Software Requirements Specification

for

Website to Rate Professor

Version <1.0>

Prepared by

Group Name: Campus Critics

Geeta Akshita Anusuri	SE22UARI051	se22uari051@mahindrauniversity.edu.in
Kuruva Bhuvana Vijaya	SE22UARI080	se22uari080@mahindrauniversity.edu.in
Laxmi Sowmya Niharika Peddireddi	SE22UARI081	se22uari081@mahindrauniversity.edu.in
Madarapu Sri Harshitha	SE22UARI087	se22uari087@mahindrauniversity.edu.in

Instructor:	Arun Avinash Chauhan
Course:	Software Engineering
Lab Section:	Tuesday 8:30 am to 10:30 am
Teaching Assistant:	Murali Krishna Bukkasamudram
Date:	04/03/2025

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
Draft 1.0	Laxmi Sowmya Niharika Peddireddi, Bhuvana Vijaya Kuruva, Madarapu Sri Harshitha, Geeta Akshita Anusuri	Initial draft of the SRS.	01/03/25

1 Introduction

1.1 Document Purpose

This document specifies the software requirements for the "Website to Rate Professor" system. It includes the functionalities, constraints, and objectives of the system. The document serves as a reference for developers, testers, and stakeholders involved in the project.

1.2 Product Scope

This is a web-based platform that helps college students make informed decisions by providing verified reviews and ratings of professors and courses. Students can rate professors on teaching quality, clarity, and responsiveness while also sharing insights on course workload and study resources.

To ensure credibility, the platform requires student authentication via ID and OTP, minimizing fake reviews. With a user-friendly interface and real-time feedback, Campus Critics enhances transparency in course selection, helping students choose the right professors and academic path.

1.3 Intended Audience and Document Overview

This project is for students to share and access professor ratings, course insights, and helpful study resources. It helps new students make informed decisions based on peer experiences while ensuring a student-driven, interactive platform.

1.4 Definitions, Acronyms and Abbreviations

- ***API** – Application Programming Interface*
- ***CRUD** – Create, Read, Update, Delete*
- ***OTP** – One-Time Password*
- ***UI** – User Interface*
- ***UX** – User Experience*

1.5 Document Conventions

This Software Requirements Specification (SRS) document follows the IEEE formatting standards. The conventions used in this document are as follows:

- ***Font & Spacing:** Arial, size 11 or 12, single-spaced.*
- ***Margins:** 1-inch margins on all sides.*
- ***Headings & Subheadings:** Follow the template for section and subsection titles.*
- ***Italics:** Used for comments or explanatory notes.*
- ***Bold:** Used for emphasis on key terms or section headings.*
- ***Acronyms & Abbreviations:** Defined in the "Definitions, Acronyms, and Abbreviations" section.*
- ***Figures & Tables:** Numbered sequentially with caption*
- ***Code Snippets & System Commands:** Displayed in a monospace font to distinguish them from regular text.*

These conventions ensure clarity, consistency, and readability throughout the document.

1.6 References and Acknowledgments

This SRS document references the following sources:

- *IEEE Standard for Software Requirements Specifications (IEEE 830-1998)*
- *University academic policies and guidelines on faculty evaluations*
- *UI/UX best practices for review and rating platforms*
- *Authentication and security standards for web applications*

2 Overall Description

2.1 Product Overview

***Campus Critics** is a web-based platform designed to assist college students in making well-informed academic decisions by providing verified reviews and ratings of professors and courses. The system allows students to rate professors based on teaching quality, clarity, and responsiveness, while also sharing insights on course workload and study resources.*

This platform is developed to fill the gap in academic transparency by offering a credible and student-authenticated rating system. Unlike unstructured social media discussions or informal peer advice, Campus Critics ensures review authenticity by implementing student authentication via university ID and OTP verification, thereby minimizing fake reviews.

```
+------------------------+       +---------------------------+       +--------------------------+
|  Student User          | -----> |  React.js Frontend    | -----> |  Flask Backend          |
|  (Writes Reviews)      |        |  (User Interface)     |        |  (Data Processing &|
|  (Reads Reviews)       | <----  |  (Search & Filter)    | <----- |  Authentication)        |
+------------------------+       +---------------------------+       +--------------------------+
                                         |
                                         v
                              +--------------------------+
                              |  Database (SQL)          |
                              |  Stores Reviews &        |
                              |  User Credentials        |
                              +---------------------------+
```

