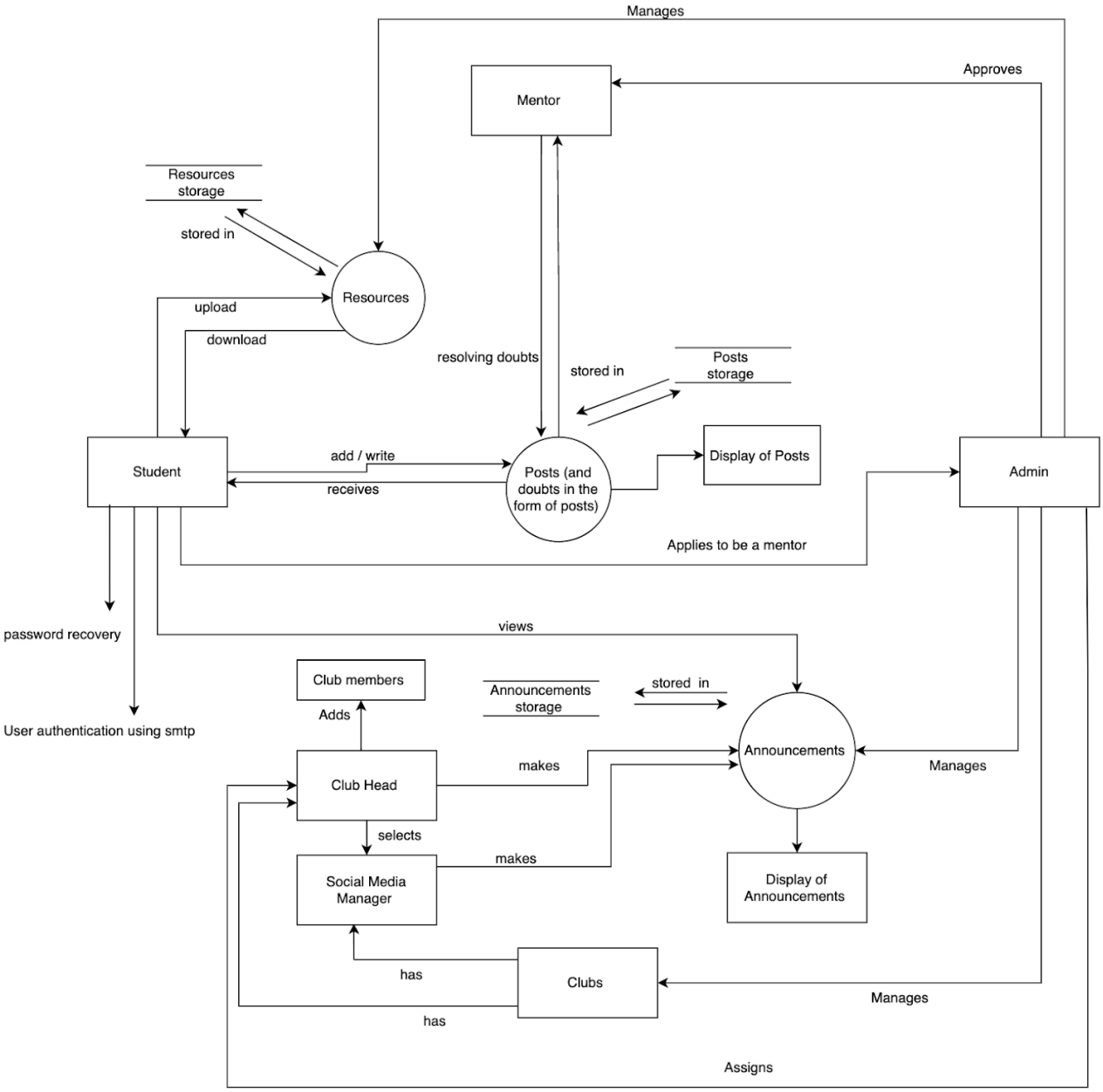
Level 0



Level1

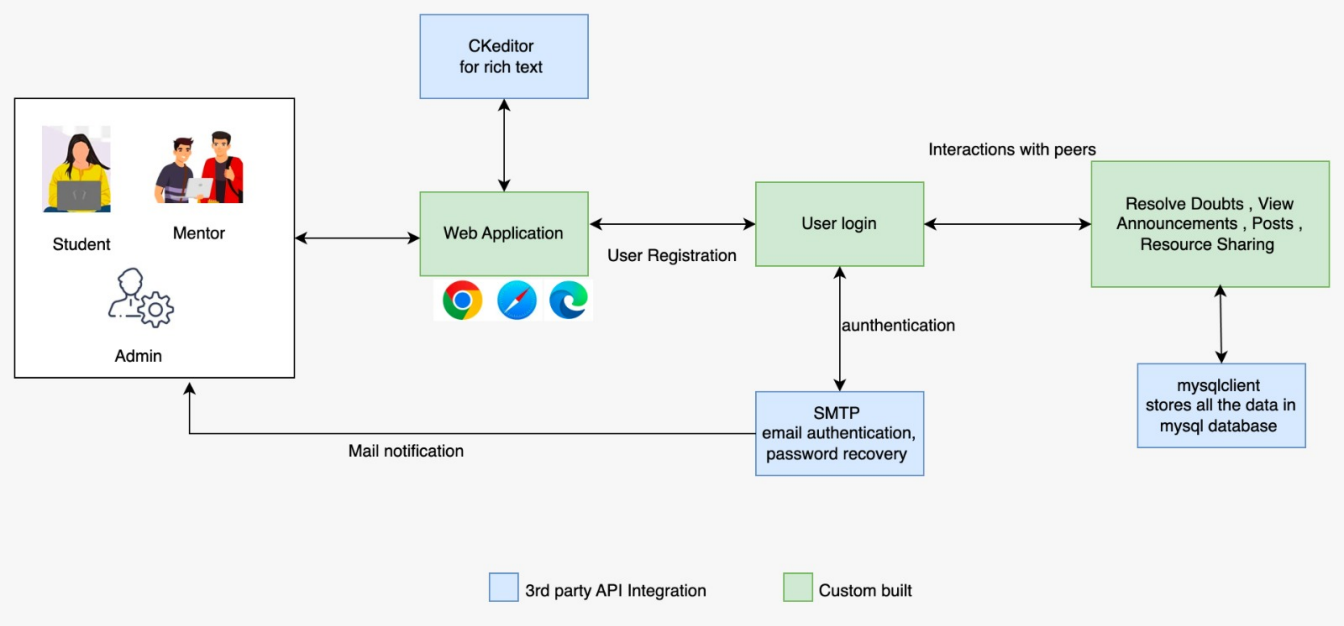

Mentor data
Mentors
Approval
Posts Storage
Application form
Doubt
User information storage
writing / viewing of posts
Posts (doubts also in the form of posts)
Display of Posts
Students
Viewing of announcements
Manages
Admin
Manages
Announcements
Sharing of resources
Makes
Display of announcements
Announcements Storage
Clubs
Approval of club and club heads
Resources
Manages
Resource Storage

Level 2

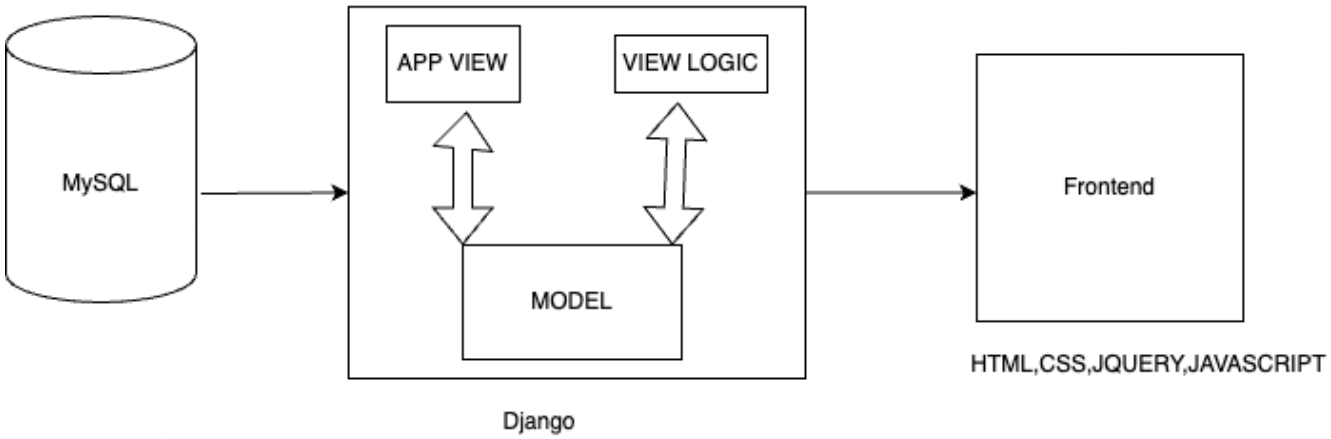

Manages
Approves
Mentor
Resources storage
stored in
Resources
upload
download
resolving doubts
stored in
Posts storage
Student
add / write
receives
Posts (and doubts in the form of posts)
Display of Posts
Admin
Applies to be a mentor
password recovery
User authentication using smtp
views
Club members
Adds
Announcements storage
stored in
Announcements
Manages
Club Head
makes
selects
makes
Social Media Manager
Display of Announcements
has
Clubs
Manages
has
Assigns

Architectural Design

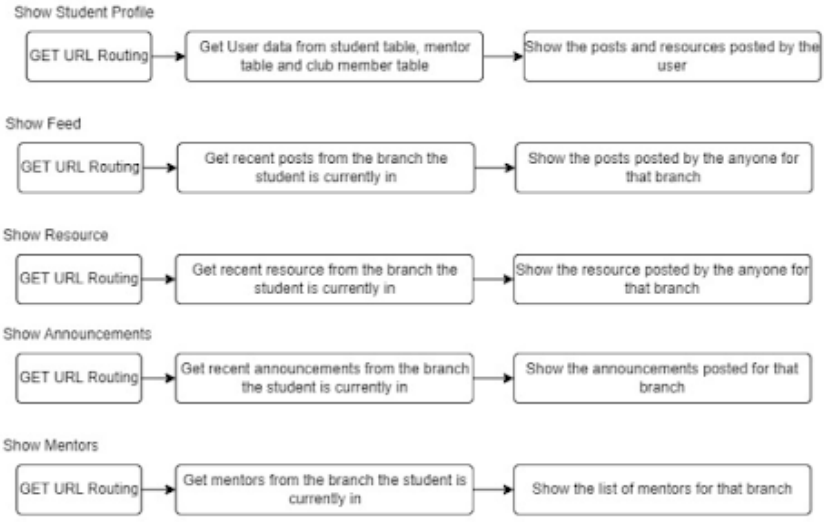
System Architecture Diagram(AR-002)



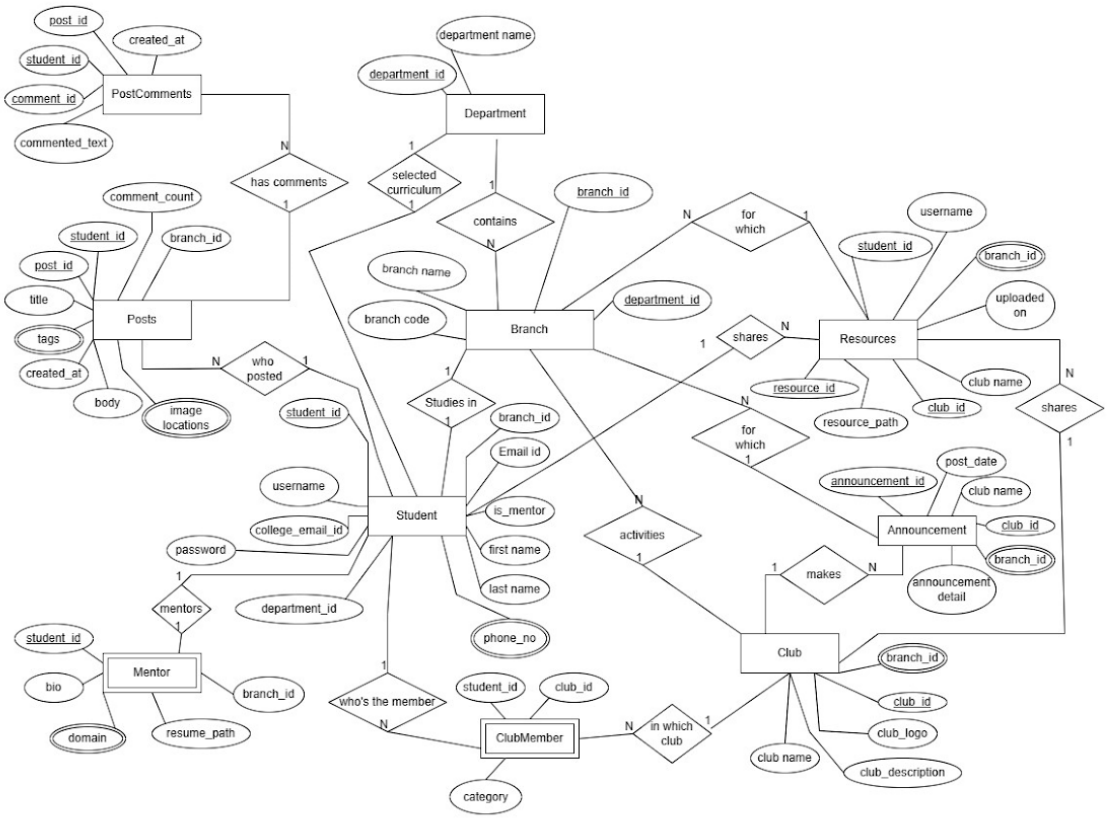
System Overview

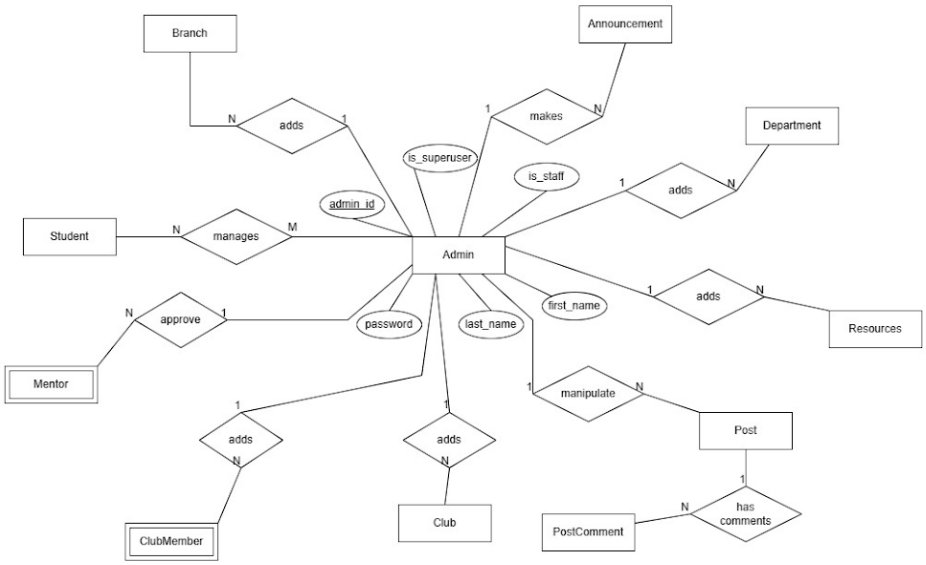


Data Flow Architecture(DR-001)

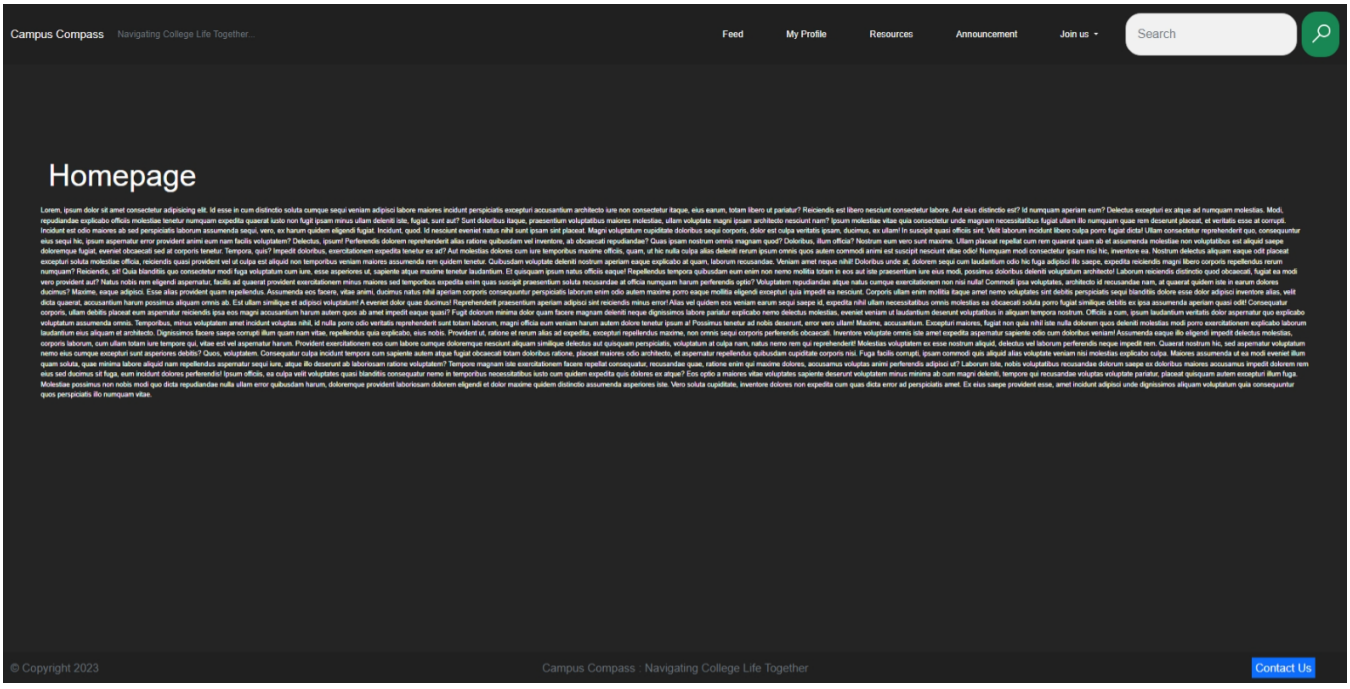


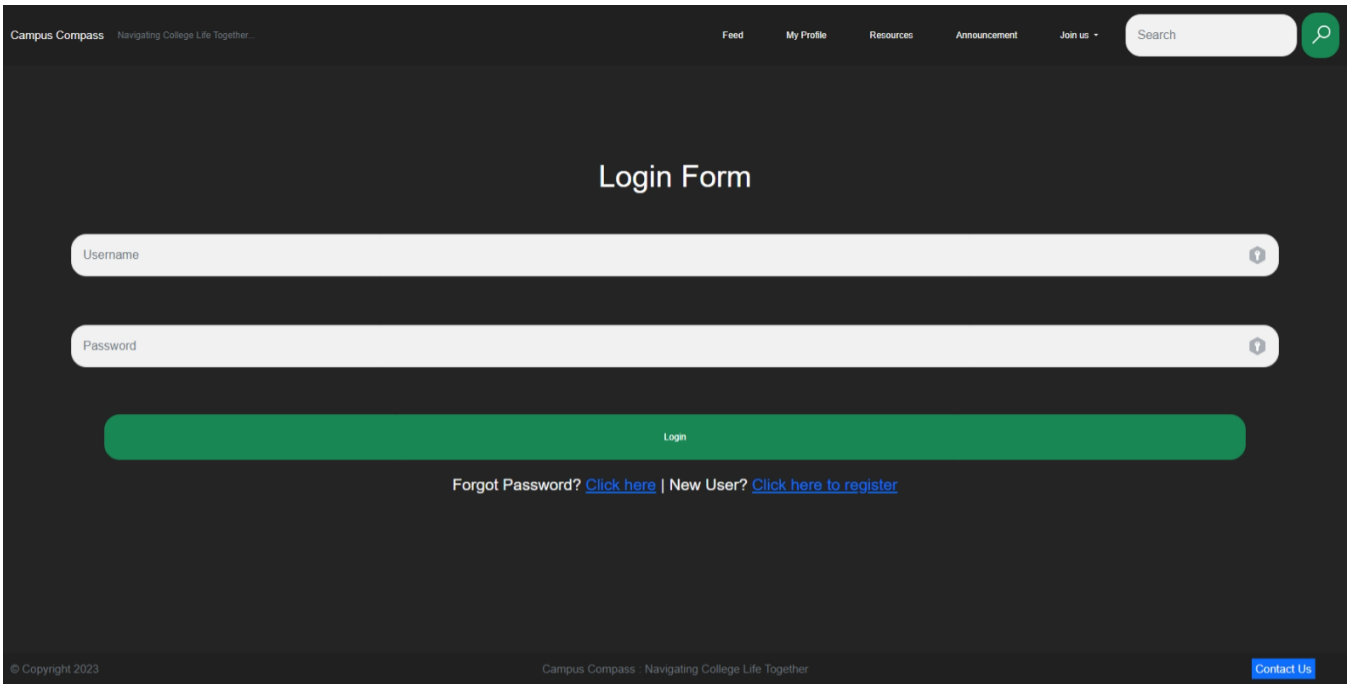
Entity Relationship Diagram(DR-002)