2.2 Product Functionality

***Professor Ratings & Reviews** – Students can rate professors based on teaching quality, clarity, responsiveness, grading leniency, and course difficulty.*

***Course Insights** – Provides details on workload, difficulty level, and student recommendations for each course.*

***Verified Student Reviews** – Requires authentication using student ID and OTP to ensure credibility and minimize fake reviews.*

***Student-Recommended Resources** – Allows students to share helpful study materials, websites, and YouTube channels for courses.*

***User Authentication & Security** – Ensures secure login and data protection for users.*

Search & Filter Options *– Enables users to search for professors and courses based on various criteria.*

User-Friendly Interface *– Provides a structured and intuitive platform for submitting and viewing reviews.*

2.3 Design and Implementation Constraints

Software Design Methodology: *This system must be designed using the COMET method, it is a well structured approach for developing real-time and distributed systems.*

Modeling Language: *The UML must be used for system modeling and design documentation.*

Reference: *UML Specification by the Object Management Group (OMG).*

Hardware Limitations: *The system should run smoothly within the available memory and processing power, ensuring good performance.*

Technology Stack: *The backend will be developed using flask, while MySQL will be used to manage the database.*

Integration Requirements*: The software will be designed to seamlessly connect with external authentication services and third-party APIs whenever needed.*

Developers will write clean, well-organized code following industry best practices, making it easy to understand, maintain, and expand as the software evolves.

2.4 Assumptions and Dependencies

The development and operation of Campus Critics rely on certain assumptions and external dependencies that could impact the project's feasibility and performance. If these assumptions change or dependencies fail, adjustments may be required in the design or implementation.

Assumptions:

University ID System Availability *– It is assumed that universities provide a digital student ID system that can be integrated for authentication.*

User Engagement *– Students will actively participate in rating and reviewing professors, ensuring a diverse and valuable dataset.*

Authentic Reviews *– Users will provide honest feedback.*

Dependencies:

Third-Party Authentication Services *– OTP-based verification and student ID validation may rely on third-party APIs or university authentication systems.*

Database Management System *– A secure and scalable database is required to store user data, reviews, and course information.*

Front-End & Back-End Technologies *– The development will rely on web frameworks like React.js (for front-end) and Flask (for back-end).*

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The rating platform will have a ***simple and user-friendly web interface*** *where students can easily find professors, rate them, and share their feedback. This platform will have a search bar for quick lookups and straightforward menus for easy navigation purposes. It has features like star ratings and comment boxes. It focuses on making the experience smooth and hassle free for students to review and submit ratings effortlessly.*

Search Bar *– To find professors easily.*
Rating System *– Star ratings and comment boxes.*
Navigation Menu *– Categories like Home, Top-Rated Professors, and Submit Review.*
Filters & Sorting *– Options to filter by department, course, or rating.*

3.1.2 Hardware Interfaces

The platform is a web-based application accessible on desktops and laptops. The primary hardware interfaces include user devices, where students and professors interact with the system through a web browser, and a web server that hosts the Flask backend and manages data processing. A database server stores and retrieves user data, ratings, and shared resources. The system communicates with these hardware components through standard request-response interactions, ensuring seamless access to stored information.