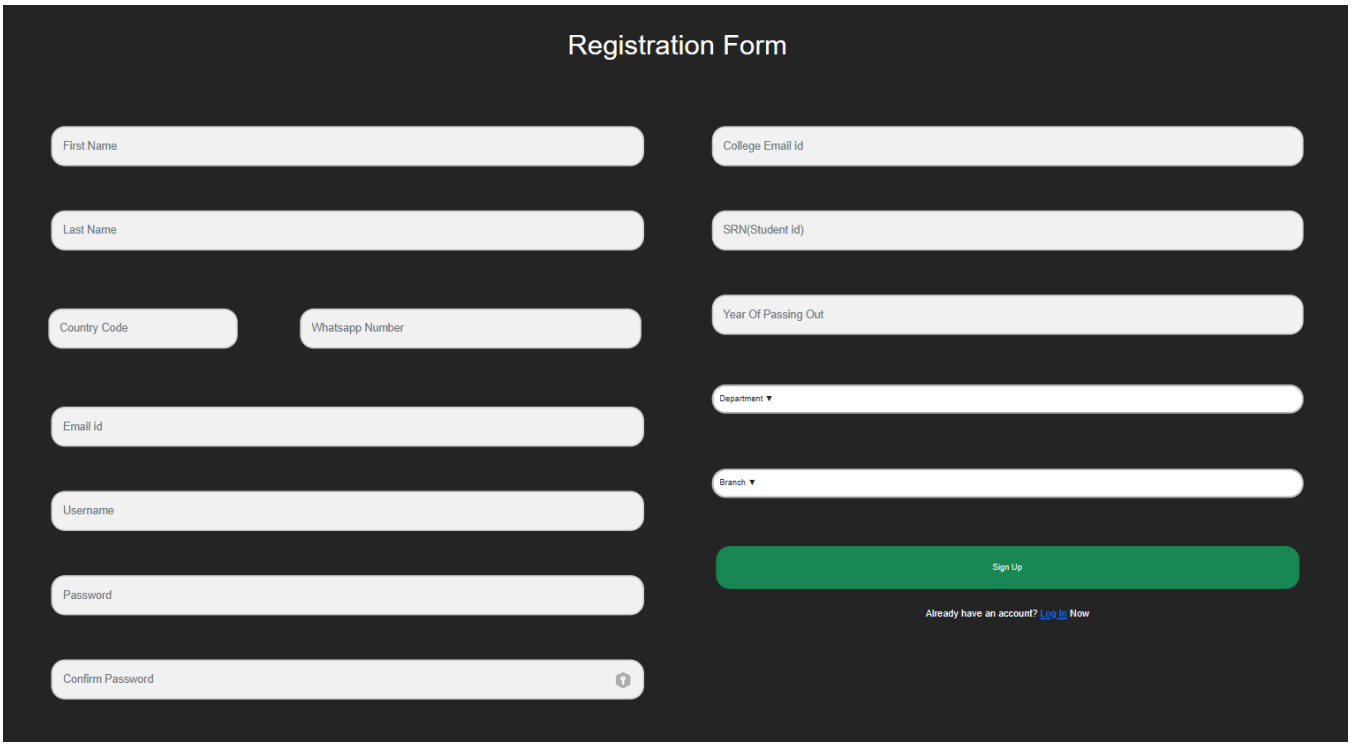


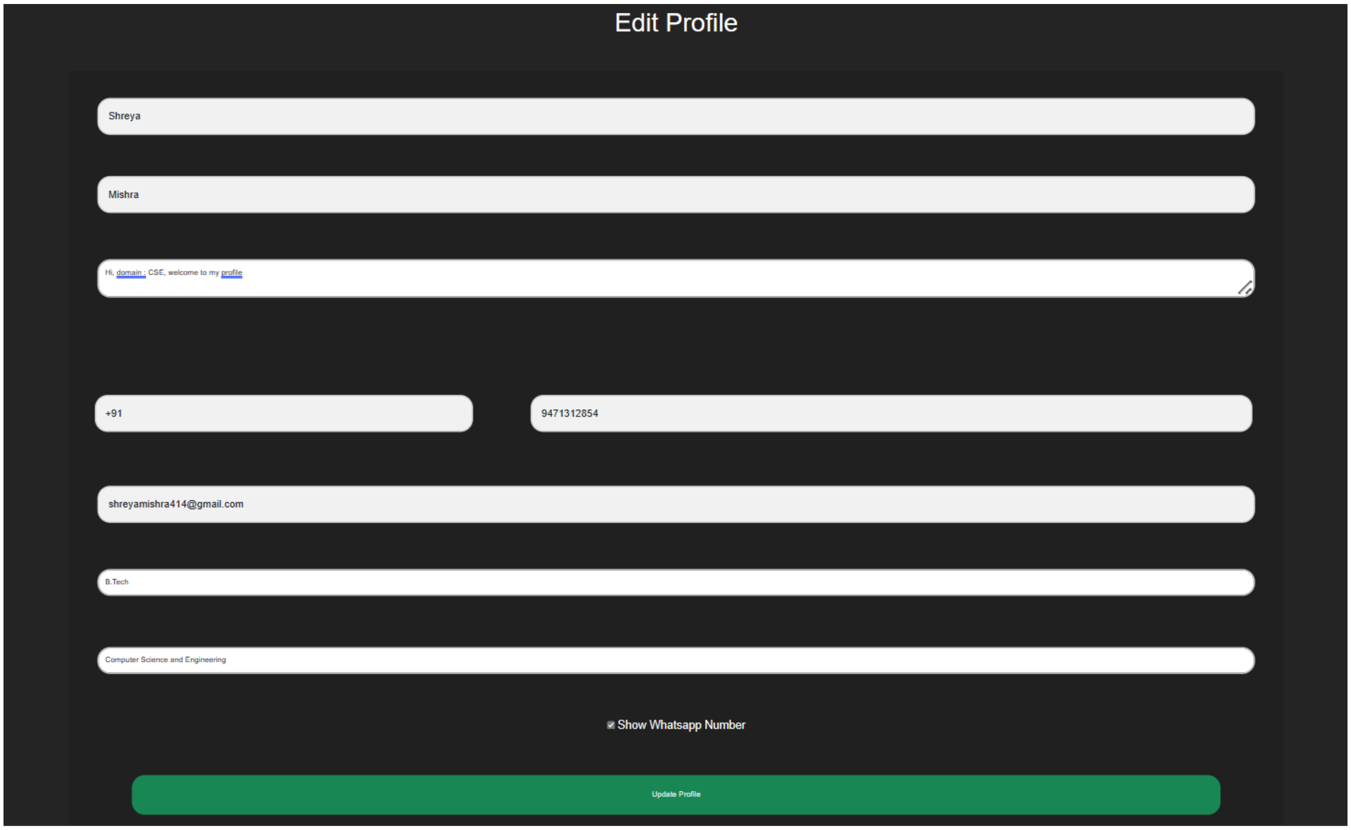


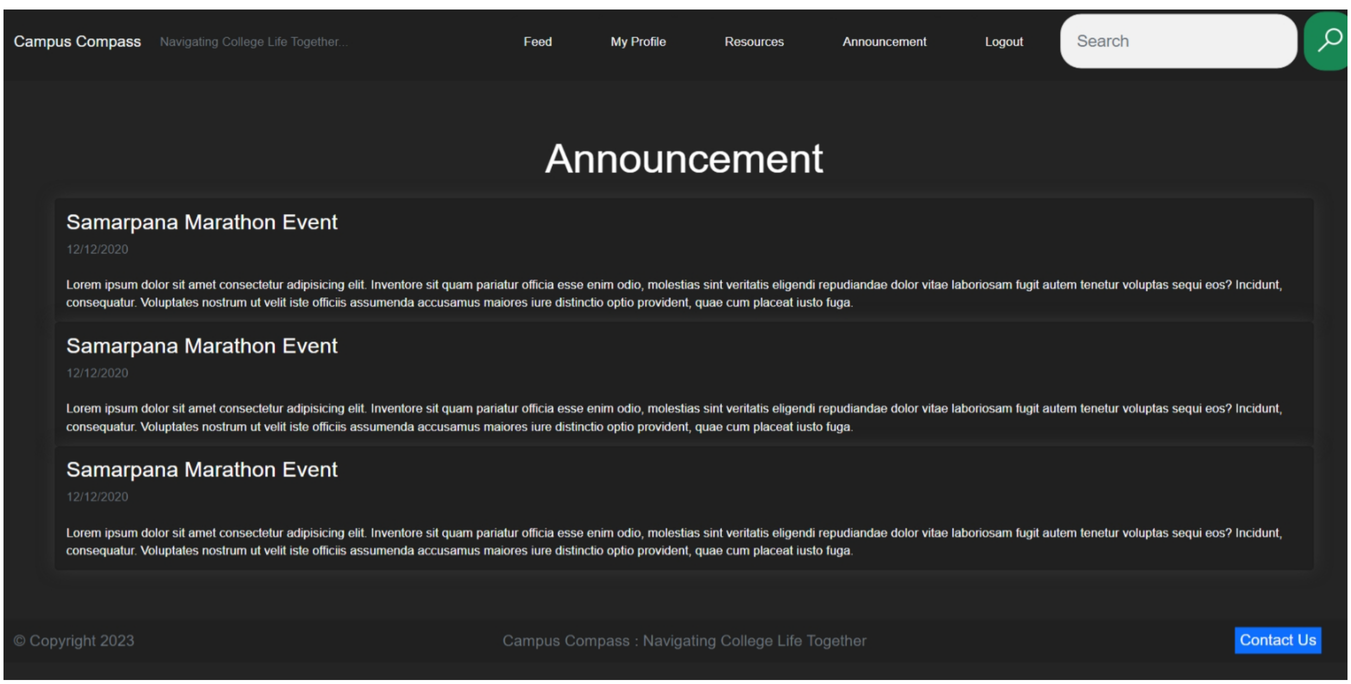
User Interface



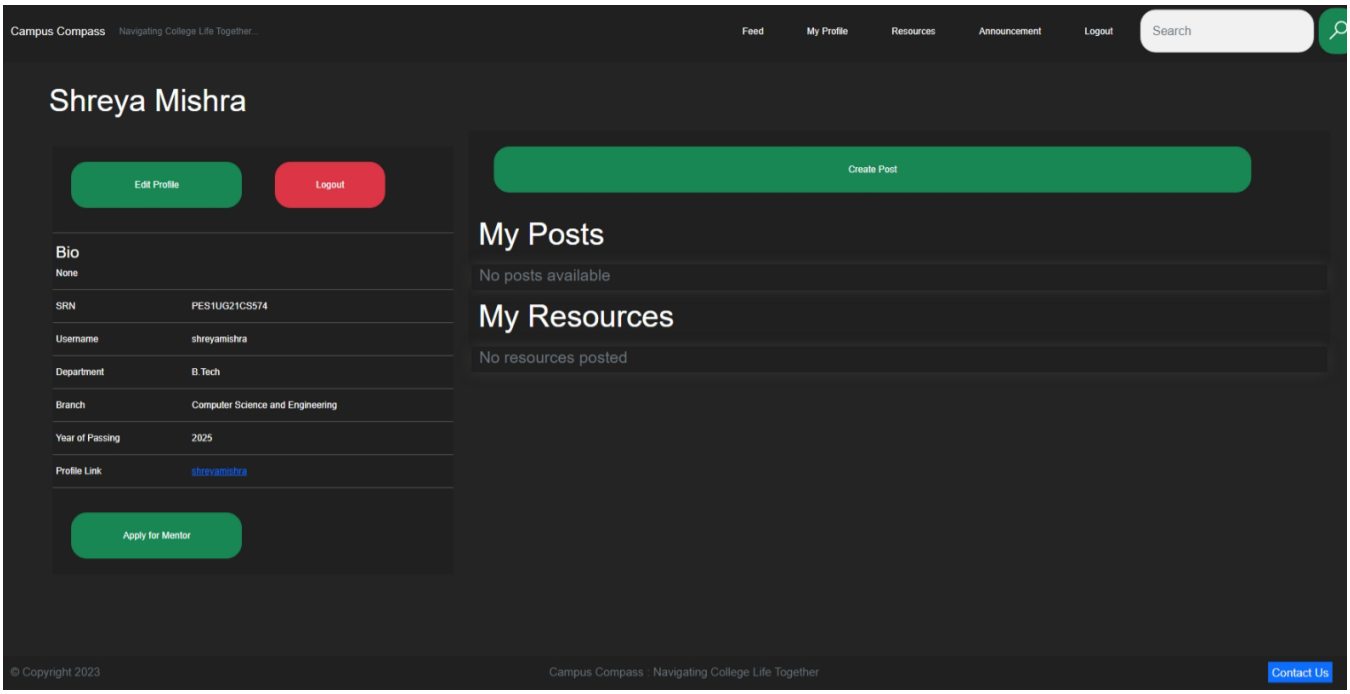

Campus Compass
Navigating College Life Together...
Feed
My Profile
Resources
Announcement
Join us
Search
Login Form
Username
Password
Login
Forgot Password? Click here | New User? Click here to register
© Copyright 2023
Campus Compass : Navigating College Life Together
Contact Us

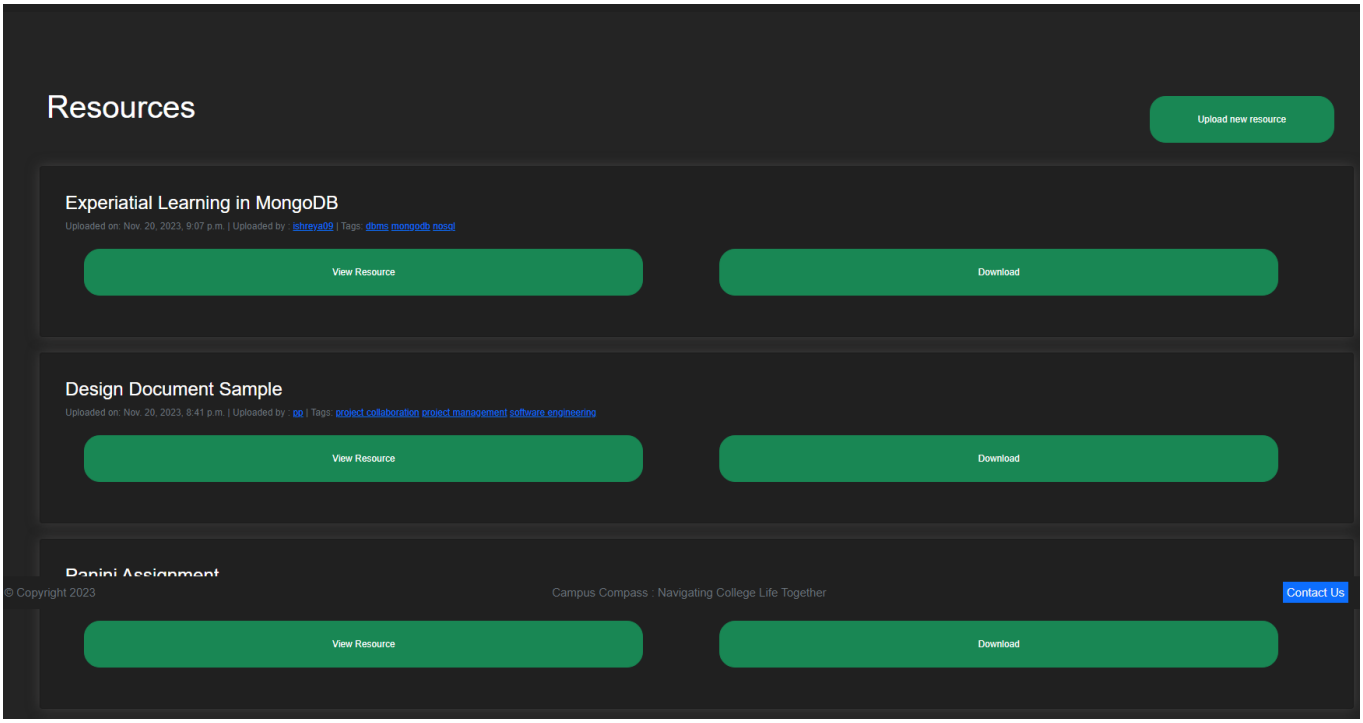

Registration Form
First Name
Last Name
Country Code
Whatsapp Number
Email id
Username
Password
Confirm Password
College Email id
SRN(Student id)
Year Of Passing Out
Department
Branch
Sign Up
Already have an account? Log In Now

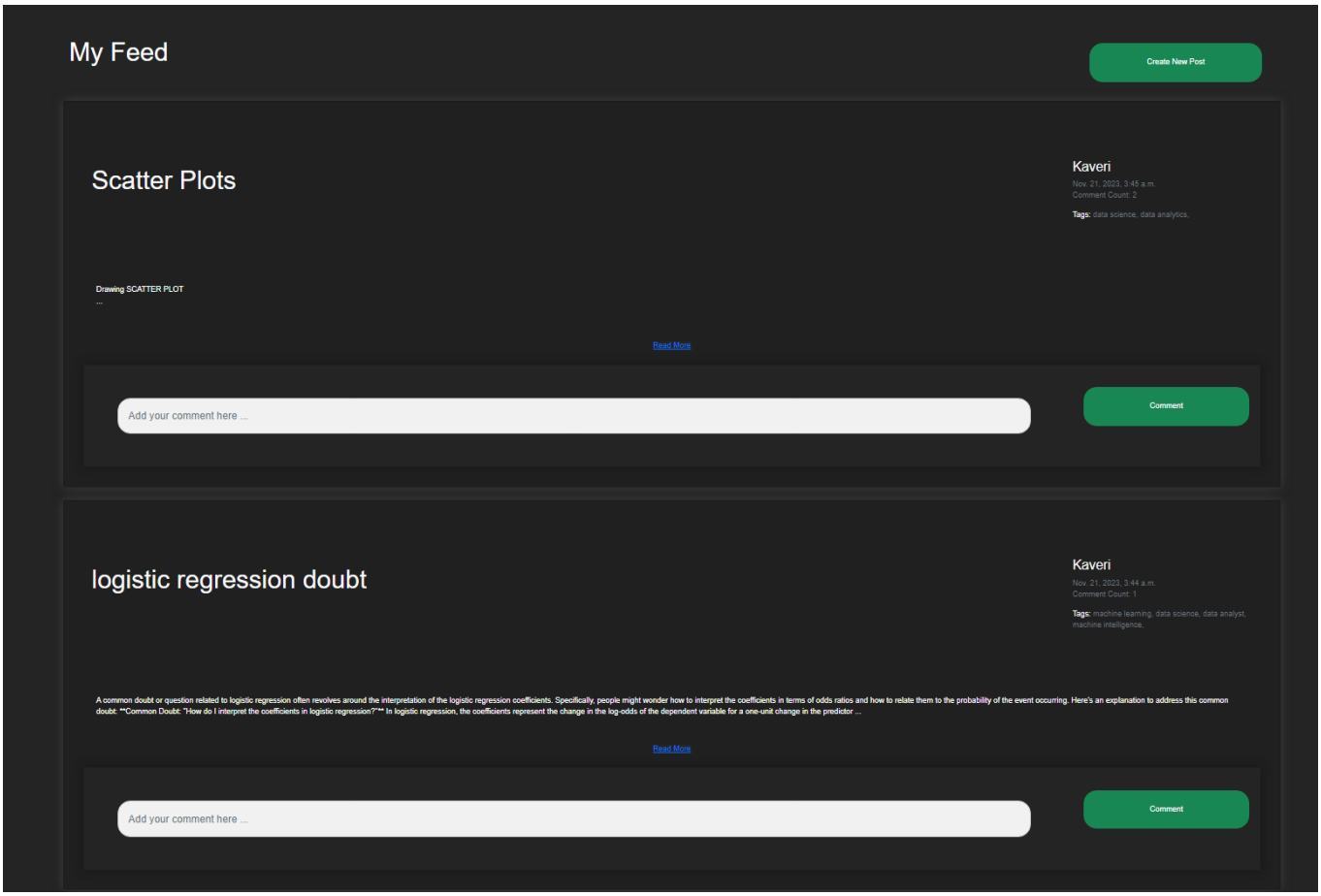

Edit Profile
Shreya
Mishra
+91
9471312854
shreyamishra414@gmail.com
Show Whatsapp Number

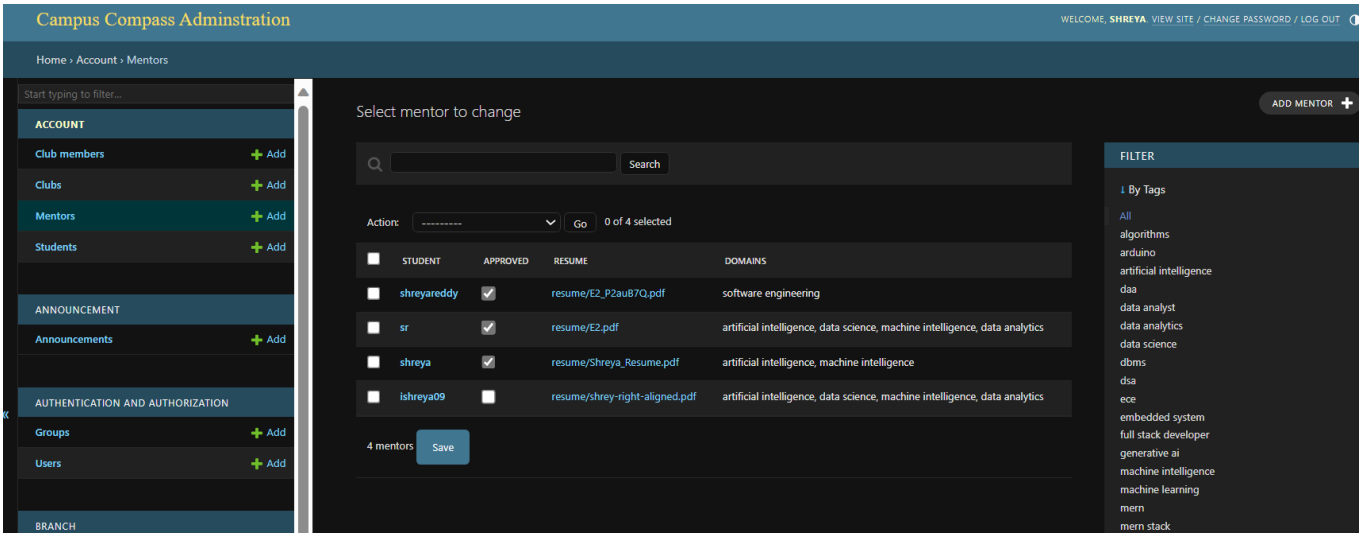

Campus Compass
Navigating College Life Together...
Feed
My Profile
Resources
Announcement
Logout
Search
Announcement
Samarpana Marathon Event
12/12/2020
Lorem ipsum dolor sit amet consectetur adipisicing elit. Inventore sit quam pariatur officia esse enim odio, molestias sint veritatis eligendi repudiandae dolor vitae laboriosam fugit autem tenetur voluptas sequi eos? Incidunt, consequatur. Voluptates nostrum ut velit iste officiis assumenda accusamus maiores iure distinctio optio provident, quae cum placeat iusto fuga.
Samarpana Marathon Event
12/12/2020
Lorem ipsum dolor sit amet consectetur adipisicing elit. Inventore sit quam pariatur officia esse enim odio, molestias sint veritatis eligendi repudiandae dolor vitae laboriosam fugit autem tenetur voluptas sequi eos? Incidunt, consequatur. Voluptates nostrum ut velit iste officiis assumenda accusamus maiores iure distinctio optio provident, quae cum placeat iusto fuga.
Samarpana Marathon Event
12/12/2020
Lorem ipsum dolor sit amet consectetur adipisicing elit. Inventore sit quam pariatur officia esse enim odio, molestias sint veritatis eligendi repudiandae dolor vitae laboriosam fugit autem tenetur voluptas sequi eos? Incidunt, consequatur. Voluptates nostrum ut velit iste officiis assumenda accusamus maiores iure distinctio optio provident, quae cum placeat iusto fuga.
© Copyright 2023
Campus Compass : Navigating College Life Together
Contact Us

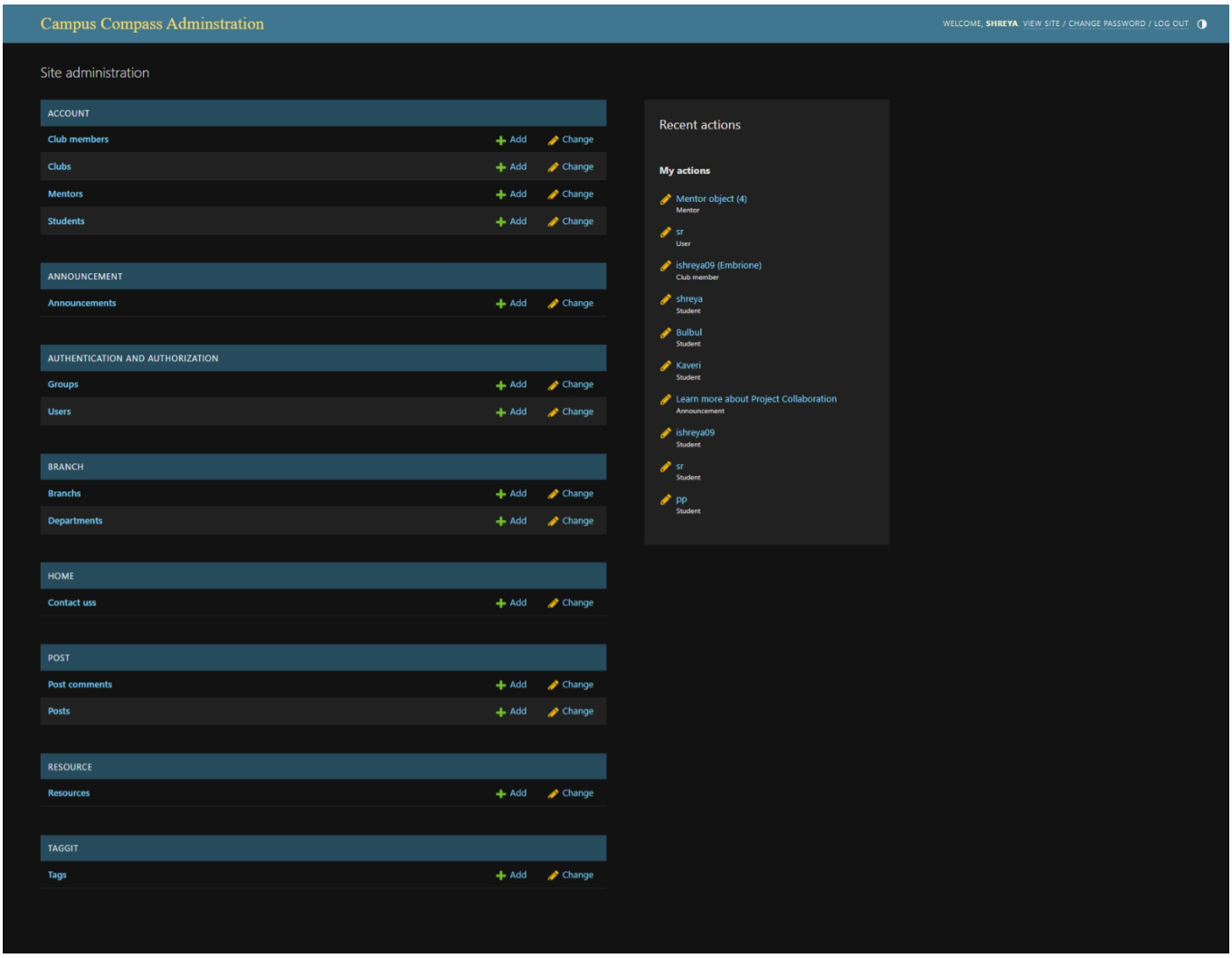

Create Post
Title:
Styles
Format
Content:
Campus Compass : Navigating College Life Together
Tags:
Branch:
Computer Science and Engineering
Electronics and Communication Engineering
Electrical and Electronics Engineering
Biotechnology
Mechanical Engineering
Civil Engineering
Computer Science and Technology(AI/ML specialization)
Computer Science and Engineering
Software Engineering
Data Science
Artificial Intelligence
Data Analytics
Network Security
Machine Learning
Embedded Systems
Communication Systems
Mechanical Engineering
Software Engineering
Database Management
Web Development
Data Science
Artificial Intelligence
Cybersecurity
Cloud Computing
Blockchain Technology
Internet of Things
Big Data Analytics
Human-Computer Interaction
General Business Administration
Entrepreneurship
Marketing
Finance
Operations Management
Human Resource Management
Digital Marketing
Digital Marketing
Management Information Systems
Submit

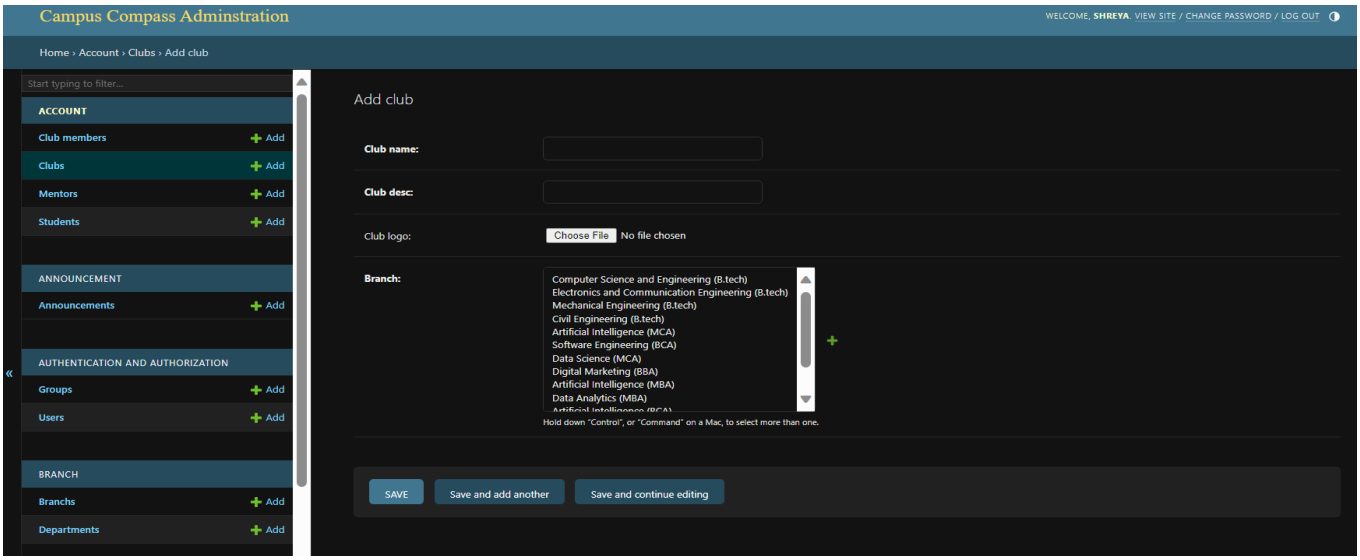

Campus Compass
Navigating College Life Together...
Feed
My Profile
Resources
Announcement
Logout
Search
Shreya Mishra
Edit Profile
Logout
Create Post
Bio
None
SRN
PES1UG21CS574
Username
shreyamishra
Department
B.Tech
Branch
Computer Science and Engineering
Year of Passing
2025
Profile Link
shreyamishra
Apply for Mentor
My Posts
No posts available
My Resources
No resources posted
© Copyright 2023
Campus Compass : Navigating College Life Together
Contact Us


Resources
Upload new resource
Experiatial Learning in MongoDB
Uploaded on: Nov. 20, 2023, 9:07 p.m. | Uploaded by : ishreya09 | Tags: dbms mongodb nosql
View Resource
Download
Design Document Sample
Uploaded on: Nov. 20, 2023, 8:41 p.m. | Uploaded by : pp | Tags: project collaboration project management software engineering
View Resource
Download
Panini Assignment
© Copyright 2023
Campus Compass : Navigating College Life Together
Contact Us
View Resource
Download

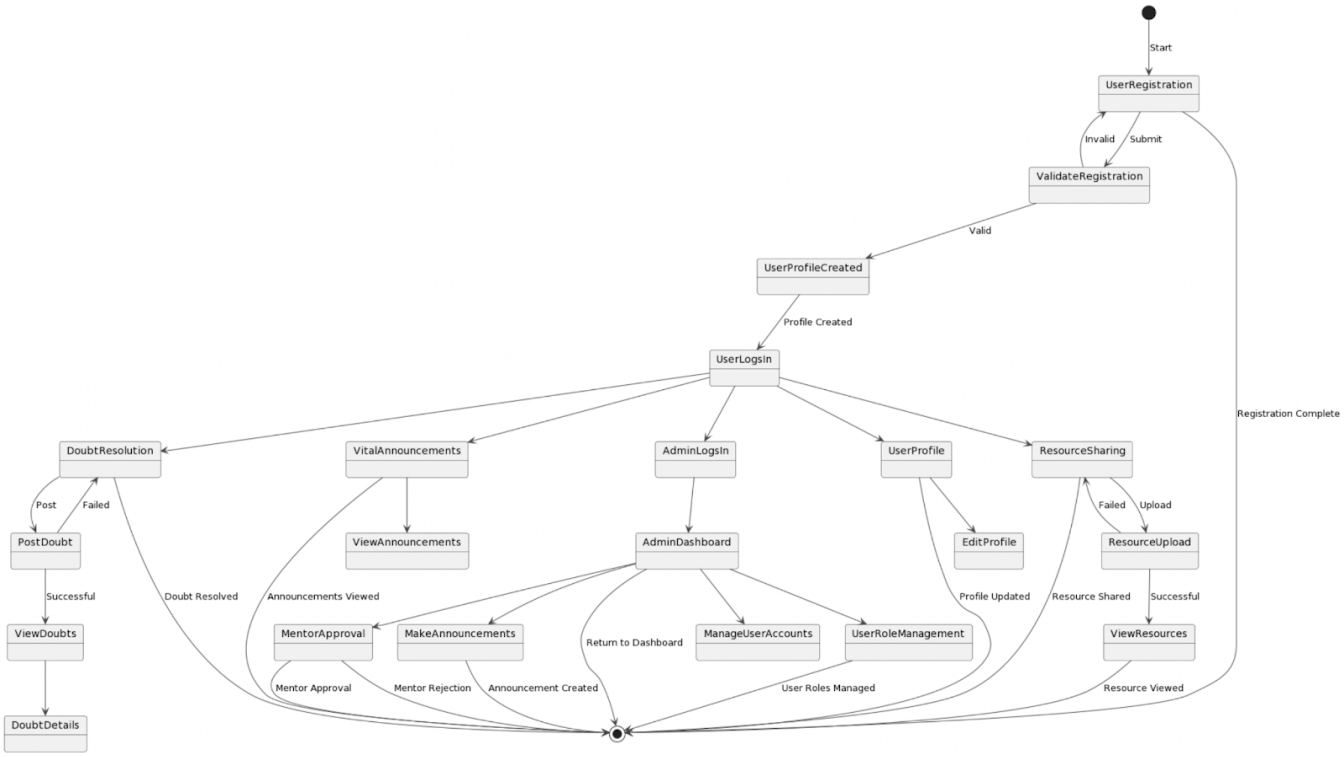

My Feed
Create New Post
Scatter Plots
Kaveri
Nov. 21, 2023, 3:45 a.m.
Comment Count: 2
Tags: data science, data analytics,
Drawing SCATTER PLOT
...
Read More
Add your comment here ...
Comment
logistic regression doubt
Kaveri
Nov. 21, 2023, 3:44 a.m.
Comment Count: 1
Tags: machine learning, data science, data analyst, machine intelligence,
A common doubt or question related to logistic regression often revolves around the interpretation of the logistic regression coefficients. Specifically, people might wonder how to interpret the coefficients in terms of odds ratios and how to relate them to the probability of the event occurring. Here's an explanation to address this common doubt: **Common Doubt: "How do I interpret the coefficients in logistic regression?"** In logistic regression, the coefficients represent the change in the log-odds of the dependent variable for a one-unit change in the predictor ...
Read More
Add your comment here ...
Comment


Campus Compass Adminstration
WELCOME, SHREYA. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Account › Mentors
Start typing to filter...
ACCOUNT
Club members + Add
Clubs + Add
Mentors + Add
Students + Add
ANNOUNCEMENT
Announcements + Add
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
BRANCH
Select mentor to change
ADD MENTOR +
Search
Action: --------- Go 0 of 4 selected
STUDENT APPROVED RESUME DOMAINS
shreyareddy resume/E2_P2auB7Q.pdf software engineering
sr resume/E2.pdf artificial intelligence, data science, machine intelligence, data analytics
shreya resume/Shreya_Resume.pdf artificial intelligence, machine intelligence
ishreya09 resume/shrey-right-aligned.pdf artificial intelligence, data science, machine intelligence, data analytics
4 mentors Save
FILTER
By Tags
All
algorithms
arduino
artificial intelligence
daa
data analyst
data analytics
data science
dbms
dsa
ece
embedded system
full stack developer
generative ai
machine intelligence
machine learning
mern
mern stack

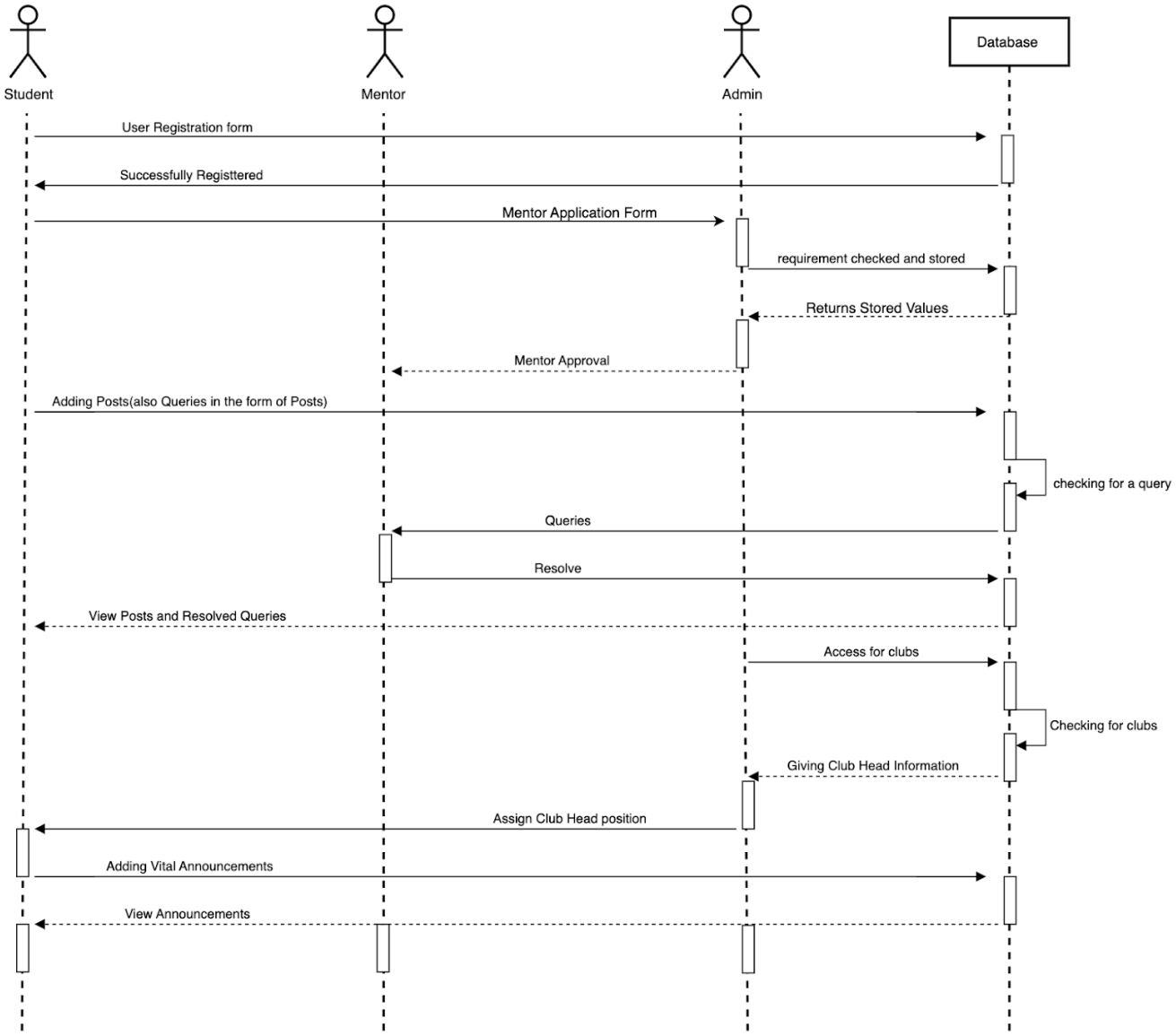

Campus Compass Adminstration
WELCOME, SHREYA. VIEW SITE / CHANGE PASSWORD / LOG OUT
Site administration
ACCOUNT
Club members Add Change
Clubs Add Change
Mentors Add Change
Students Add Change
ANNOUNCEMENT
Announcements Add Change
AUTHENTICATION AND AUTHORIZATION
Groups Add Change
Users Add Change
BRANCH
Branchs Add Change
Departments Add Change
HOME
Contact uss Add Change
POST
Post comments Add Change
Posts Add Change
RESOURCE
Resources Add Change
TAGGIT
Tags Add Change
Recent actions
My actions
Mentor object (4)
Mentor
sr
User
ishreya09 (Embrione)
Club member
shreya
Student
Bulbul
Student
Kaveri
Student
Learn more about Project Collaboration
Announcement
ishreya09
Student
sr
Student
pp
Student