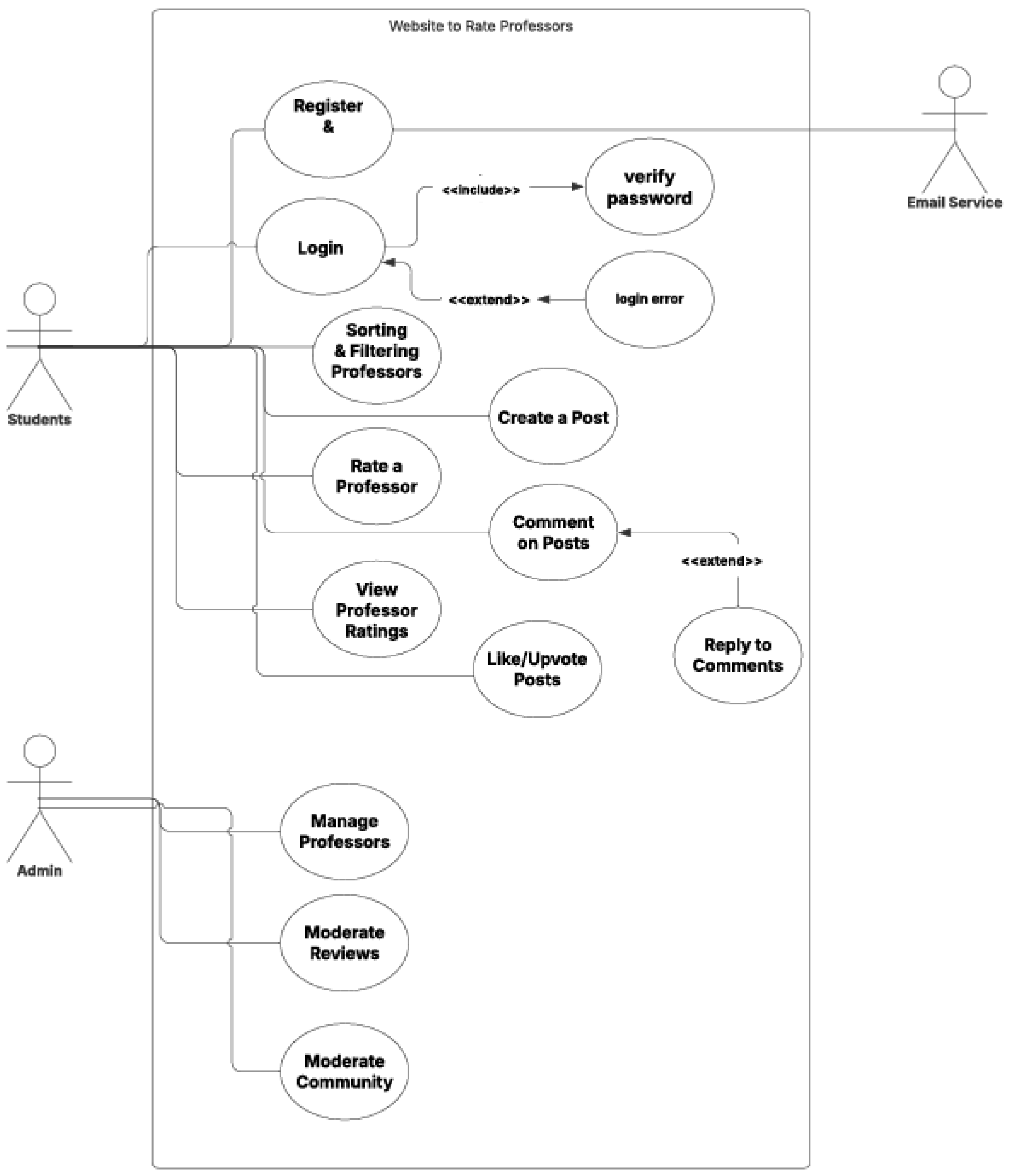
3.1.3 Software Interfaces

The platform's software components interact through well-defined interfaces. The frontend, developed using React.js, connects with the backend, which runs on Flask. The backend processes requests, manages authentication, and handles data storage and retrieval through a database. An email service is integrated to facilitate OTP-based authentication for verifying student identities.

3.2 Functional Requirements

User Registration & Login: *Students must be able to create accounts, log in securely, and manage their profiles.*
Search & Filter Professors: *Users should be able to search for professors by name, department, or university and filter results based on ratings and reviews.*
Rate & Review Professors: *Students can submit ratings using a star-based system and provide written feedback on their experiences.*
View & Sort Ratings: *Users can view professor ratings, with options to sort by highest-rated, most-reviewed, or most recent feedback.*
Flag Inappropriate Reviews: *Users should have the ability to report or flag inappropriate reviews for moderation.*
Admin Moderation: *Administrators must be able to review flagged content, approve or remove reviews, and manage reported users.*
Responsive Web Interface: *The platform should be accessible on various devices, including desktops, tablets, and mobile phones.*

3.3 Use Case Model



Use Case Specifications

3.3.1 Use Case U1: Login

Author: *Kuruva Bhuvana Vijaya*
Purpose: *To allow students to authenticate and access the system.*
Requirements Traceability: *Tied to the functional requirement that the system shall provide login functionality using student ID and password.*
Priority: *High*
Preconditions:

- *The student must have a valid student ID.*
- *The student must be registered in the system.*

Postconditions:

- *The student is logged into the system.*
- *A session is created.*

Actors: *Student*
Extends: *Login Error*

Flow of Events:

- ***Basic Flow:***
  1. *The student inputs their student ID and password.*
  2. *The system validates the ID and password.*
  3. *The student can login now.*
- ***Alternative Flow:*** *If the student ID is not valid, an error is displayed.*
- ***Includes:*** *Verify password*
- ***Notes/Issues:*** *Consider adding multi-factor authentication for increased security.*

3.3.2 Use Case U2: Register

Author: *Laxmi Sowmya Niharika Peddireddi*
Purpose: *To allow new students to register for the system.*
Requirements Traceability: *The system shall allow students to create an account using their student ID and email.*

Priority: *High*
Preconditions:

- *The student must have a valid student ID and email.*
- *The student ID must not already be registered.*

Postconditions:

- *The student is successfully registered and can log in.*

Actors: *Student, Email Service*

Flow of Events:

- ***Basic Flow:***
    1. *The student provides their student ID and email.*
    2. *The system validates the details.*
    3. *The system sends an email verification.*
    4. *The student verifies the email.*
    5. *The system completes the registration.*
- ***Alternative Flow:*** *If the student ID is already registered, an error is shown.*

- ***Exceptions:*** *Email service failure prevents account verification.*

- ***Includes:*** *Verify Password*

- ***Notes/Issues:*** *Consider implementing CAPTCHA to prevent bot registrations.*

3.3.3 Use Case U3: Sort & Filter Professors

Author: *Kuruva Bhuvana Vijaya*
Purpose: *To allow students to search and filter professor ratings.*
Priority: *Medium*
Preconditions: *The student must be logged in.*
Postconditions: *The student sees a list of filtered professor ratings.*
Actors: *Student*

Flow of Events:

- ***Basic Flow:*** *The student selects sorting/filtering options and views results.*
- ***Alternative Flow:*** *If no professors match, an empty result is returned.*
- ***Exceptions:*** *If the database is down, an error message is displayed.*

3.3.4 Use Case U4: Rate a Professor

Author: *Madarapu Sri Harshitha*
Purpose: *To allow students to submit professor ratings.*
Priority: *High*
Preconditions: *The student must be logged in.*
Postconditions: *The rating is successfully submitted.*
Actors: *Student*

Flow of Events:

- ***Basic Flow:***
    1. *The student selects a professor.*
    2. *The student enters a rating and review.*
    3. *The system saves the rating.*
- ***Alternative Flow:*** *The student edits or deletes their rating.*
- ***Exceptions:*** *If the professor does not exist, an error message is shown.*

3.3.5 Use Case U5: View Professor Ratings

Author: *Laxmi Sowmya Niharika Peddireddi*
Purpose: *To allow students to view professor ratings.*
Priority: *High*
Preconditions: *The student must be logged in.*
Postconditions: *The student sees professor ratings.*
Actors: *Student*

Flow of Events:

- ***Basic Flow:*** *The student selects a professor and views ratings.*
- ***Alternative Flow:*** *If no ratings exist, an empty state is shown.*

3.3.6 Use Case U6: Create a Post

Author: *Madarapu Sri Harshitha*
Purpose: *To allow students to create community posts.*
Priority: *High*
Preconditions: *The student must be logged in.*
Postconditions: *The post is successfully created.*
Actors: *Student*

Flow of Events:

- ***Basic Flow:***
    1. *The student enters post content.*
    2. *The system saves the post.*

3.3.7 Use Case U7: Comment on Posts

Author: *Kuruva Bhuvana Vijaya*
Purpose: *To allow students to comment on posts.*
Priority: *High*
Preconditions: *The student must be logged in.*
Postconditions: *The comment is successfully posted.*
Actors: *Student*

Flow of Events:

- ***Basic Flow:***
    1. *The student selects a post.*
    2. *The student enters a comment.*
    3. *The system saves the comment.*

3.3.8 Use Case U8: Reply to Comments

Author: *Laxmi Sowmya Niharika Peddireddi*
Purpose: *To allow students to reply to comments.*
Priority: *Medium*
Preconditions: *The student must be logged in.*
Postconditions: *The reply is successfully posted.*
Actors: *Student*

3.3.9 Use Case U9: Like/Upvote Posts

Author: *Madarapu Sri Harshitha*
Purpose: *To allow students to like or upvote posts.*
Priority: *Medium*
Preconditions: *The student must be logged in.*
Postconditions: *The like/upvote is recorded.*
Actors: *Student*

3.3.10 Use Case U10: Manage Professors

Author: *Kuruva Bhuvana Vijaya*
Purpose: *To allow admins to manage the list of professors, including adding, updating, and removing professors.*
Priority: *High*
Preconditions: *The user must be an admin and logged in.*
Postconditions: *The professor list is updated.*
Actors: *Admin*