Campus Compass Adminstration
WELCOME, SHREYA. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Account › Clubs › Add club
Start typing to filter...
ACCOUNT
Club members Add
Clubs Add
Mentors Add
Students Add
ANNOUNCEMENT
Announcements Add
AUTHENTICATION AND AUTHORIZATION
Groups Add
Users Add
BRANCH
Branchs Add
Departments Add
Add club
Club name:
Club desc:
Club logo:
Choose File
No file chosen
Branch:
Computer Science and Engineering (B.tech)
Electronics and Communication Engineering (B.tech)
Mechanical Engineering (B.tech)
Civil Engineering (B.tech)
Artificial Intelligence (MCA)
Software Engineering (BCA)
Data Science (MCA)
Digital Marketing (BBA)
Artificial Intelligence (MBA)
Data Analytics (MBA)
Hold down "Control", or "Command" on a Mac, to select more than one.
SAVE
Save and add another
Save and continue editing

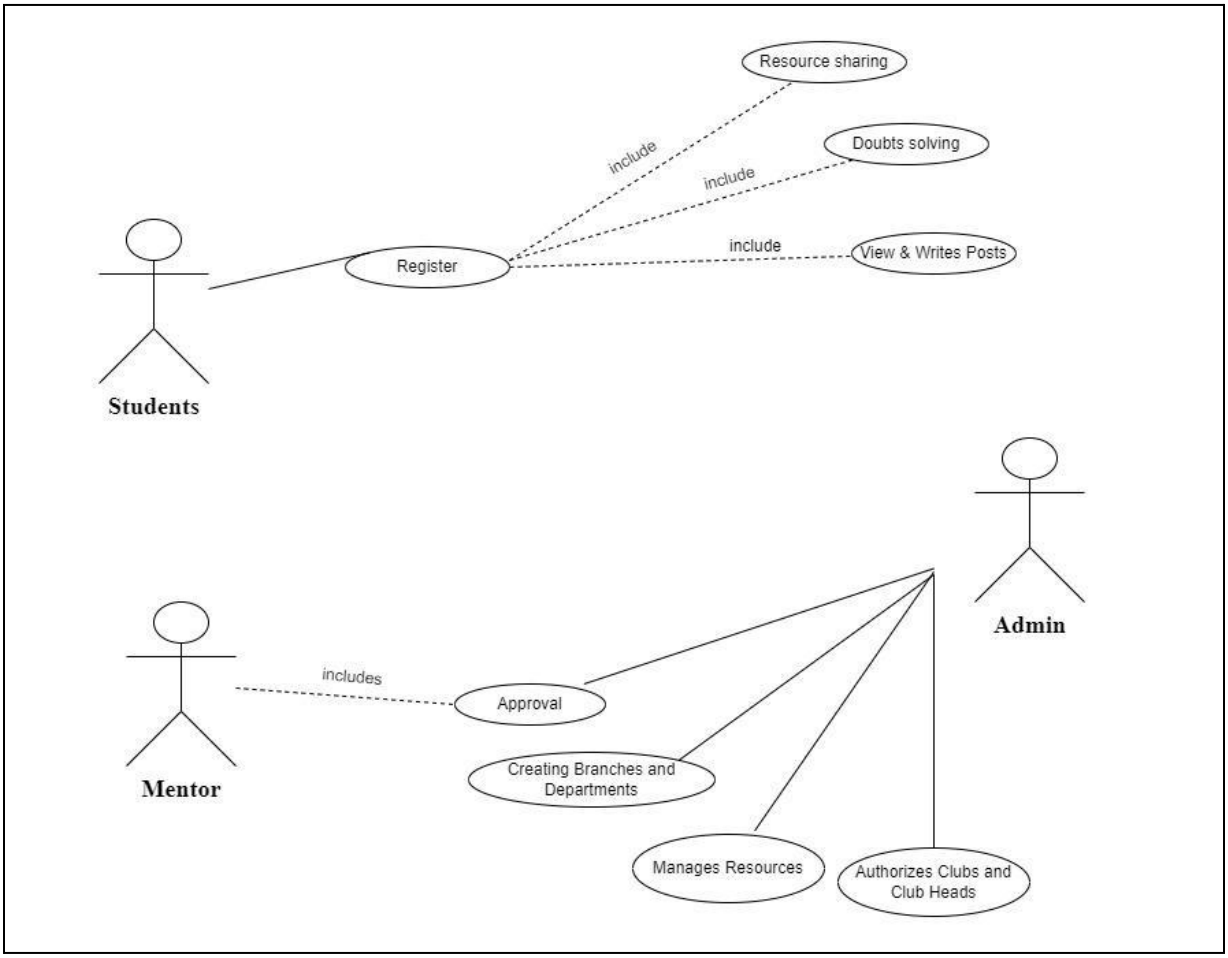
State Diagram(DR-003)



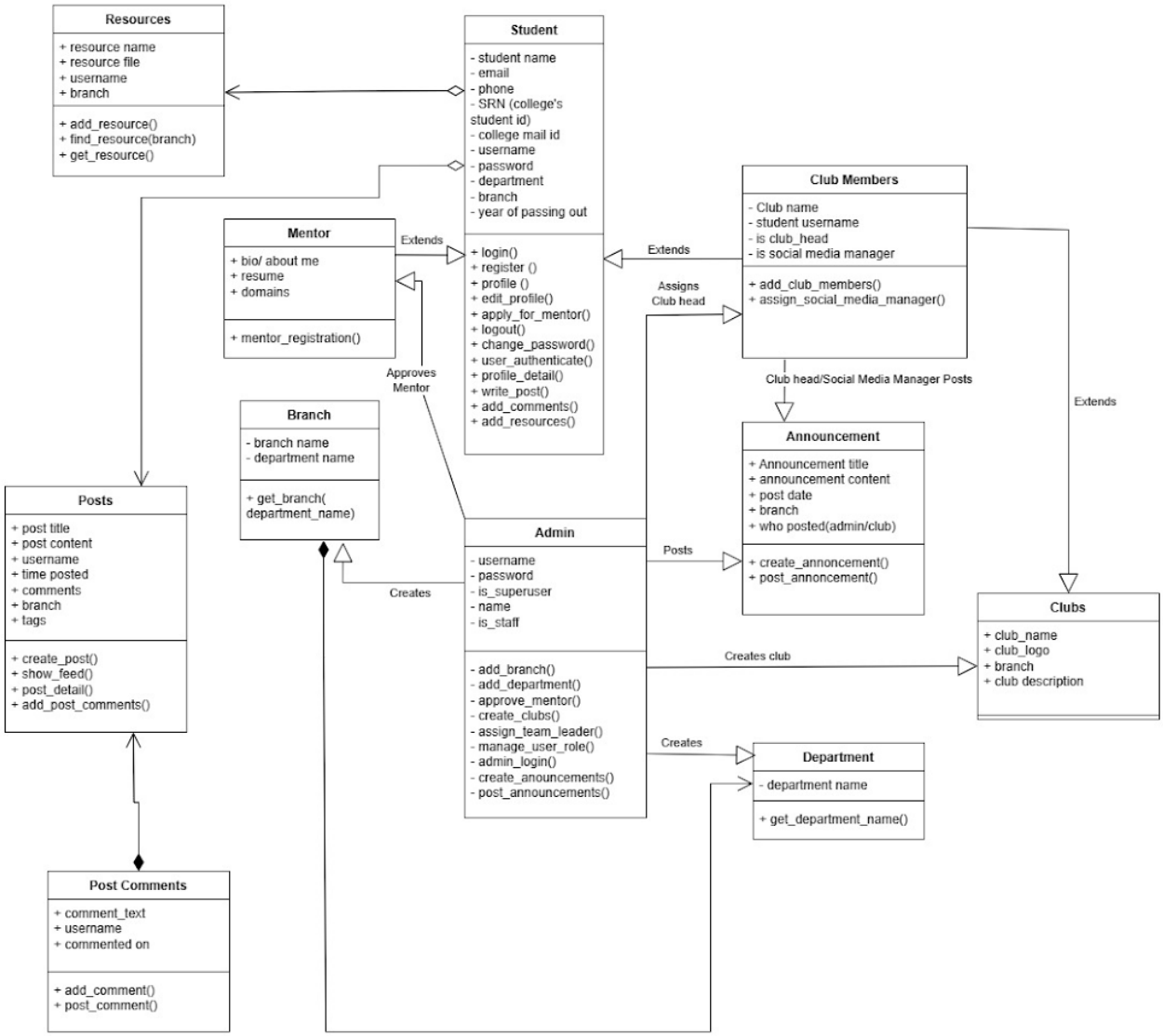
Sequence Diagrams(DR-004)



Use Case Diagram(DR-005)



Class Diagram(DR-006)



5. Development - Code Files

Repository Link
https://github.com/ishreya09/Campus-Compass

Project Structure
Frontend:
The frontend is organized using the Django app structure. In django, it is handled through assets, templates and static files which contain HTML, CSS, JavaScript and Images.

Backend:
Backend logic follows Django's model-view-template (MVT) architecture.

Model (M):
Represents the data structure and business logic of the application.
In Django, models define the structure of the database and encapsulate data-related logic.
Models are responsible for interacting with the database, performing CRUD operations, and validating data.

View (V):
Handles the presentation logic and user interface.
In Django, views receive user requests, process them, and return appropriate responses.
Views in Django are responsible for rendering templates, handling user input, and interacting with models to fetch or store data.

Template (T):
All the templates can be found here along with base.html :
https://github.com/ishreya09/Campus-Compass/tree/master/CampusCompass/templates

Represents the presentation layer in Django, responsible for generating HTML markup.
Templates define how data from the views should be displayed.
They provide a way to separate HTML structure from Python code in views.

Database Models:
Django models are used to define the database schema.

Static Files:

Static files such as CSS, JavaScript, and images are organized in the static directory.

All the static files can be found here:
https://github.com/ishreya09/Campus-Compass/tree/master/CampusCompass/asset

Testing:
Unit tests and integration tests are organized in the testing directory.

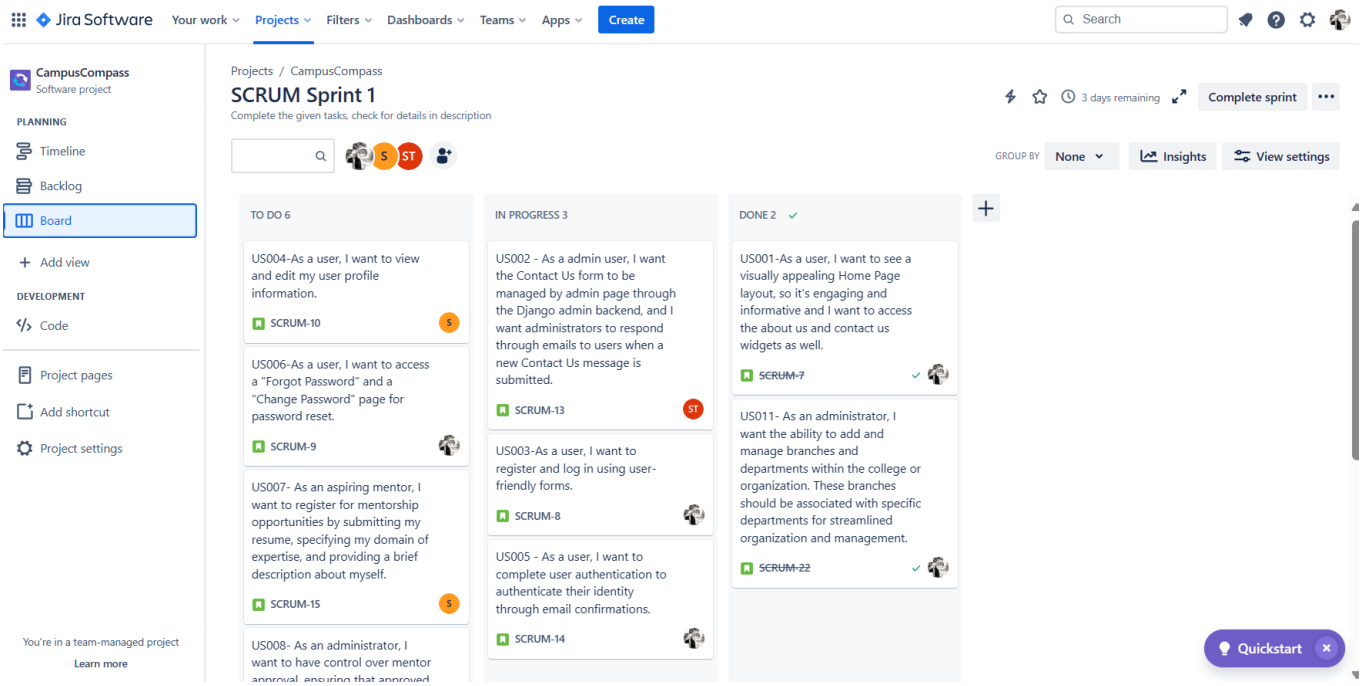
Configuration Files:
Key configurations are managed in Django settings files.

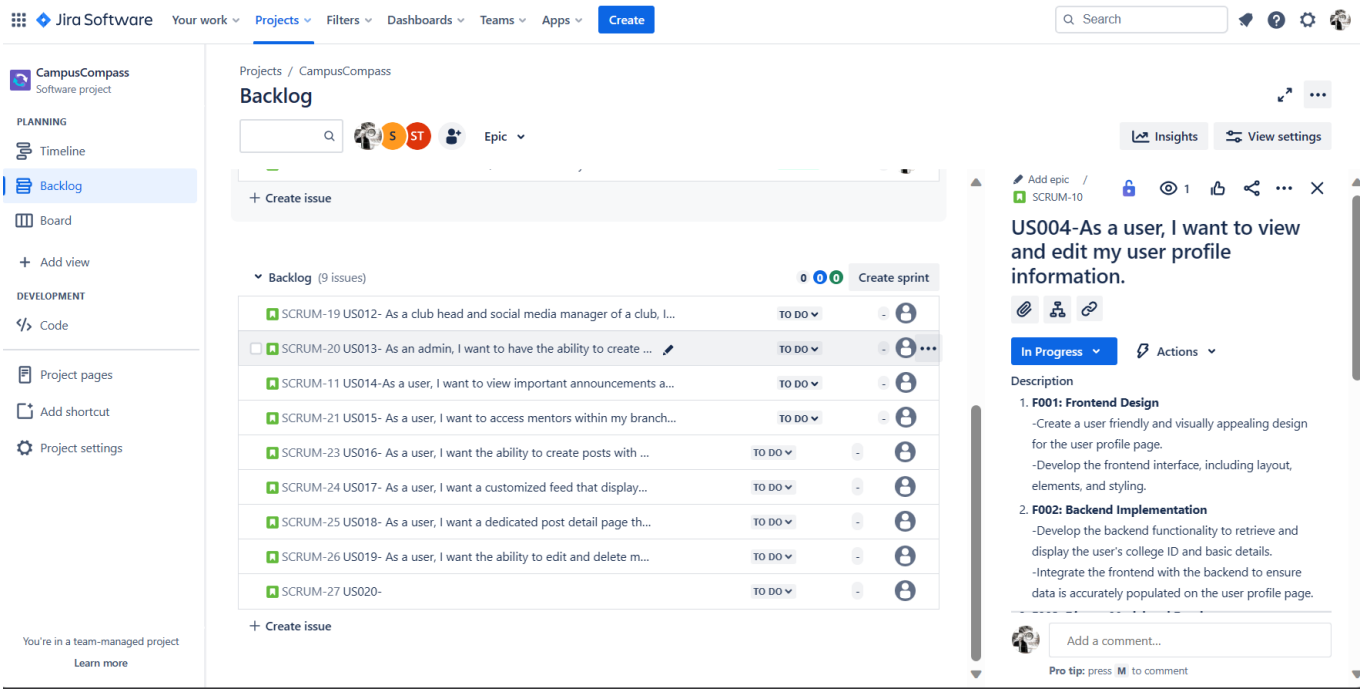
```
"""
Django settings for CampusCompass project.

Generated by 'django-admin startproject' using Django 4.2.6.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/ref/settings/
"""

from pathlib import Path
import os
from decouple import config # for protection of sensitive information


# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = config('SECRET_KEY')

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = config('DEBUG')


ALLOWED_HOSTS = ['*']


# Application definition

INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",

    # User Installed Apps
    "home.apps.HomeConfig",
    "post.apps.PostConfig",
    "account.apps.AccountConfig",
    "branch.apps.BranchConfig",
    "resource.apps.ResourceConfig",
    "announcement.apps.AnnouncementConfig",
    "search.apps.SearchConfig",

    # helper apps
    'taggit',
    'ckeditor',
    'ckeditor_uploader',
]

MIDDLEWARE = [
    "django.middleware.security.SecurityMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
    # 'django.contrib.staticfiles.middleware.StaticFilesMiddleware',
]
```