3.3.11 Use Case U11: Moderate Reviews

Author: *Laxmi Sowmya Niharika Peddireddi*
Purpose: *To allow admins to moderate and manage professor reviews.*
Priority: *High*
Preconditions: *The user must be an admin and logged in.*
Postconditions: *The review is removed or approved.*
Actors: *Admin*

3.3.12 Use Case U12: Moderate Community

Author: *Madarapu Sri Harshitha*
Purpose: *To allow admins to moderate community posts and comments.*
Priority: *High*
Preconditions: *The user must be an admin and logged in.*
Postconditions: *The post or comment is approved or removed.*
Actors: *Admin*

4 Other Non-functional Requirements

4.1 Performance Requirements

1. ***P1. Login & OTP Verification***
   - *The OTP should be sent within **10 seconds** after entering the student ID.*
   - *OTP verification should be completed within **5 seconds** after submission.*
2. ***P2. Website Load Time***
   - *The homepage should load in **under 3 seconds** on a standard internet connection.*
   - *The rating form should appear in **2 seconds or less** after clicking.*

3. ***P4. Search Performance***
   - *Searching for a professor or course should take **less than 2 seconds**.*
   - *Basic optimization techniques should be applied to speed up searches.*
4. ***P5. OTP Security Rules***
   - *OTPs should expire **within 10 minutes** after being sent.*
   - *A student can request **up to 5 OTPs per hour** to account for possible errors.*
5. ***P6. API Response Time***
   - *Requests between the frontend and backend should respond **in under 1 second** under normal conditions.*
   - *During high traffic, response time should not exceed **3 seconds**.*
6. ***P7. Rating & Review Submission***
   - *Ratings and reviews should be saved and visible **within 3 seconds** after submission.*

4.2 Safety and Security Requirements

- Only authorized admins can manage professor profiles and moderate reviews.
- Students and faculty must log in using a **secure authentication method**, such as **email and password**
- *All sensitive information, including passwords and submitted ratings, will be encrypted to prevent unauthorized access.*

Safety and Security Requirements

1. ***S1. Role-Based Access Control (RBAC)***
   - *Administrative actions such as managing professor profiles, moderating reviews, and handling user reports should be restricted to authorized personnel.*
   - *Students can only submit ratings and reviews but cannot modify or delete other users' content.*
2. ***S2. User Authentication & Verification***
   - *Students must authenticate using their **student ID and OTP verification** before accessing the rating system.*
   - *OTPs must be sent to the **registered student email** to prevent unauthorized access.*
3. ***S3. Data Privacy & Protection***
   - *Student information, including email and student ID, must be **encrypted** in storage and during transmission.*
   - *Personally identifiable information (PII) must not be exposed to other users.*
   - *The system must comply with **FERPA (Family Educational Rights and Privacy Act)** to protect student records.*

4. ***S4. Secure Communication***
   - *All data exchanged between the frontend (React) and backend (Flask) must be* ***encrypted using HTTPS (SSL/TLS)****.*
   - *OTPs and authentication data must not be stored in plaintext.*
5. ***S5. Protection Against Unauthorized Access***
   - ***Rate limiting*** *must be implemented to prevent brute-force login attempts.*
   - *If a user enters the wrong OTP* ***5 times****, their account must be temporarily locked for* ***10 minutes****.*
   - *Students should not be able to submit more than* ***3 ratings per professor per semester*** *to prevent spam.*
6. ***S6. Secure Storage & Database Access***
   - *All passwords and sensitive data must be hashed using* ***bcrypt or a similar hashing algorithm****.*
   - *The database should be protected from* ***SQL injection*** *and* ***NoSQL injection*** *attacks.*
7. ***S7. Moderation & Reporting System***
   - *Users should be able to report inappropriate reviews.*
   - *Admins must have tools to* ***flag, edit, or remove*** *content that violates guidelines.*
8. ***S8. Session Security***
   - *User sessions should automatically* ***expire after 30 minutes of inactivity****.*
   - *Session tokens must be securely stored and not exposed in URLs.*
9. ***S9. Backup & Recovery***
   - *The database must be backed up* ***daily*** *to prevent data loss.*
   - *A rollback mechanism should be in place in case of accidental deletion or corruption.*

4.3 Software Quality Attributes

4.3.1 Reliability

The system will minimize downtime through error handling, database backups, and logging to monitor performance.