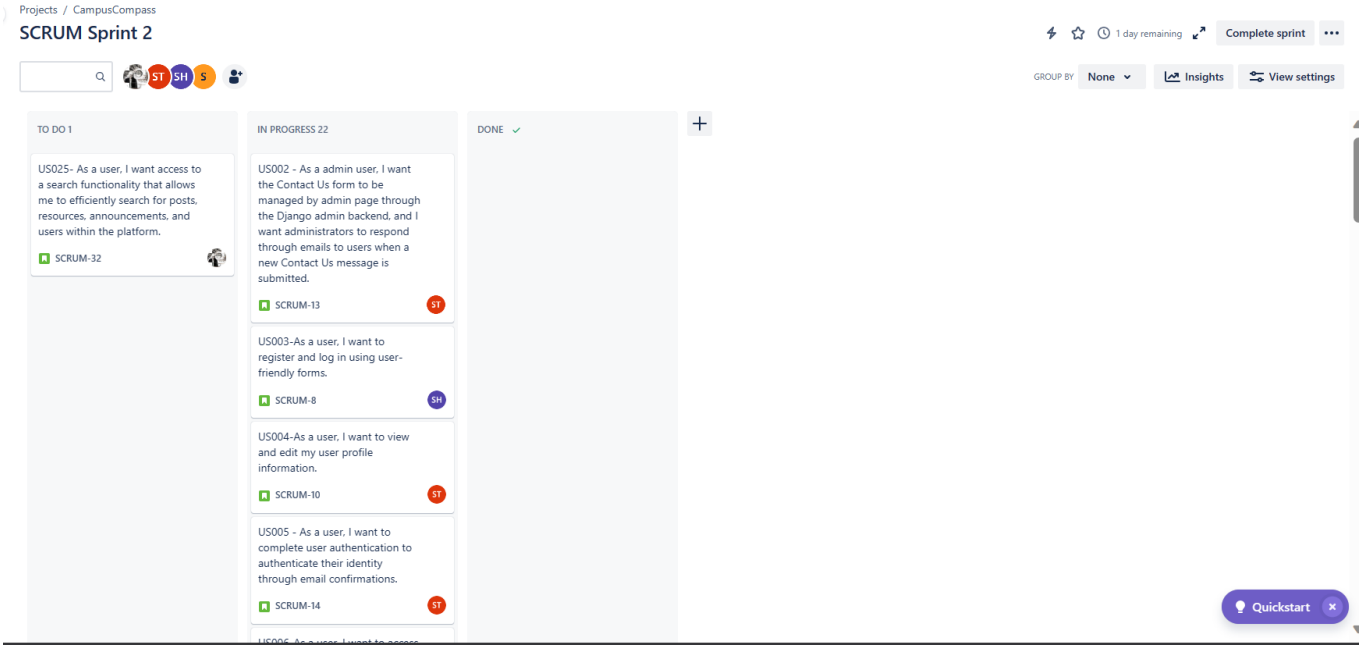
```
ROOT_URLCONF = "CampusCompass.urls"

TEMPLATES = [
    {
        "BACKEND": "django.template.backends.django.DjangoTemplates",
        "DIRS": [os.path.join(BASE_DIR,'templates')],
        "APP_DIRS": True,
        "OPTIONS": {
            "context_processors": [
                "django.template.context_processors.debug",
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",
                "django.contrib.messages.context_processors.messages",
            ],
        },
    },
]

WSGI_APPLICATION = "CampusCompass.wsgi.application"


# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases


# DATABASES = {
#     "default": {
#         "ENGINE": "django.db.backends.mysql",
#         "NAME": config('DB_NAME'),
#         "USER": config('DB_USER'),
#         "PASSWORD": config('DB_PASSWORD'),
#         "HOST": config('DB_HOST'),
#         "PORT": config('DB_PORT'),
#     }
# }
```

```
STATIC_URL = "/static/"
STATICFILES_DIRS=[
    os.path.join(BASE_DIR , 'static/'),

]
STATIC_ROOT = os.path.join(BASE_DIR, 'asset/')


MEDIA_URL ='/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')


# SMTP setup
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_USE_TLS = config('EMAIL_USE_TLS')
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = config('EMAIL_PORT')
EMAIL_HOST_USER = config('EMAIL_HOST_USER')
EMAIL_HOST_PASSWORD = config('EMAIL_HOST_PASSWORD')

CKEDITOR_UPLOAD_PATH= "ckeditor_upload/"
CKEDITOR_JQUERY_URL = '//ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js'  # Specify jQuery URL if necessary
CKEDITOR_CONFIGS = {
    'default': {
        'toolbar': [
                    {'name': 'clipboard', 'items': ['Cut', 'Copy', 'Paste', 'PasteText', 'PasteFromWord', '-', 'Undo', 'Redo']},
                    {'name': 'basicstyles', 'items': ['Bold', 'Italic', 'Underline', 'Strike', 'Subscript', 'Superscript', '-', 'RemoveFormat']},
                    {'name': 'paragraph', 'items': ['NumberedList', 'BulletedList', '-', 'Outdent', 'Indent', '-', 'Blockquote']},
                    {'name': 'links', 'items': ['Link', 'Unlink', 'Anchor']},
                    {'name': 'insert', 'items': ['Image', 'Table', 'HorizontalRule', 'SpecialChar']},
                    '/',
                    {'name': 'styles', 'items': ['Styles', 'Format']},
                    {'name': 'colors', 'items': ['TextColor', 'BGColor']},
                ],
        'height':600,
        'width':1200,
    },
}


# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        "NAME": "django.contrib.auth.password_validation.UserAttributeSimilarityValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.MinimumLengthValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.CommonPasswordValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.NumericPasswordValidator",
    },
]


# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE_CODE = "en-us"

TIME_ZONE = "UTC"

USE_I18N = True

USE_TZ = True



# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"

# TESTS = {
#     'CLIENT': {
#         'TIMEOUT': 100,  # Adjust the timeout value as needed
#     },
# }
```

Jira Work Management







Features and Development

User Registration and Authentication - High Priority (CR001)

```
def register(request):
    department= Department.objects.all()
    context={'department':department}
    return render(request, 'account/register.html',context)


def register_submit(request):
    context={}
    if request.method=='POST':
        firstname = request.POST.get('firstname',False)
        lastname = request.POST.get('lastname',False)
        country_code= request.POST.get('countrycode',False)
        whatsappnumber = request.POST.get('phone',False)
        email = request.POST.get('email',False)
        username = request.POST.get('username',False)
        password1 = request.POST.get('password1',False)
        password2 = request.POST.get('password2',False)
        college_email = request.POST.get('collegemail',False)
        SRN = request.POST.get('SRN',False) # student id
        year_of_passing_out = request.POST.get('yearofpassingout',False)
        branch = request.POST.get('branch',False)
        department = request.POST.get('department',False)

        whatsappno =country_code+" "+whatsappnumber

        # trim + from country code if any
        if (country_code[0] == "+"):
            country_code = country_code[1:]

        whatsappnumber=country_code+whatsappnumber


        whatsapplink="https://api.whatsapp.com/send?phone="+whatsappnumber

        if password1 == password2:
            if User.objects.filter(username= username).exists():
                messages.info (request, "Username taken")
                return redirect('/account/register')
            elif (User.objects.filter(email = email).exists()):
                messages.info (request, "Email already present")
                return redirect('/account/register')
            elif (Student.objects.filter(college_email=college_email).exists()):
                messages.info (request, "Account already present")
                return redirect('/account/register')
            else:
                user = User.objects.create_user(first_name= firstname, last_name= lastname ,username= username, password= password1, email= email)
                user.save()
```

```
                student= Student()
                student.user=User.objects.get(username=username)
                student.department= Department.objects.get(department_id= department)
                student.branch= Branch.objects.get(branch_code= branch)

                data=("",whatsappno,whatsapplink,SRN,college_email,year_of_passing_out)
                student.bio,student.whatsapp_number,student.whatsapp_link,student.student_id,student.college_email,student.year_of_passing_out=data

                student.save()
                auth.login(request, user)


                messages.success(request, 'Verify Your Account now!!')
                return redirect ('/account/login')
        else:
            messages.danger (request, "password does not match")
            return redirect('/account/register')

    return redirect ('/account/register')




def login(request):
    context={}
    return render(request, 'account/login.html',context)

def login_submit(request):
    context={}
    if request.method == 'POST':
        username = request.POST.get('username',False)
        password = request.POST.get('password',False)

        user = auth.authenticate(username=username, password= password)
        if user is not None:
            auth.login(request, user)
            messages.success(request, 'You are logged in successfully')
            return redirect('/account/profile')
        else:
            messages.error(request, 'invalid username or password')

            return redirect ('/account/login')
    else:
        pass

def logout(request):
    auth.logout(request)
    messages.success(request,"You are logged out")
    return redirect('/')
```

```
def confirm_email(request):
    user=request.user
    current_site = get_current_site(request)
    subject = 'Activate Your College Connect Account'
    print(user)
    message = render_to_string('account/account_activation_email.html', {
                'user': user,
                'domain': current_site.domain,
                'uid': urlsafe_base64_encode(force_bytes(user.pk)),
                'token': account_activation_token.make_token(user),
            })
    user.email_user(subject, message)

    messages.success(request, ('Your email confirmation is pending! Verify now'))
    return redirect('/')


class ActivateAccount(View):

    def get(self, request, uidb64, token, *args, **kwargs):
        try:
            # uid = force_text(urlsafe_base64_decode(uidb64))
            uid=str(urlsafe_base64_decode(uidb64), 'utf-8')
            user = User.objects.get(pk=uid)
        except (TypeError, ValueError, OverflowError, User.DoesNotExist):
            user = None

        if user is not None and account_activation_token.check_token(user, token):
            user.is_active = True
            user.student.email_confirmed = True
            student=Student.objects.get(user=user)
            student.email_confirmed=True
            student.save()
            user.save()
            auth.login(request, user)
            messages.success(request, ('Your email has been confirmed.'))
            return redirect('/')
        else:
            messages.warning(request, ('The confirmation link was invalid, possibly because it has already been used.'))
            return redirect('/')
```

User Profile Management - High Priority (CR002)

```
@login_required(login_url='/account/login')
def profile(request):
    username = request.user
    if username.is_anonymous:
        messages.error(request, "Please sign in first")
        return redirect('/account/login')

    # Retrieve user profile data using the joins query
    user_profile = (
        User.objects
        .filter(username=username)
        .select_related('student')  # Perform a join for the student
        .prefetch_related(
            'student__mentor', 'student__branch'
        )  # Perform joins for mentor, club members, and student branches
        .first()  # Retrieve the first matching user (you can adjust the query as needed)
    )

    if user_profile:
        user = user_profile
        student = user.student
        mentor = user_profile.student.mentor if user_profile.student.is_mentor else None

        club_head = is_club_head(username)
        social_media_manager = is_social_media_manager(username)
        club_member = is_club_member(username)
        club_names=[]
        # Check which clubs the user is in
        if(is_club_member):
            club_members = ClubMember.objects.filter(user=user)
            club_names = [club.club.club_name for club in club_members]

        # Filter posts by the user's branch
        posts = Post.objects.filter(user=user)

        # Sort posts by the most recent
        posts = posts.order_by('-created_at')

        resources = Resource.objects.filter(user=user)
        resources = resources.order_by('-uploaded_at')

        context = {
            'user': user,
            'student': student,
            'mentor': mentor,
            'club_head': club_head,
            'social_media_manager': social_media_manager,
            'club_member': club_member,
            'clubs': club_names,
            'post': posts,
            'resources': resources,
        }

        return render(request, 'account/profile.html', context)
    else:
        messages.error(request, "User profile not found.")
        return redirect('/account/login')
```

```
@login_required(login_url='/account/login')
def profile_user(request,username):
    who=request.user.__str__()
    if(who=="AnonymousUser"):
        messages.error(request,"Please sign in first")
        return redirect("/account/login" )
    print(username)
    user_profile = (
        User.objects
        .filter(username=username)
        .select_related('student')  # Perform a join for the student
        .prefetch_related(
            'student__mentor', 'student__branch'
        )  # Perform joins for mentor, club members, and student branches
        .first()  # Retrieve the first matching user (you can adjust the query as needed)
    )

    if user_profile:
        user = user_profile
        student = user.student
        mentor = user_profile.student.mentor if user_profile.student.is_mentor else None

        club_head = is_club_head(username)
        social_media_manager = is_social_media_manager(username)
        club_member = is_club_member(username)

        # Check which clubs the user is in
        if(is_club_member):
            club_members = ClubMember.objects.filter(user=user)
            club_names = [club.club.club_name for club in club_members]

        # Filter posts by the user's branch
        posts = Post.objects.filter(user=user)

        # Sort posts by the most recent
        posts = posts.order_by('-created_at')

        resources = Resource.objects.filter(user=user)
        resources = resources.order_by('-uploaded_at')

        context = {
            'user': user,
            'student': student,
            'mentor': mentor,
            'club_head': club_head,
            'social_media_manager': social_media_manager,
            'club_member': club_member,
            'clubs': club_names,
            'post': posts,
            'resources': resources,
        }

        return render(request, 'account/profile_detail.html', context)
    else:
```

```
@login_required(login_url='/account/login')
def change_password(request):
    context ={}
    return render(request, 'account/change_password.html',context)

@login_required(login_url='/account/login')
def edit_profile(request):
    username= request.user
    if(username.__str__() != 'AnonymousUser'):
        department=Department.objects.all()
        user= User.objects.get(username=username)
        student= Student.objects.get(user=user)
        branch=Branch.objects.filter(department=student.department)

        # print(branch)
        # print(student.branch.branch_code)
        context ={'department':department,'user':user,'student':student,'branch':branch}
        return render(request, 'account/editprofile.html',context)
    else:
        messages.error(request,"Please sign in first")
        return redirect('/account/login')
```

Resource Sharing - Medium Priority (CR003)

```
login_required(login_url='/account/login')
def resource(request):
    username=request.user.__str__()
    if(username=='AnonymousUser'):
        return redirect("/account/login")
    print(request.user.student.branch)
    user_branch = request.user.student.branch
    # Filter resources by the user's branch
    resource = Resource.objects.filter(branch= user_branch)
    # sort wrt to most recent
    resource = resource.order_by('-uploaded_at')
    context={'resources':resource}
    return render(request, 'resource/resource.html',context)

login_required(login_url='/account/login')
def resource_by_tag(request,tag):
    username=request.user.__str__()
    try:
        # get tag
        tag = Tag.objects.get(slug=tag)
        # get resources
        resource = Resource.objects.filter(tags__name__in=[tag])
        # sort wrt to most recent
        resource = resource.order_by('-uploaded_at')
        context={'resources':resource}

        return render(request, 'resource/resource.html',context)
    except:
        messages.error(request, "No resources found for this tag.")
        return redirect('/error404')
```

```
login_required(login_url='/account/login')
def upload_resource(request):
    username=request.user.__str__()
    if request.method == 'POST':
        form = ResourceForm(request.POST, request.FILES)
        if form.is_valid():
            # get branches
            branches = form.cleaned_data['branch']
            # get tags
            tags = form.cleaned_data['tags']
            print(branches,tags)
            # get title
            title = form.cleaned_data['title']

            r= form.save(commit=False)
            r.user = User.objects.get(username=username)
            r.save()
            # add branches
            for branch in branches:
                r.branch.add(branch)
            # add tags
            for tag in tags:
                tag = tag.strip().lower()
                domain, created = Tag.objects.get_or_create(name=tag)
                r.tags.add(tag)

            messages.success(request, "Resource successfully added!")
            return redirect('/resource')
        else:
            messages.error(request, "Error in form submission. Please correct the errors below.")

    else:
        form = ResourceForm()
    return render(request, 'resource/upload_resource.html', {'form': form,"name":"Upload Resources"})
```

```
login_required(login_url='/account/login')
def edit_resource(request,id=None):
    username=request.user.__str__()
    # get resource
    resource = Resource.objects.get(id=id)
    # get user
    user=User.objects.get(username=username)

    if request.method == 'POST':
        form = ResourceForm(request.POST, request.FILES)
        if form.is_valid() and resource.user==user:
            # get branches
            branches = form.cleaned_data['branch']
            # get tags
            tags = form.cleaned_data['tags']
            print(branches,tags)

            resource.title = form.cleaned_data['title']
            # check if files
            if form.cleaned_data['files']:
                resource.files = form.cleaned_data['files']
            resource.save()
            for branch in branches:
                resource.branch.add(branch)
            resource.tags.clear()
            for tag in tags:
                tag = tag.strip().lower()
                domain, created = Tag.objects.get_or_create(name=tag)
                resource.tags.add(tag)

            messages.success(request, "Resource successfully updated!")
            return redirect('/resource')
        else:
             # Check if 'files' field is empty
            if 'files' not in form.cleaned_data:
                branches = form.cleaned_data['branch']
                # get tags
                tags = form.cleaned_data['tags']
                # No new file provided, retain the existing file
                resource.title = form.cleaned_data['title']
                resource.save()
                for branch in branches:
                    resource.branch.add(branch)
                resource.tags.clear()
                for tag in tags:
                    tag = tag.strip().lower()
                    domain, created = Tag.objects.get_or_create(name=tag)
                    resource.tags.add(tag)

                messages.success(request, "Resource successfully updated!")
                return redirect('/resource')
            else:
                messages.error(request, "Error in form submission. Please correct the errors below.")

    else:
        form = ResourceForm(instance=resource)
    return render(request, 'resource/upload_resource.html', {'form': form,"name":"Edit Resources"})

login_required(login_url='/account/login')
def delete_resource(request,id=None):
    username=request.user.__str__()
    # get resource
    resource = Resource.objects.get(id=id)
    # get user
    user=User.objects.get(username=username)
    if resource.user==user:
        resource.delete()
        messages.success(request, "Resource successfully deleted!")
        return redirect('/resource')
```

```
    else:
        messages.error(request, "You are not authorized to delete this resource.")
        return redirect('/resource')

login_required(login_url='/account/login')
def download_resource(request,id):
    username=request.user.__str__()
    resource = Resource.objects.get(id=id)
    # Open and read the file
    with open(resource.files.path, 'rb') as file:
        response = HttpResponse(file.read(), content_type='application/octet-stream')

    # Set the Content-Disposition header to suggest a filename for the downloaded file
    response['Content-Disposition'] = f'attachment; filename="{resource.files.name}"'

    return response
```

Mentorship and Guidance - High Priority (CR004)

```
def can_apply_again(mentor):
    six_months_ago = timezone.now() - timezone.timedelta(days=180)  # 6 months = 180 days
    if mentor.last_application_date < six_months_ago:
        # get mentor
        mentor = Mentor.objects.filter(user=mentor).first()
        mentor.last_application_date = None
        mentor.save()

@login_required(login_url='/account/login')
def mentor_registration(request):
    username=request.user.__str__()
    if(username=="AnonymousUser"):
        messages.error(request,"Please sign in first")
        return redirect("/account/login" )
    if request.method == "POST":

        print(request.FILES)

        if  'fileupload' in request.FILES:
            resume = request.FILES['fileupload']
            domains_input = request.POST.get("domains")
            domain_list = domains_input.split(",")  # Assuming domains are comma-

            user=User.objects.get(username=username)
            student = Student.objects.get(user=user)
            if (Mentor.objects.filter(student=student).exists()):
                mentor=Mentor.objects.get(student=student)
            else:
                mentor = Mentor()

            mentor.student=student
            mentor.approved=0
            mentor.resume = resume
            mentor.description = request.POST.get("description")
            mentor.last_application_date= timezone.now()
            mentor.save()
            # mentor.domain.add(*domain_list)
```

```
            for domain_name in domain_list:
                # strip domain of spaces and make it in lower case only
                domain_name = domain_name.strip().lower()
                domain, created = Tag.objects.get_or_create(name=domain_name)
                mentor.domain.add(domain)

            messages.success(request,"Role for mentor successfully applied!, We will get back to you in 1 week")
            return redirect('/account/profile')  # Redirect to a success page or any desired location

        else:
            messages.error(request,"Add your resume first!")
            return redirect('/account/mentor_registration')

    context={}
    user=User.objects.get(username=username)
    student = Student.objects.get(user=user)
    if Mentor.objects.filter(student=student).exists():
        mentor=Mentor.objects.get(student=student)
        can_apply_again(mentor)
        context['mentor']=mentor
    return render(request, 'account/mentor_registration.html',context)  # Render the form again if it's a GET request
```

Doubt Resolution - Medium Priority (CR005)

```
class PostQuerySet(models.QuerySet):
    def search(self, query=None):
        qs = self
        if query is not None:
            or_lookup = (
                Q(title__icontains=query) |
                Q(content__icontains=query) |
                Q(author__icontains=query) |
                Q(branch__branch_name__icontains=query) |
                Q(tags__name__icontains=query)
            )
            qs = qs.filter(or_lookup).distinct().only(
                'post_id', 'title', 'slug', 'content', 'author', 'created_at', 'branch__branch_name','tags__name'
            )  # Add more fields as needed
        return qs


# Create your models here.

class Post(models.Model):
    user=  models.ForeignKey(User,on_delete=models.CASCADE)
    post_id= models.BigAutoField(primary_key=True,auto_created=True,serialize=False)
    title = models.CharField(max_length=500)
    slug = models.SlugField(unique=True,auto_created=True)
    content= RichTextUploadingField(blank=True,null=True)
    author= models.CharField(max_length=500,default="Anonymous")
    branch = models.ManyToManyField(Branch,verbose_name='branches')
    tags=TaggableManager()
    created_at=models.DateTimeField(auto_now_add=True)
    objects = PostQuerySet.as_manager()

    def __str__(self):
        return self.title
    def get_model_name(self):
        return 'Post'

class PostComment(models.Model):
    comment_id= models.BigAutoField(primary_key=True,auto_created=True,serialize=False)
    user= models.ForeignKey(User,on_delete=models.CASCADE)
    comment= models.CharField(max_length=5000, blank=True,null=True)
    created_at= models.DateTimeField(auto_now_add=True)
    post= models.ForeignKey(Post,on_delete=models.CASCADE)
    def __str__(self):
        return self.user
```

```
def create_slug(username,title):
    # remove all special character except space
    slug = re.sub(r'[^a-zA-Z0-9\s]', '', title)
    # replace all space with -
    slug= username+" "+slug
    slug = slug.lower()
    slug = re.sub(r'\s', '-', slug)
    return slug

@login_required(login_url='/account/login')
def make_post(request):
    username=request.user.__str__()

    if request.method == 'POST':
        form = PostForm(request.POST, request.FILES)
        if form.is_valid():
            # get branches
            branches = form.cleaned_data['branch']
            # get tags
            tags = form.cleaned_data['tags']
            print(branches,tags)
            # get title
            title = form.cleaned_data['title']

            # create slug
            slug = create_slug(username,title)

            post = form.save(commit=False)
            post.user = User.objects.get(username=username)
            post.author = post.user.first_name+" "+post.user.last_name
            post.slug = slug
            post.save()
            # add branches
            for branch in branches:
                post.branch.add(branch)
            # add tags
            for tag in tags:
                tag = tag.strip().lower()
                domain, created = Tag.objects.get_or_create(name=tag)
                post.tags.add(tag)
            messages.success(request, "Post successfully created!")
            return redirect("/post/feed")
        else:
            messages.error(request, "Error in form submission. Please correct the errors below.")

    form = PostForm()
    # print("Hello")
    context={
        'form':form
    }
    return render(request,'post/make_post.html',context)
```

Community Engagement - Medium Priority (CR006)

```
# create an api call to add comments
@login_required(login_url='/account/login')
def add_comment(request,slug=None):
    username=request.user.__str__()
    if request.method == 'POST':
        comment = request.POST.get('comment')
        post = Post.objects.get(slug=slug)
        c = PostComment()
        c.user=request.user
        c.comment=comment
        c.post=post
        c.save()
        messages.success(request, "Comment successfully added!")
        return redirect("/post/"+slug)




@login_required(login_url='/account/login')
def tag_post(request,slug):
    username=request.user.__str__()
    try:
        # get tag
        tag = Tag.objects.get(slug=slug)
        # get posts
        post = Post.objects.filter(tags__name__in=[tag])
        # sort wrt to most recent
        post = post.order_by('-created_at')

        # Annotate each post with its comment count using a subquery
        posts = post.annotate(
            comment_count=Subquery(
                PostComment.objects.filter(post_id=OuterRef('post_id'))
                .values('post_id')
                .annotate(count=Count('comment_id'))
                .values('count')[:1]
            )
        )

        context={'post':posts,"name":"Tag: "+tag.name}
        return render(request, 'post/feed.html',context)
    except:
        messages.error(request, "No posts found for this tag.")
```

```
@login_required(login_url='/account/login')
def feed(request):
    username=request.user.__str__()
    # print(request.user.student.branch)
    user_branch = request.user.student.branch
    # Filter posts by the user's branch
    post = Post.objects.filter(branch=user_branch)

    # sort wrt to most recent
    post = post.order_by('-created_at')

    # Annotate each post with its comment count using a subquery
    posts = post.annotate(
        comment_count=Subquery(
            PostComment.objects.filter(post_id=OuterRef('post_id'))
            .values('post_id')
            .annotate(count=Count('comment_id'))
            .values('count')[:1]
        )
    )
    context={'post':posts,"name":"My Feed"}
    return render(request, 'post/feed.html',context)
```

Vital Announcements - High Priority (CR007)

```
class AnnouncementQuerySet(models.QuerySet):
    def search(self, query=None):
        qs = self
        if query is not None:
            or_lookup = (
                Q(title__icontains=query) |
                Q(content__icontains=query)
            )
            qs = qs.filter(or_lookup).distinct().only(
                'announcement_id', 'title', 'content', 'date_created'
            )  # Add more fields as needed
        return qs

class Announcement(models.Model):
    announcement_id= models.BigAutoField(primary_key=True,auto_created=True)
    title = models.CharField(max_length=100)
    featured_img = models.ImageField(upload_to='announcement/',null=True,blank=True)
    content = models.TextField()
    date_created = models.DateTimeField(auto_now_add=True)
    branch = models.ManyToManyField(Branch,related_name='branches')
    post_by = models.ForeignKey(User,on_delete=models.CASCADE)
    objects = AnnouncementQuerySet.as_manager()

    def __str__(self):
        return self.title
    def get_model_name(self):
        return 'Announcement'
```

```
@login_required(login_url='/account/login')
def announcement(request):
    user_branch = request.user.student.branch
    # retrive all
    announcements = Announcement.objects.filter(branch= user_branch)
    announcements = announcements.order_by('-date_created')


    context={'announcements':announcements}
    return render(request, 'announcement/announcement.html',context )

@login_required(login_url='/account/login')
@club_head_or_social_media_manager_required
def create(request):
    if request.method == 'POST':
        form = AnnouncementForm(request.POST, request.FILES)
        if form.is_valid():
            # save
            announcement = form.save(commit=False)
            announcement.post_by = request.user
            announcement.save()
            form.save_m2m() # for saving many to many
            return redirect('/announcement')
        else:
            return render(request, 'announcement/create_announcement.html', {'form': form})
    else:
        form = AnnouncementForm()
        return render(request, 'announcement/create_announcement.html', {'form': form})
```

Admin Management - High Priority (CR008)

```
class BranchAdmin(admin.ModelAdmin):
    list_display=(
        'department',
        'branch_name',
    )
    search_fields=[
        'department',
        'branch_name',

    ]
    list_editable= ['branch_name']
    list_filter=['department']

class DepartmentAdmin(admin.ModelAdmin):
    list_display=[
        'department_name'
    ]

    search_fields=[
        'department_name',

    ]

admin.site.register(Branch,BranchAdmin)

admin.site.register(Department,DepartmentAdmin)
```

```
class AnnoncementAdmin(admin.ModelAdmin):
    list_display=[
        'title',
        'content',
        'post_by',
        'list_branch'
    ]

    search_fields=[
        'title',
        'content',
        'post_by',
        'list_branch'
    ]

    list_filter=[
        'branch'
    ]

    def list_branch(self, obj):
        # Retrieve the list of domain tags for the mentor and return them as a string
        branch = obj.branch.all()  # Assuming 'domain' is the name of the TaggableManager field
        return ', '.join(b.branch_name for b in branch)

    list_branch.short_description = "branches"  # Set a custom column header for the domains


admin.site.register(Announcement,AnnoncementAdmin)
```

```
class ContactUsAdmin(admin.ModelAdmin):
    list_display = ['name', 'email', 'query', 'response','closed']
    search_fields = ['name', 'email', 'query']
    list_filter = ['sent_on']
    list_editable=['response','closed']
    readonly_fields = ('name', 'email', 'query', 'sent_on')


# Register the ContactUs model with the custom admin class
admin.site.register(ContactUs, ContactUsAdmin)
```

```
# Register your models here.
from .models import Post
from .models import PostComment

admin.site.register(Post)
admin.site.register(PostComment)
```

```
# Register your models here.
from django.contrib import admin
from .models import Resource

admin.site.register(Resource)
```

```
class StudentAdmin(admin.ModelAdmin):
    list_display=(
        'student_id',
        'user',
        'department',
        'branch',
        'college_email',
        'is_mentor',
        'year_of_passing_out',
    )
    search_fields=[
        'department',
        'branch',
        'student_id',
        'user',
        'college_email',
        'whatsapp_number',

    ]
    list_editable= [
        "is_mentor",
    ]
    list_filter=[
        'department',
        'is_mentor',
        'email_confirmed',
        'show_number',
        'year_of_passing_out',

    ]
    # readonly_fields = (
    #     'student_id',
    #     'whatsapp_number',
    #     'whatsapp_link',
    #     'user',
    #     'department',
    #     'branch',
    #     'college_email',
    #     'show_number',
    #     'email_confirmed',
    #     'year_of_passing_out',
    # )


class MentorAdmin(admin.ModelAdmin):
    list_display=(
        'student',
        'approved',
        'resume',
        'list_domains',
    )
    search_fields=[
        'student',
        'approved',
        'domain'
    ]
    list_editable= [
        'approved'
    ]
    list_filter=[
        'domain',
        'approved',

    ]
```

```
    def list_domains(self, obj):
        # Retrieve the list of domain tags for the mentor and return them as a string
        domain_tags = obj.domain.all()  # Assuming 'domain' is the name of the TaggableManager field
        return ', '.join(tag.name for tag in domain_tags)

    list_domains.short_description = "Domains"  # Set a custom column header for the domains

class ClubAdmin(admin.ModelAdmin):
    list_display= (
        'club_name',
        'club_desc',
        # 'list_branch',
    )

    search_fields=[
        'branch',
        'club_name',
        'club_desc',
    ]

    list_filter=[
        'branch',
    ]

    list_editable=[
        'club_desc',
    ]

    def list_branch(self, obj):
        # Retrieve the list of domain tags for the mentor and return them as a string
        branch = obj.branch.all()  # Assuming 'domain' is the name of the TaggableManager field
        return ', '.join(b.branch_name for b in branch)

    list_branch.short_description = "branches"  # Set a custom column header for the domains

class ClubMemberAdmin(admin.ModelAdmin):
    list_display= (
        'user',
        'club',
        'club_head',
        'social_media_manager',
        'joined_on',
    )

    search_fields=[
        'user',
        'club',
```

Mentor Page - Medium Priority (CR009)

```
@login_required(login_url='/account/login')
def show_mentor(request):
    # get all mentors of branch
    username= request.user.__str__()
    # get Student
    user=User.objects.get(username=username)
    student=Student.objects.get(user=user)
    # Branch
    branch=Branch.objects.get(branch_code=student.branch.branch_code)
    mentors=Mentor.objects.filter(student__branch__branch_code=branch.branch_code,approved=True)
    #print(mentors[0].student.user.username)
    context={'mentors':mentors}
    return render(request, 'account/show_mentor.html',context)

def show_mentors_by_domain(request,domain):
    # get all mentors of branch
    username= request.user.__str__()
    # get Student
    user=User.objects.get(username=username)
    student=Student.objects.get(user=user)
    # Branch
    mentors=Mentor.objects.filter(approved=True,domain__slug=domain)
    #print(mentors[0].student.user.username)
    context={'mentors':mentors}
    return render(request, 'account/show_mentor.html',context)
```

Mentor Approval System - High Priority (CR010)

```
class Mentor(models.Model):
    student = models.OneToOneField(Student, on_delete=models.CASCADE,related_name='mentor')
    resume= models.FileField(upload_to='resume/')
    domain= TaggableManager()
    description= models.CharField(max_length=5000,null=True, blank= True)
    approved= models.BooleanField(default=False)
    last_application_date = models.DateTimeField(auto_now_add=True,null=True, blank=True)

    # write list domains function which returns a string
    def list_domains(self):
        return ", ".join([p.name for p in self.domain.all()])
    def set_ismentor(self):
        if self.approved:
            # find student and update database
            s=Student.objects.get(pk=self.student.pk)
            s.is_mentor=True
            s.save()
        else:
            s=Student.objects.get(pk=self.student.pk)
            s.is_mentor=False
            s.save()
    def send_approval_mail(self):
        # seven_day_ago=timezone.now() - timezone.timedelta(days=7)
        if self.approved:
            u=User.objects.get(username=self.student.user.username)
            print("hello")
            email_from = settings.EMAIL_HOST_USER
            subject = 'Mentor Application Approved'
            message = 'Congratulations '+ u.first_name+" "+ u.last_name +'!! Your mentor application has been approved. You can now mentor students at our platform Campus Compass.'
            recipient_list = [self.student.college_email,u.email ]
            # send_mail( subject, message, email_from, recipient_list ,fail_silently=False)
            # u.email_user(subject,message)
            # elif self.last_application_date> seven_day_ago:
            try:
                send_mail(subject, message, email_from, recipient_list, fail_silently=False)
            except Exception as e:
                print(f"Email sending failed: {e}")
        else:
            u=User.objects.get(username=self.student.user.username)
            email_from = settings.EMAIL_HOST_USER
            subject = 'Removed as Mentor'
            message = 'Sorry '+ u.first_name+" "+ u.last_name +', You can no longer mentor students at our platform Campus Compass.'
            recipient_list = [self.student.college_email,u.email ]
            # send_mail( subject, message, email_from, recipient_list )
            # u.email_user(subject,message)
            try:
                send_mail(subject, message, email_from, recipient_list, fail_silently=False)
            except Exception as e:
                print(f"Email sending failed: {e}")


@receiver(pre_save, sender=Mentor) # a trigger that's run everytime
def Mentor_pre_save(sender, instance, **kwargs):
    if instance.pk is not None:
        instance.set_ismentor()
        instance.send_approval_mail()
```

Implementation Highlights

Customized Decorator for Club Heads Verification:

Implemented a custom decorator to verify if users have club head privileges.

Ensures secure access control by restricting certain functionalities to authorized club heads.

Enhances the project's security and user role management.

Email Verification Using SMTP:

Utilized the Simple Mail Transfer Protocol (SMTP) for user email verification.

Implemented a robust email verification system to enhance user account security.

Ensures that user-provided email addresses are valid and verified before account activation.

Integration of Rich Text Editor:

Introduced a rich text editor for enhanced content creation capabilities.

Users can now create and edit content with a user-friendly and feature-rich text editing experience.

Improved the overall user interaction with the platform, especially in content-heavy sections.

Adherence to Coding Standards

1. Maintained adherence to industry-standard coding practices in HTML, CSS, JS, and Python.
2. Followed consistent naming conventions, indentation, and code organization for improved readability and maintainability.
3. Prioritized clean and modular code design, contributing to the project's overall quality.

6. Test Plans

1)Test Items
1.1 Modules to be Tested
- a)Models
- b)Views
- c)Urls
- d)Forms
- e)Integration of Components

1.2 Modules Not to be Tested
- a)Apps
- b)Admin

Testing Objectives
2.1 Validate Model Functionality
- a)Test model str methods.
- b)Verify model-specific methods (e.g., get_model_name).
- c)Test model fields and their validations.

2.2 Test View Functionality
- a)Test views for correct HTTP responses.
- b)Validate view interactions with models.
- c)Test view rendering and template contexts.

2.3 Verify Form Validations
- a)Test form validations for correctness.
- b)Verify form submissions for various scenarios.
- c)Test form rendering and template contexts.

2.4 Test Integration of Components
- a)Verify the integration between models, views, and forms.
- b)Test the flow of data between different components.
- c)Validate the behavior of the entire system.

3.Testing Environment

3.1 Software Requirements

a) Django
b) Web browser
c) Database management system(MySQL)

3.2 Network Requirements

a) Web Server

4.Test Approach

4.1 Testing Levels

a)Unit Testing
b)Integration Testing
c)System Testing

4.2 Testing Types

a)Functional Testing
b)Integration Testing
c)User Interface Testing

4.3 Entry Criteria

a)Code review completed.
b)Environment setup for testing.

4.4 Exit Criteria

a)All critical and high-priority defects resolved.
b)Test cases executed and passed.

5.Test Schedule

5.1 Test Planning

10 November: Drafting and reviewing the test plan.
12 November: Finalizing the test plan.

5.2 Test Design

13 November: Creating test cases for each module.

5.3 Test Execution

15 November: Executing test cases.

5.4 Defect Reporting

17 November November: Identifying and documenting defects.

5.5 Regression Testing

18 November: Retesting after defect resolution.

5.6 Final Testing and Release

19 November: Final validation before project release.

6. Test Cases

a) test_contact_us_view
b) test_register_view
c) test_login_view
d) test_profile_view
e) test_account_authentication
f) test_mentor_registration_view
g) test_approve_membership_view
h) test_club_member_model

i) test_approve_membership_view
j) test_make_post_view_POST_create_post
k) test_tag_post_view
l) test_add_comment
m) test_add_comment
n) test_delete_post
o) test_edit_post
p) test_tagging_system
q) test_branch_selection_interface
r) test_search_view
s) test_view_returns_branches_with_correct_department_name

7. Test Cases and Test Results Matrix – including Screenshots of Inputs and Resulting Outputs of Execution of Test Cases

US001:

As a user, I want to see a visually appealing Home Page layout, so it's engaging and informative and I want to access the about us and contact us widgets as well.

Code:

```
class ContactUsViewTest(TestCase):
    def test_contact_us_view(self):
        # Define the valid form data
        form_data = {
            'name': 'John Doe',
            'email': 'johndoe@example.com',
            'query': 'Test query content.',
        }

        # Send a POST request to the contactus view with the valid form data
        response = self.client.post(reverse('contactus'), form_data)

        # Check if the form submission was successful and redirected to the expected URL
        self.assertEqual(response.status_code, 302)  # 302 indicates a redirect
        self.assertRedirects(response, reverse('contactsuccess'))

        # Check if a ContactUs instance was created with the provided data
        self.assertEqual(ContactUs.objects.count(), 1)
        contact = ContactUs.objects.first()
        self.assertEqual(contact.name, form_data['name'])
        self.assertEqual(contact.email, form_data['email'])
        self.assertEqual(contact.query, form_data['query'])
        Print ("Test contact us passed")
```

Output

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test home
Found 4 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..Test contact us passed
..
----------------------------------------------------------------------
Ran 4 tests in 0.066s

OK
Destroying test database for alias 'default'...
```

US002:

As an admin user, I want the Contact Us form to be managed by the admin page through the Django admin backend, and I want administrators to respond through emails to users when a new Contact Us message is submitted.

Code:

```
class ContactUsModelTest(TestCase):
    def test_send_response_mail_with_response(self):
        # Create a ContactUs instance with a response and an email address
        contact = ContactUs.objects.create(
            name='John Doe',
            email='shreyamishra414@gmail.com',
            query='Test query content.',
            response='Test response content.',
        )

        # response email should be sent because of pre_save

        # Check that an email was sent
        self.assertEqual(len(mail.outbox), 1)

        # Check the email details
        sent_email = mail.outbox[0]
        self.assertEqual(sent_email.subject, 'Response to your query')
        self.assertEqual(sent_email.from_email, settings.EMAIL_HOST_USER)
        self.assertEqual(sent_email.to, ['shreyamishra414@gmail.com'])
        self.assertIn('Test response content.', sent_email.body)

    def test_send_response_mail_without_response(self):
        # Create a ContactUs instance without a response
        contact = ContactUs.objects.create(
            name='Jane Smith',
            email='shreyamishra414@gmail.com',
            query='Another test query.',
        )

        # Send the response email (should not send an email)
        contact.send_response_mail()

        # Check that no email was sent
        self.assertEqual(len(mail.outbox), 0)
```

Output:

```
(.venv) PS D:\CampusCompass\CampusCompass> python manage.py test home.tests.ContactUsModelTest
Found 2 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..
----------------------------------------------------------------------
Ran 2 tests in 0.018s

OK
Destroying test database for alias 'default'...
(.venv) PS D:\CampusCompass\CampusCompass>
```

US003-As a user, I want to register and log in using user-friendly forms.

Code:

```
    def test_register_view(self):
        # Test the register view
        response = self.client.get('/account/register')
        self.assertEqual(response.status_code, 200)
        print("Test register_view passed")

    def test_login_view(self):
        # Test the login view
        response = self.client.get('/account/login')
        self.assertEqual(response.status_code, 200)
        print("Testlogin_view passed")
```

Output:

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test account
Found 10 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
Test account_activation passed
.Test approve_membership_view passed
.Test edit_profile_view passed
.Testlogin_view passed
.<MultiValueDict: {}>
Test mentor_registration_view passed
.Test profile_view passed
.Test register_view passed
.Test club_model passed
..Test mentor_model passed
.
----------------------------------------------------------------------
Ran 10 tests in 2.815s

OK
Destroying test database for alias 'default'...
```

US004-As a user, I want to view and edit my user profile information.