4.3.2 Maintainability

A modular code structure with well-documented components will allow easy updates and bug fixes. Version control will track changes.

4.3.3 Usability

A simple, intuitive interface with search functionality and clear navigation will enhance user experience. Usability testing will be conducted.

4.3.4 Scalability

The database and backend will be optimized to handle growing users and data, with indexing for fast queries and efficient request handling.

Appendix A – Data Dictionary

Variable/Entity	***Type***	***Possible Values/States***	***Description***	***Operations/Requirements***
Username	*String*	*Alphanumeric (A-Z, a-z, 0-9, _, .)*	*Unique identifier for the user*	*Must be unique, case-insensitive, 4-20 characters long*
Password	*String*	*Any combination of characters*	*User authentication credential*	*At least 8 characters, includes letters & numbers*
Email	*String*	*Valid email format*	*User's registered email*	*Must be unique, used for login & notifications*
UserType	*Enum*	*{Student, Admin}*	*Defines role of the user*	*Determines access rights to functionalities*
ReviewID	*Integer*	*Auto-generated unique ID*	*Identifier for a review*	*Assigned when a review is created*
ProfessorID	*Integer*	*Auto-generated unique ID*	*Identifier for a professor*	*Used to fetch professor details*
Rating	*Integer*	*{1, 2, 3, 4, 5}*	*Numeric rating for the professor*	*Must be between 1 and 5*
LikesCount	*Integer*	*Non-negative integer*	*Number of likes/upvotes on a review*	*Updated when users like a review*
Comments	*String*	*Text*	*User comments on a review*	*Can be added by students*
ReportID	*Integer*	*Auto-generated*	*Identifier for*	*Created when a user*

		unique ID	*reported reviews*	*reports a review*
ReportReason	*String*	*Predefined reasons (Spam, etc.)*	*Reason for reporting a review*	*Must be selected from predefined list*
Register/Login	*Function*	*Inputs: Username, Password, Email*	*Allows users to create an account or log in*	*Must validate credentials before granting access*
FilterOption	*Enum*	*{Highest Rated, Most Recent, Most Liked}*	*Sorting/filtering criteria for reviews*	*Applied to displayed results*
ProfessorList	*List<Integer>*	*List of ProfessorIDs*	*Collection of professor profiles*	*Generated based on search/filter*
Submit Review & Rating	*Function*	*Inputs: ReviewContent, Rating, ProfessorID*	*Allows students to post reviews and ratings*	*Stores data in the database*
Edit/Delete Review	*Function*	*Inputs: ReviewID, ReviewContent*	*Allows users to modify or delete their reviews*	*Only the original author can edit/delete*
Store Reviews	*Database*	*Stores: ReviewContent, Rating, Timestamp*	*Maintains all submitted reviews*	*Securely stores reviews with timestamps*

Appendix B - Group Log

Meeting 1: Project Planning

Date: *03-02-2025*
Attendees: *Kuruva Bhuvana Vijaya, Laxmi Sowmya Niharika Peddireddi, Madarapu Sri Harshitha, Geeta Akshita Anusuri*
Discussion Points:

- *Defined project scope and objectives.*
- *Chose React.js for frontend and Flask for backend.*

- *Decided on authentication using student ID and OTP.*

  Action Items:
- *Research similar platforms for reference.*
- *Set up a shared repository for version control.*

Meeting 2: System Design

Date: *18-02-2025*
Attendees: *Kuruva Bhuvana Vijaya, Laxmi Sowmya Niharika Peddireddi, Madarapu Sri Harshitha, Geeta Akshita Anusuri*
Discussion Points:

- *Finalized database schema.*
- *Discussed UI/UX design.*
- *Finalized other design specifications*

  Action Items:
- *Begin frontend development.*
- *Create wireframes for the user interface.*

Meeting 3: Documentation and Design

Date: *01-03-2025*
Attendees: *Kuruva Bhuvana Vijaya, Laxmi Sowmya Niharika Peddireddi, Madarapu Sri Harshitha, Geeta Akshita Anusuri*
Discussion Points:

- *Created the Use Case Diagram.*
- *Worked on the Software Requirements Specification (SRS) document.*
- *Discussed system functionality and planned next steps.*

  Action Items:
- *Finalize and review the SRS document.*
- *Continue refining the system design.*
- *Plan for initial development phase.*