Code:

```
def test_profile_view(self):
    # Test the profile view
    response = self.client.get('/account/profile')
    self.assertEqual(response.status_code, 302)  # Redirects to login because user is not authenticated

    # Log in the user
    self.client.login(username='testuser', password='testpassword')

    response = self.client.get('/account/profile')
    self.assertEqual(response.status_code, 200)
    print("Test profile_view passed")

def test_edit_profile_view(self):
    # Test the edit profile view
    response = self.client.get('/account/edit_profile')
    self.assertEqual(response.status_code, 302)  # Redirects to login because user is not authenticated

    # Log in the user
    self.client.login(username='testuser', password='testpassword')

    response = self.client.get('/account/edit_profile')
    self.assertEqual(response.status_code, 200)
    print("Test edit_profile_view passed")
```

Output:

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test account
Found 10 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
Test account_activation passed
.Test approve_membership_view passed
.Test edit_profile_view passed
.Testlogin_view passed
.<MultiValueDict: {}>
Test mentor_registration_view passed
.Test profile_view passed
.Test register_view passed
.Test club_model passed
..Test mentor_model passed
.
----------------------------------------------------------------------
Ran 10 tests in 2.815s

OK
Destroying test database for alias 'default'...
```

US005 - As a user, I want to complete user authentication to authenticate their identity through email confirmations.

Code:

```
def test_account_authentication(self):
    # Test the account activation process
    token = account_activation_token.make_token(self.user)
    uidb64 = urlsafe_base64_encode(force_bytes(self.user.pk))

    # Ensure the user has a related student object before accessing student attributes
    if not hasattr(self.user, 'student') or not self.user.student:
        self.user.student = Student.objects.create(user=self.user)
    self.user.student.email_confirmed = True  # Set email_confirmed to True for testing
    self.user.save()

    response = self.client.get(reverse('activate', args=[uidb64, token]))
    self.assertEqual(response.status_code, 302)  # Redirects after successful activation
    print("Test account_authentication passed")
```

Output:

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test account
Found 10 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
Test account_activation passed
.Test approve_membership_view passed
.Test edit_profile_view passed
.Testlogin_view passed
.<MultiValueDict: {}>
Test mentor_registration_view passed
.Test profile_view passed
.Test register_view passed
.Test club_model passed
..Test mentor_model passed
.
----------------------------------------------------------------------
Ran 10 tests in 2.815s

OK
Destroying test database for alias 'default'...
```

US006- As a user, I want to access a "Forgot Password" and a "Change Password" page for password reset.

Code

```
def test_forgot_password_view(self):
    # Test the forgot password view
    response = self.client.get('/account/forgot_password')
    self.assertEqual(response.status_code, 200)
    print("Test forgot_password_view passed")

def test_reset_password_view(self):
    # Test the reset password view

    # Step 1: Request a password reset
    response = self.client.post('/account/forgot_password', {'email': 'test@example.com'})
    self.assertEqual(response.status_code, 302)  # Redirects to a success or confirmation page
    print("Test reset_password_view (Step 1) passed")

    # Step 2: Get the user and token from the password reset email (simulated here)
    user = User.objects.get(email='test@example.com')
    token = default_token_generator.make_token(user)
    uidb64 = urlsafe_base64_encode(force_bytes(user.pk))

    # Step 3: Access the password reset page
    reset_url = reverse('reset_password', args=[uidb64, token])
    response = self.client.get(reset_url)
    self.assertEqual(response.status_code, 200)
    print("Test reset_password_view (Step 3) passed")

    # Step 4: Submit a new password
    new_password = 'newtestpassword'
    response = self.client.post(reset_url, {'new_password1': new_password, 'new_password2': new_password})
```

Output

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test account
Found 10 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
Test forgot_password passed
Test rest_password passed
.Test approve_membership_view passed
.Test edit_profile_view passed
.Testlogin_view passed
.<MultiValueDict: {}>
Test mentor_registration_view passed
.Test profile_view passed
```

US007- As an aspiring mentor, I want to register for mentorship opportunities by submitting my resume, specifying my domain of expertise, and providing a brief description about myself.

Code:

```
def test_mentor_registration_view(self):
    # Test the mentor registration view
    response = self.client.get('/account/mentor_registration')
    self.assertEqual(response.status_code, 302)  # Redirects to login because user is not authenticated

    # Log in the user
    self.client.login(username='testuser', password='testpassword')

    response = self.client.get('/account/mentor_registration')
    self.assertEqual(response.status_code, 200)

    # Test submitting mentor registration form
    response = self.client.post('/account/mentor_registration', {'fileupload': 'test_resume.pdf', 'domains': 'Domain1
    self.assertEqual(response.status_code, 302)  # Redirects to profile after submission
    print("Test mentor_registration_view passed")
```

Output

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test account
Found 10 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
Test account_activation passed
.Test approve_membership_view passed
.Test edit_profile_view passed
.Testlogin_view passed
.<MultiValueDict: {}>
Test mentor_registration_view passed
.Test profile_view passed
.Test register_view passed
.Test club_model passed
..Test mentor_model passed
.
----------------------------------------------------------------------
Ran 10 tests in 2.815s

OK
Destroying test database for alias 'default'...
```

US008- As an administrator, I want to have control over mentor approval, ensuring that approved mentors receive confirmation emails.

Code:

```
def test_approve_membership_view(self):
    # Test the approve membership view
    response = self.client.get('/account/approve_membership')
    self.assertEqual(response.status_code, 302)  # Redirects to login because user is not authenticated

    # Log in the user
    self.client.login(username='testuser', password='testpassword')

    response = self.client.get('/account/approve_membership')
    self.assertEqual(response.status_code, 302)

    # Test submitting approve membership form
    response = self.client.post('/account/approve_membership', {'srn': 'SRN123', 'social_media_manager': True})
    self.assertEqual(response.status_code, 302)  # Redirects to profile after submission
    print("Test approve_membership_view passed")
```

Output:

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test account
Found 10 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
Test account_activation passed
.Test approve_membership_view passed
.Test edit_profile_view passed
.Testlogin_view passed
.<MultiValueDict: {}>
Test mentor_registration_view passed
.Test profile_view passed
.Test register_view passed
.Test club_model passed
..Test mentor_model passed
.
----------------------------------------------------------------------
Ran 10 tests in 2.815s

OK
Destroying test database for alias 'default'...
```

US009- As an administrator, I want the ability to create and manage clubs, including designating club heads, to facilitate the organization of club activities within the system.

Code

```
def test_club_member_model(self):
    club = Club.objects.create(
        club_name='Test Club',
        club_desc='Test Club Description',
        club_logo="/path/to/logo.jpg"
    )

    club_member = ClubMember.objects.create(
        user=self.user,
        club=club,
        club_head=True,
        social_media_manager=False
    )

    # Adjust the expected result based on your __str__ implementation
    self.assertEqual(club_member.__str__(), 'testuser (Test Club)')
    print("Test club_model passed")

def test_club_model(self):
    club = Club.objects.create(
        club_name='Test Club',
        club_desc='Test Club Description',
        club_logo="/path/to/logo.jpg"
    )

    club.branch.add(self.branch)

    self.assertEqual(club.__str__(), 'Test Club')
```

Output:

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test account
Found 10 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
Test account_activation passed
.Test approve_membership_view passed
.Test edit_profile_view passed
.Testlogin_view passed
.<MultiValueDict: {}>
Test mentor_registration_view passed
.Test profile_view passed
.Test register_view passed
.Test club_model passed
..Test mentor_model passed
.
----------------------------------------------------------------------
Ran 10 tests in 2.815s

OK
Destroying test database for alias 'default'...
```

US010- As a club head, I want the ability to add social media managers and club members to help manage our club's online presence.

Code

```
    def test_club_member_model(self):
        club = Club.objects.create(
            club_name='Test Club',
            club_desc='Test Club Description',
            club_logo="/path/to/logo.jpg"
        )

        club_member = ClubMember.objects.create(
            user=self.user,
            club=club,
            club_head=True,
            social_media_manager=False
        )

        # Adjust the expected result based on your __str__ implementation
        self.assertEqual(club_member.__str__(), 'testuser (Test Club)')

    def test_club_model(self):
        club = Club.objects.create(
            club_name='Test Club',
            club_desc='Test Club Description',
            club_logo="/path/to/logo.jpg"
        )

        club.branch.add(self.branch)

        self.assertEqual(club.__str__(), 'Test Club')
```

Output

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test account
Found 10 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
Test account_activation passed
.Test approve_membership_view passed
.Test edit_profile_view passed
.Testlogin_view passed
.<MultiValueDict: {}>
Test mentor_registration_view passed
.Test profile_view passed
.Test register_view passed
.Test club_model passed
..Test mentor_model passed
.
----------------------------------------------------------------------
Ran 10 tests in 2.815s

OK
Destroying test database for alias 'default'...
```

US011: As an administrator, I want the ability to add and manage branches and departments within the college or organization. These branches should be associated with specific departments for streamlined organization and management.

Code:

```
class DepartmentModelTest(TestCase):
# Existing test cases...

def test_create_department(self):
    new_department = Department.objects.create(department_name='New Department')
    self.assertIsNotNone(new_department.department_id)
    self.assertEqual(new_department.department_name, 'New Department')

def test_update_department_name(self):
    self.department.department_name = 'Updated Department'
    self.department.save()
    updated_department = Department.objects.get(department_id=self.department.department_id)
    self.assertEqual(updated_department.department_name, 'Updated Department')

def test_delete_department(self):
    department_id = self.department.department_id
    self.department.delete()
    deleted_department = Department.objects.filter(department_id=department_id).first()
    self.assertIsNone(deleted_department)
```

```
class BranchModelTest(TestCase):
    # Existing test cases...

    def test_create_branch(self):
        new_branch = Branch.objects.create(branch_name='New Branch', department=self.department)
        self.assertIsNotNone(new_branch.branch_code)
        self.assertEqual(new_branch.branch_name, 'New Branch')
        self.assertEqual(new_branch.department, self.department)

    def test_update_branch_name(self):
        self.branch.branch_name = 'Updated Branch'
        self.branch.save()
        updated_branch = Branch.objects.get(branch_code=self.branch.branch_code)
        self.assertEqual(updated_branch.branch_name, 'Updated Branch')

    def test_delete_branch(self):
        branch_code = self.branch.branch_code
        self.branch.delete()
        deleted_branch = Branch.objects.filter(branch_code=branch_code).first()
        self.assertIsNone(deleted_branch)
```

Output:

```
Found 19 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
............Test view_returns_branches_with_correct_department_name passed
Test view_returns_branches_with_correct_department_name passed
Test view_returns_branches_with_correct_department_name passed
Test view_returns_branches_with_correct_department_name passed
Test view_returns_branches_with_correct_department_name passed
....Test create_department passed
Test update_department passed
Test delete_department passed
.Test create_branch passed
Test update_branch passed
Test delete_branch passed
```

US012- As a club head and social media manager of a club, I want to create and manage announcements for my club's events and activities.

Code:

```
def test_club_head_can_create_announcement(self):
    # Assuming you have a ClubMember model linking users to clubs
    club_member = ClubMember.objects.create(user=self.user, club=self.club, club_head=True)

    # Log in the club head
    self.client.force_login(self.user)

    # Create a new announcement
    response = self.client.post(reverse('create'), {
        'title': 'Club Event Announcement',
        'content': 'Exciting event coming up!',
        'featured_img': SimpleUploadedFile("test_image.jpg", b"file_content", content_type="image/jpeg"),
    })

    self.assertEqual(response.status_code, 302)  # Redirect after successful form submission
    self.assertTrue(Announcement.objects.filter(title='Club Event Announcement').exists())
    print("Test club_head_can_create_announcement passed")

def test_social_media_manager_can_create_announcement(self):
    # Assuming you have a ClubMember model linking users to clubs
    club_member = ClubMember.objects.create(user=self.user, club=self.club, social_media_manager=True)

    # Log in the social media manager
    self.client.force_login(self.user)

    # Create a new announcement
    response = self.client.post(reverse('create'), {
        'title': 'Social Media Post',
        'content': 'Follow us on social media!',
```

```
def test_non_club_head_cannot_create_announcement(self):
    # Log in a user who is not a club head or social media manager
    self.client.force_login(self.user)

    # Attempt to create an announcement
    response = self.client.post(reverse('create'), {
        'title': 'Invalid Announcement',
        'content': 'This should not be created',
        'featured_img': SimpleUploadedFile("test_image.jpg", b"file_content", content_type="image/jpeg"),
    })

    self.assertEqual(response.status_code, 403)  # Forbidden
    print("Test non_club_head_cannot_create_announcement passed")

def test_announcement_validation(self):
    # Ensure that an announcement without a title or content cannot be created
    with self.assertRaises(ValidationError):
        Announcement.objects.create(
            title='',
            content='This announcement should fail validation.'
        )

    with self.assertRaises(ValidationError):
        Announcement.objects.create(
            title='Title is present, but content is empty',
            content=''
        )
        print("Test announcement_validation passed")
dd more tests as needed based on your model's functionlity
```

Output:

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test announcement
Found 6 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
<ul class="errorlist"><li>branch<ul class="errorlist"><li>This field is required.</li></ul></li></ul>
Test non_club_head_cannot_create_announcement passed
.Test announcement_validation passed
.Test social_media_manager_can_create_announcement passed
.Test club_head_can_create_announcement passed
...
----------------------------------------------------------------------
Ran 6 tests in 1.260s
```

US013- As an admin, I want to have the ability to create and manage announcements from the

admin panel for various aspects of college life, including college clubs, hackathons, events

Code:

```
def test_announcement_creation_view(self):
    # Test F001: Announcement Creation
    self.client.force_login(self.user)

    response = self.client.get(reverse('create'))
    self.assertEqual(response.status_code, 302)

    # Assuming your HTML form contains these fields
    form_data = {
        'title': 'Test Announcement',
        'content': 'This is a test announcement.',
        'featured_img': '',  # Replace with the actual file data if needed
    }

    form = AnnouncementForm(data=form_data)

    if not form.is_valid():
        print(form.errors)
    self.assertFalse(form.is_valid())

    response = self.client.post(reverse('create'), form_data)
    self.assertEqual(response.status_code, 302)  # Redirect after successful form submission
    self.assertTrue(Announcement.objects.filter(title='Test Announcement').exists())
    print("Test announcement creatione view passed")
```

Output:

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test announcement
Found 6 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
<ul class="errorlist"><li>branch<ul class="errorlist"><li>This field is required.</li></ul></li></ul>
Test announcement_creatione_view passed
.Test announcement_validation passed
```

US014: As a user, I want to view important announcements and updates on college events, including club events.

Code:

```
def test_feed_view(self):
    self.client.login(username='testuser', password='12345')
    response = self.client.get(reverse('feed'))
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, 'Test Post')
    print("Test feed_view passed")
```

Output:

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test post
Test delete_post passed
Found 19 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.Test edit_post passed
.........Test feed_view passed
.Test tag_post_view passed
```

US015- As a user, I want to access mentors within my branch for guidance. These mentors should also be recognized by their domains of expertise, with their resumes and basic information available and an option to chat.

Code:

```
def test_view_returns_branches_with_correct_department_name(self):
    response = self.client.get(reverse('get_branches', args=[self.department.department_id]))
    branches = json.loads(response.content)['branches']
    for branch in branches:
        # get department from branch_code
        branch_code = branch['branch_code']
        branch = Branch.objects.get(branch_code=branch_code )
        expected_department = branch.department
        self.assertEqual(expected_department.department_name, self.department.department_name)
        print("Test view_returns_branches_with_correct_department_name passed")
```

Output:

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test branch
Found 19 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
............Test view_returns_branches_with_correct_department_name passed
Test view_returns_branches_with_correct_department_name passed
Test view_returns_branches_with_correct_department_name passed
Test view_returns_branches_with_correct_department_name passed
Test view_returns_branches_with_correct_department_name passed
.......
----------------------------------------------------------------------
Ran 19 tests in 0.086s
```

US016- As a user, I want the ability to create posts with a rich text editor, select the branches for which the post is intended, and set tags to categorize the post.

Code

```
def test_tag_post_view(self):
    self.client.login(username='testuser', password='12345')
    response = self.client.get(reverse('tag_post', args=[self.tag.slug]))
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, 'Test Post')
    print("Test tag_post_view passed")
```

```
def test_make_post_view_POST_creates_post(self):
    self.browser.get(self.live_server_url + '/account/login')  # replace with the actual login url
    username_input = self.browser.find_element(By.NAME,"username")
    password_input = self.browser.find_element(By.NAME,"password")
    username_input.send_keys('testuser')
    password_input.send_keys('12345')
    password_input.send_keys(Keys.RETURN)

    WebDriverWait(self.browser, 2).until(
        EC.url_to_be(self.live_server_url + '/account/profile')
    )

    self.browser.get(self.live_server_url + '/post/make_post')  # replace with the actual make post url
    title_input = self.browser.find_element(By.ID,"id_title")
    content_input = self.browser.find_element(By.ID,"id_content")
    tags_input = self.browser.find_element(By.ID,"id_tags")
    branch_input = self.browser.find_element(By.ID,"id_branch")
    title_input.send_keys('Test Post')
    content_input.send_keys('This is a test post.')
    tags_input.send_keys('test, post')

    branch_code = str(int(self.branch.branch_code)-1)
    # branch_checkbox_id = f"id_branch_{branch_code}"
    branch_checkbox_id = f"id_branch_0"
    branch_checkbox = self.browser.find_element(By.ID, branch_checkbox_id)
```

Output

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test post
Found 19 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..........Test tag_post_view passed
```

US017-As a user, I want a dedicated post detail page that provides an in-depth view of a specific post, including all associated comments. Additionally, I would like the option to interact with the post by comments.

Code

```
def test_add_comment(self):
    # Log in
    self.browser.get(self.live_server_url + '/account/login')
    username_input = self.browser.find_element(By.NAME,"username")
    password_input = self.browser.find_element(By.NAME,"password")
    username_input.send_keys('testuser')
    password_input.send_keys('12345')
    password_input.send_keys(Keys.RETURN)

    WebDriverWait(self.browser, 2).until(
        EC.url_to_be(self.live_server_url + '/account/profile')
    )

    # Navigate to the post detail page
    self.browser.get(self.live_server_url + '/post/' + self.post.slug)

    # Fill in the comment form
    comment_input = self.browser.find_element(By.NAME, 'comment')
    comment_input.send_keys('Test Comment')

    # Submit the form
    comment_input.submit()

    # Wait for the page to reload
    WebDriverWait(self.browser, 2).until(
        EC.url_to_be(self.live_server_url + '/post/' + self.post.slug)
    )

    # Check that the comment was added
    body = self.browser.find_element(By.TAG_NAME, 'body')
    self.assertIn('Test Comment', body.text)
```

Output:

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test post
Found 19 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..........Test tag_post_view passed
```

US018- As a user, I want a dedicated post detail page that provides an in-depth view of a specific post, including all associated comments. Additionally, I would like the option to interact with the post by comments.

Code:

```
def test_add_comment(self):
    # Log in
    self.browser.get(self.live_server_url + '/account/Login')
    username_input = self.browser.find_element(By.NAME,"username")
    password_input = self.browser.find_element(By.NAME,"password")
    username_input.send_keys('testuser')
    password_input.send_keys('12345')
    password_input.send_keys(Keys.RETURN)

    WebDriverWait(self.browser, 2).until(
        EC.url_to_be(self.live_server_url + '/account/profile')
    )

    # Navigate to the post detail page
    self.browser.get(self.live_server_url + '/post/' + self.post.slug)

    # Fill in the comment form
    comment_input = self.browser.find_element(By.NAME, 'comment')
    comment_input.send_keys('Test Comment')

    # Submit the form
    comment_input.submit()

    # Wait for the page to reload
    WebDriverWait(self.browser, 2).until(
        EC.url_to_be(self.live_server_url + '/post/' + self.post.slug)
    )
```

Output:

```
.Test edit_post passed
.........Test feed_view passed
.Test tag_post_view passed
.
DevTools listening on ws://127.0.0.1:50349/devtools/browser/9e489088-9694-464c-9769-9ee098a87823
.
DevTools listening on ws://127.0.0.1:50403/devtools/browser/5595cb2b-94e5-4b24-88d2-0d7627eea6b5
[24/Nov/2023 12:25:25,697] - Broken pipe from ('127.0.0.1', 50447)
[24/Nov/2023 12:25:25,697] - Broken pipe from ('127.0.0.1', 50448)
.
DevTools listening on ws://127.0.0.1:50461/devtools/browser/ef421ba0-e5fa-4299-aa59-3776897cc1d8
.
DevTools listening on ws://127.0.0.1:50506/devtools/browser/ddecf834-5ad6-4431-9c4c-8fb8428ca07e
Test add_comment passed
```

US019-As a user, I want the ability to edit and delete my posts, as well as view all the posts I have authored on my profile for easy content management and reference.

Code:

```
def test_delete_post(self):
    # Log in
    self.client.login(username='testuser', password='12345')

    # Delete the post
    response = self.client.post(reverse('delete_post', kwargs={'slug': 'test-post'}))

    # Check that the response has a status code of 302 (redirect)
    self.assertEqual(response.status_code, 302)

    # Check that the post was deleted
    with self.assertRaises(Post.DoesNotExist):
        Post.objects.get(slug='test-post')

    # check if post exists
    self.assertFalse(Post.objects.filter(slug='test-post').exists())

def tearDown(self):
    # Log out
    self.client.logout()
print("Test delete_post passed")
```

```
def test_edit_post(self):
    # Log in
    self.client.login(username='testuser', password='12345')

    # Edit the post
    response = self.client.post(reverse('edit_post', kwargs={'slug': self.post.slug}), {
        'title': 'Updated Post',
        'content': 'Updated Content',
        'branch': [self.new_branch.branch_code],
        'tags': 'updatedtag',
    })

    # Check that the response has a status code of 302 (redirect)
    self.assertEqual(response.status_code, 302)

    # Check that the post was updated
    post = Post.objects.get(post_id=self.post.post_id)
    self.assertEqual(post.title, 'Updated Post')
    self.assertEqual(post.content, 'Updated Content')
    self.assertIn(self.branch, post.branch.all())
    self.assertIn('updatedtag', [tag.name for tag in post.tags.all()])

def tearDown(self):
```

Output

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test post
Found 19 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..........Test tag_post_view passed
```

US020- As a user, I want to upload educational resources in PDF format, tagging them with target audiences and branches. Users should be able to view and download these resources.

Code:

```
class ResourceModelTestCase(TestCase):
    def apload_resource(self):
        # Create a test department
        self.department = Department.objects.create(department_name='Test Department')

        # Create test objects for dependencies
        self.branch = Branch.objects.create(branch_name='Test Branch', department=self.department)
        self.user = User.objects.create_user(username='testuser', password='testpassword')

        # Create a test resource
        # Use SimpleUploadedFile to simulate file upload in tests
        file_content = b'Test file content'
        uploaded_file = SimpleUploadedFile("testfile.txt", file_content, content_type="text/plain")
        self.resource = Resource.objects.create(
            user=self.user,
            title='Test Resource',
            files=uploaded_file,
        )
        self.resource.branch.add(self.branch)

        # Add tags to the resource
        self.tag1 = Tag.objects.create(name='tag1')
        self.tag2 = Tag.objects.create(name='tag2')
        self.resource.tags.add(self.tag1, self.tag2)
        print("Test upload_resource passed")
```

Output:

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test resource
Found 4 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
Test upload_resource passed
Branch selection interface test success!
.Test upload_resource passed
Test edit_resource success
.Test upload_resource passed
File storage test success!
.Test upload_resource passed
Tagging system test success!
```

US021- As a user, I want to filter and view resources based on my branch and tags meant for a certain audience.

Code:

```
def test_tagging_system(self):
    # Test if tags are associated with the resource correctly
    self.assertTrue(self.tag1 in self.resource.tags.all())
    self.assertTrue(self.tag2 in self.resource.tags.all())
    print("Tagging system test success!")

def test_branch_selection_interface(self):
    # Test if the branch is associated with the resource correctly
    self.assertEqual(self.resource.branch.count(), 1)
    self.assertEqual(self.resource.branch.first(), self.branch)
    print("Branch selection interface test success!")
```

Output:

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> pyth
Found 4 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
Branch selection interface test success!
.Metadata test success!
.File storage test success!
.Tagging system test success!
.
----------------------------------------------------------------------
Ran 4 tests in 0.750s
```

US022: As a user, I want the ability to edit resources that I have previously uploaded, allowing me to update and maintain the content for improved accuracy and relevance.

Code:

```
def test_edit_resource(self):
    # Simulate editing the resource
    edited_file_content = b'Updated file content'
    edited_uploaded_file = SimpleUploadedFile("updatedfile.txt", edited_file_content, content_type="text/plain")
    self.resource.title = 'Updated Resource'
    self.resource.files = edited_uploaded_file

    # Save the edited resource
    self.resource.save()

    # Retrieve the updated resource from the database
    updated_resource = Resource.objects.get(id=self.resource.id)

    # Assertions to check if the changes are reflected correctly
    self.assertEqual(updated_resource.title, 'Updated Resource')
    self.assertEqual(updated_resource.files.read(), edited_file_content)
    self.assertEqual(updated_resource.user, self.user)

    # Ensure that the tags and branch associations remain unchanged
    self.assertTrue(self.tag1 in updated_resource.tags.all())
    self.assertTrue(self.tag2 in updated_resource.tags.all())
    self.assertEqual(updated_resource.branch.count(), 1)
    self.assertEqual(updated_resource.branch.first(), self.branch)

    print("Edit resource test success!")
```

Output:

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test resource
Found 4 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
Branch selection interface test success!
.Test edit_resource success
.File storage test success!
.Tagging system test success!
.
----------------------------------------------------------------------
Ran 4 tests in 1.231s
```

US023 - As a user, I want to access a comprehensive profile page that allows me to view all my authored posts, uploaded resources, and the clubs I am a part of.

Code:

```
def test_comprehensive_profile_view(self):
    # Test the comprehensive profile view
    response = self.client.get(reverse('comprehensive_profile'))
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, 'Test Bio')  # Check if the bio i
    self.assertContains(response, 'Test Club')  # Check if the join
    self.assertContains(response, 'Test Announcement')  # Check if
    self.assertContains(response, 'Test Resource')  # Check if the
    print("Test comprehensive_profile_view passed")
```

Output:

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test account
Found 10 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
Test forgot_password passed
Test comprehensive_profile_view passed
.Test approve_membership_view passed
.Test edit_profile_view passed
.Testlogin_view passed
.<MultiValueDict: {}>
Test mentor_registration_view passed
.Test profile_view passed
.Test register_view passed
.Test club_model passed
```

US024- As a user, I want the ability to view other users' profile pages, where I can access their basic information, initiate a chat with them, and browse the posts and resources they have shared.

Code:

```
def test_view_other_user_profile(self):
    # Test the ability to view other users' profile pages
    response = self.client.get(reverse('view_profile', args=[self.user2.username]))
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, 'User 2 Bio')  # Check if the bio is displayed
    self.assertContains(response, '2222222222')  # Check if the WhatsApp number is disp
    self.assertContains(response, 'User 2 Announcement')  # Check if the authored annou
    self.assertContains(response, 'User 2 Resource')  # Check if the uploaded resource
    print("Test view_other_user_profile passed")
```

Output:

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage.py test account
Found 10 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
Test view_other_user_profile passed
.Test approve_membership_view passed
.Test edit_profile_view passed
.Testlogin_view passed
.<MultiValueDict: {}>
Test mentor_registration_view passed
```

US025-As a user, I want access to a search functionality that allows me to efficiently search for

posts, resources, announcements, and users within the platform.

Code:

```
def test_search_view(self):
    # Assume you have a URL named 'search' in your urls.py
    url = reverse('search')

    # Test with a valid query
    response = self.client.post(url, {'q': 'Test'})
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'search/search.html')
    print("test search_view passed")
    # Add more assertions based on your specific use case and expected behavior
```

Output

```
(.venv) PS C:\Se_project\Campus_Compass\Campus-Compass\CampusCompass> python manage
Found 3 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
[<Post: Test Post>, <Resource: Test Resource>, <Announcement: Test Announcement>, <
test search_view passed
.[]
..
----------------------------------------------------------------------
Ran 3 tests in 0.740s

OK
```

8. Final Gantt Chart (Baseline and Final Timelines)

9. Conclusions

In conclusion, the development and implementation of Campus Compass mark a significant milestone in addressing the dynamic needs of college students. The project, grounded in the Agile Software Development Model with the Scrum framework, embodies adaptability, collaboration, and iterative progress. The use of robust tools, such as Jira for project management and Figma for design, underscores the commitment to efficiency and transparency throughout the development lifecycle.

The architecture and design choices, as depicted in detailed UML diagrams, highlight the careful consideration given to system structure and behavior. The implementation, featuring a Django-based backend and Bootstrap for frontend design, reflects a harmonious blend of functionality and aesthetics. Notable features, including the mentorship network, doubt resolution system, and resource sharing module, signify the platform's commitment to fostering a supportive and engaged college community.

The project adheres to coding standards, employs quality assurance measures, and ensures adherence to industry best practices. The introduction of a rich text editor, keyboard shortcuts, and other user-centric elements enhances the platform's usability and aligns with educational best practices.

Throughout the project's lifecycle, from initiation to ongoing support and maintenance, meticulous planning and execution have been prioritized. The Gantt chart provides a visual representation of the well-structured timeline, emphasizing task dependencies and milestones.

As Campus Compass moves beyond the development phase, the focus shifts to ongoing support, maintenance, and continuous improvement. Post-launch, monitoring system performance, addressing user feedback, and implementing updates based on evolving needs will be essential for sustaining the platform's effectiveness.

In essence, Campus Compass stands as a testament to the collaborative effort of the development team, the insights of stakeholders, and the dedication to creating a valuable resource for college students navigating the complexities of academic life. As the project concludes, the anticipation is high for the positive impact it will have on facilitating knowledge sharing, mentorship, and community engagement within the college environment.

In the comprehensive development of Campus Compass, a myriad of functional features have

been seamlessly integrated to create a holistic platform catering to the diverse needs of college students. The user-centric design incorporates robust authentication for secure access, fostering a community-driven environment. The mentorship module establishes a vital connection between experienced students and those seeking guidance, promoting knowledge sharing and community engagement. A sophisticated resource-sharing system facilitates the exchange of educational materials, contributing to a collaborative learning ecosystem. The doubt resolution mechanism ensures timely academic support, enhancing the overall student experience. With an intuitive user interface and interactive components, the platform offers a rich text editor, keyboard shortcuts, and adherence to GUI standards, prioritizing usability and accessibility. The announcement and event management system keeps users informed about vital updates, fostering a sense of community and connection. As Campus Compass evolves beyond development, its amalgamation of features stands poised to empower college students in navigating their academic journey with efficiency and engagement.

10. Appendix A: Glossary of Abbreviations and Acronyms

Agile Software Development Model: An iterative and incremental approach to software development that prioritizes flexibility and collaboration.

Admin: The college administrator for a specific college, utilizing the platform to communicate with students and share important information.

API (Application Programming Interface): A set of rules that enables one software application to interact with another.

Architecture & Design Choices: Decisions made regarding the system's overall structure and design.

Bootstrap 5.1.3: A front-end framework utilized for design and styling elements in the Campus Compass system, providing standard design components and conventions.

Button Placement: Ensuring that action buttons are appropriately positioned and labeled for clarity.

Community Engagement Guidelines: Rules and principles for fostering positive and respectful interactions within a community.

Concurrency: The ability of the system to handle multiple operations simultaneously.

Content Hierarchy: Prioritizing the arrangement of content based on its importance and relevance to the user.

Corporate Policies: Guidelines set by the corporate entity, in this case, the educational institution.

CSS (Cascading Style Sheets): Style sheet language used for describing the look and formatting of a document written in HTML.

CSRF Attacks (Cross-Site Request Forgery attacks): A security threat to be protected against.

Data Backup: The process of creating copies of data to prevent data loss.

Data Exchange and Flow: Processing user requests through Django, communication with the MySQL database, and presentation of data through Bootstrap.

Data Flow Architecture: A depiction of how data moves through the system, illustrating processes, data stores, and data flow paths.

Data Security: Measures to protect data from unauthorized access or alterations.

Data Transfer Protocols: Mechanisms for transferring data between different software components.

Development Code Files: The actual source code files containing the implementation of the system's functionalities.

Doubt Resolution: The process of seeking academic answers from peers and mentors.

Educational Best Practices: Accepted principles and methods for effective teaching and learning.

Encryption: The process of converting information into a code to prevent unauthorized access.

Entity Relationship Diagram (ERD): A visual representation of the relationships between

entities in a database, showcasing how data is organized and connected.

Error Categories: Classification of error messages into categories such as validation errors, system errors, user authentication errors, and data entry errors.

Error Message Display Standards: Consistent format, prominent positioning, plain language, specificity, color coding, and categorization of error messages based on type.

Flexibility: The ability to adapt to changing requirements by organizing work into time-boxed sprints.

Frontend: The user interface part of the system, organized using the Django app structure.

GDPR (General Data Protection Regulation): European Union regulations for data protection and privacy.

GUI Standards: Design standards applied to containers, content sections, forms, data presentation, dashboard elements, dialog boxes, modal windows, and other user-facing modules.

HTML (HyperText Markup Language): Standard markup language for creating web pages.

HTTP (Hypertext Transfer Protocol): Standard protocol for communication with web servers in the Campus Compass system.

Intellectual Property Rights: Legal rights that protect creations of the mind, such as inventions, designs, and artistic works.

Internationalization: Designing and developing software to support multiple languages and locales.

Jira: A comprehensive task tracking, agile planning, collaboration, and reporting platform.

JS (JavaScript): Programming language that enables interactive web pages.

Keyboard Shortcuts: Shortcut commands to enhance user productivity, including text styling, text alignment, undo/redo, and other operations within the rich text editor.

Levels of DFD: Diagrams illustrating different levels of abstraction in a Data Flow Diagram.

Levels of Effort Estimation: The estimation of effort required for different project tasks and phases.

Linux: A family of open-source Unix-like operating systems.

Maintainability: The ease with which the system can be updated and maintained.

macOS: Operating system developed by Apple Inc. for its Macintosh line of computers.

Mentorship Network: Connection between experienced students (mentors) and those seeking guidance.

Model-View-Template (MVT) Architecture: Django's architecture where Models represent data, Views handle presentation logic, and Templates manage the presentation layer.

MVC (Model-View-Controller): A software design pattern for developing web applications.

MySQL: The relational database management system used for data storage and retrieval.

Operating System (OS): Software that manages computer hardware and software resources.

ORM (Object-Relational Mapping): Django's system for managing data and communication with the MySQL database.

PDF (Portable Document Format): A common file format for educational resources.

Performance Requirement: Criteria specifying the desired speed and efficiency of system processes.

Portability: The ability of software to run on different platforms.

Project Structure: The organization and layout of the project's files and directories, including frontend, backend, testing, and configuration files.

QA (Quality Assurance): Ensuring the quality and compliance of the final product.

Readability: The presentation of text and visual elements in a manner that is easy to read, considering factors such as typography, spacing, and contrast.

Readme: A file containing information about the project, typically in the root directory of the repository.

Reliability: The consistency and dependability of system operations.

Reporting and Analytics: Features that allow the monitoring and analysis of system usage and user behavior.

Requirement Traceability Matrix: A document that correlates system requirements with the elements of the system design.

Repository Link: The link to the version control repository (GitHub) where the project's source code is stored.

Resource Sharing Module: A component of Campus Compass facilitating the exchange of educational resources among students.

RDBMS (Relational Database Management System): A type of database management system that stores data in tables with relationships.

Rich Text Editor: A text editor with formatting capabilities, integrated into the project for enhanced content creation.

Role-Based Access Control: A system where access permissions are determined by user roles.

Selenium: Testing tool for automated testing of web applications.

Sequence Diagrams: Diagrams that depict the interactions between different system components or actors over time.

SMTP (Simple Mail Transfer Protocol): A protocol used in sending and receiving emails.

Software Interfaces: Interactions with MySQL database, web browsers (Chrome, Edge), external APIs, libraries/frameworks (Django, JavaScript, Bootstrap), and external components.

SMTP (Simple Mail Transfer Protocol): Used for email communication, particularly for account registration and password reset in the Campus Compass system.

SRS (Software Requirements Specification): A document outlining the software requirements for the "Campus Compass" project.

State Diagram: A diagram illustrating the different states a system or component can be in and the transitions

11. Appendix B: RTM (Final version)

Sl. no.	Requirement ID	Brief Description of Requirement	Architecture Reference	Design Reference	Code File Reference	Test Case ID	Status
1	FEATURE-1	User Registration and Authentication - High Priority	AR-001, AR-002	DR-001,002, 003,004,005, 006	CR-001	US-003, US-005, US-006	Completed
2	FEATURE-2	User Profile Management - High Priority	AR-001, AR-002	DR-001,002, 003,004,005, 006	CR-002	US-004, US-023, US-024	Completed
3	FEATURE-3	Resource Sharing - Medium Priority	AR-001, AR-002	DR-001,002, 006	CR-003	US-020, US-021, US-022	Completed
4	FEATURE-4	Mentorship and Guidance - High Priority	AR-001, AR-002	DR-001,002, 003,004,005, 006	CR-004	US-007	Completed
5	FEATURE-5	Doubt Resolution - Medium	AR-001, AR-002	DR-001,002, 006	CR-005	US-016, US-018, US-019	Completed

		Priority					
6	FEATURE-6	Community Engagement - Medium Priority	AR-001, AR-002	DR-001,002, 006	CR-006	US-017	Completed
7	FEATURE-7	Vital Announcements - High Priority	AR-001, AR-002	DR-001,002, 003, 004,006	CR-007	US-010, US-012, US-013	Completed
8	FEATURE-8	Admin Management - High Priority	AR-001, AR-002	DR-001,002, 003,004,005, 006	CR-008	US-009, US-011, US-013	Completed
9	FEATURE-9	Mentor Page - Medium Priority	AR-001, AR-002	DR-001,002, 005, 006	CR-009	US-015	Completed
10	FEATURE-10	Mentor Approval System - High Priority	AR-001, AR-002	DR-001,002, 003,004,005, 006	CR-010	US-008	Completed

12. Appendix C: Technology stack and References [Books, Links to web pages/portals, tools]

Links:

Scrum Guide
https://scrumguides.org
Git Documentation
https://git-scm.com/doc
Jira Documentation
https://confluence.atlassian.com/jira
Selenium Documentation
https://www.selenium.dev/documentation/
PyTest Documentation
https://docs.pytest.org/en/stable/
Python Documentation

https://docs.python.org/3/
Django Documentation
https://docs.djangoproject.com/en/4.2/
Bootstrap Documentation
https://getbootstrap.com/docs/4.1/getting-started/introduction/
Technical Documentation in Software Development
https://www.altexsoft.com/blog/technical-documentation-in-software-development-types-best-practices-and-tools/
Testing Documentation in Software Engineering
https://www.geeksforgeeks.org/testing-documentation-software-engineering/
Design Documentation in Software Engineering
https://www.geeksforgeeks.org/design-documentation-in-software-engineering/
Use Case Diagram for Library Management System
https://www.geeksforgeeks.org/use-case-diagram-for-library-management-system/
Data Flow Diagram
https://www.lucidchart.com/pages/data-flow-diagram
Data Flow Architecture
https://www.tutorialspoint.com/software_architecture_design/data_flow_architecture.htm
System Context Diagram
https://online.visual-paradigm.com/knowledge/system-context-diagram/what-is-system-context-diagram/
Introduction of ER Model
https://www.geeksforgeeks.org/introduction-of-er-model/
How to Write Software Requirements Specification (SRS) Document
https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document
Traceability Matrix
https://www.guru99.com/traceability-matrix.html
Gantt Chart in Agile Project Management
https://www.atlassian.com/agile/project-management/gantt-chart
Testing Django with Selenium
https://ordinarycoders.com/blog/article/testing-django-selenium
Django Selenium Documentation
https://django-selenium.readthedocs.io/en/latest/

Books:

"Code Complete" by Steve McConnell:
McConnell, S. (2004). Code Complete: A Practical Handbook of Software Construction. Microsoft Press.

"The Mythical Man-Month" by Frederick P. Brooks Jr.:
Brooks Jr., F. P. (1995). The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley.

"Clean Code: A Handbook of Agile Software Craftsmanship" by Robert C. Martin:
Martin, R. C. (2008). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.

"Design Patterns: Elements of Reusable Object-Oriented Software" by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, and Grady Booch:
Gamma, E., Helm, R., Johnson, R., Vlissides, J., & Booch, G. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

"The Pragmatic Programmer: Your Journey to Mastery" by Andrew Hunt and David Thomas:
Hunt, A., & Thomas, D. (1999). The Pragmatic Programmer: Your Journey to Mastery. Addison-Wesley.

Tools:
GitHub
https://github.com/
Git
https://git-scm.com/
Jira
https://www.atlassian.com/software/jira
Draw.io
https://app.diagrams.net/
Selenium
https://www.selenium.dev/