Software Requirements Specification

for

Website to Rate Professors

Version 1.0

Prepared by

Group Name: <place your group name here>

|  |  |  |
| --- | --- | --- |
| J.Srihitha | SE22UCSE117 | [se22ucse117@mahindrauniversity.edu.in](mailto:se22ucse117@mahindrauniversity.edu.in) |
| J.Hasini | SE22UCSE114 | [se22ucse114@mahindrauniversity.edu.in](mailto:se22ucse114@mahindrauniversity.edu.in) |
| J.Akanksha | SE22UCSE119 | [se22ucse119@mahindrauniversity.edu.in](mailto:se22ucse119@mahindrauniversity.edu.in) |
| M.Srihitha | SE22UCSE306 | [se22ucse306@mahindrauniversity.edu.in](mailto:se22ucse306@mahindrauniversity.edu.in) |
| M.Ashritha  Anjusree | SE22UCSE158  SE22UCSE283 | [**se22ucse158@mahindrauniversity.edu.in**](mailto:se22ucse158@mahindrauniversity.edu.in)  [se22ucse283@mahindrauniversity.edu.in](mailto:se22ucse283@mahindrauniversity.edu.in) |

|  |  |
| --- | --- |
| Instructor: | Vijay Rao |
| Course: | Software Engineering |
| Lab Section: | CSE |
| Teaching Assistant: | Sravanthi Acchugatla |
| Date: | 10-03-2025 |

Contents

Contents ii

Revisions ii

1 Introduction 1

1.1 Document Purpose 1

1.2 Product Scope 1

1.3 Intended Audience and Document Overview 1

1.4 Definitions, Acronyms and Abbreviations 2

1.5 Document Conventions 2

1.6 References and Acknowledgments 2

2 Overall Description 3

2.1 Product Overview 3

2.2 Product Functionality 3

2.3 Design and Implementation Constraints 3

2.4 Assumptions and Dependencies 4

3 Specific Requirements 5

3.1 External Interface Requirements 5

3.2 Functional Requirements 6

3.3 Use Case Model 7

4 Other Non-functional Requirements 13

4.1 Performance Requirements 13

4.2 Safety and Security Requirements 13

4.3 Software Quality Attributes 13

5 Other Requirements 14

Appendix A – Data Dictionary 15

Appendix B - Group Log 16

Revisions

| Version | Primary Author(s) | Description of Version | Date Completed |
| --- | --- | --- | --- |
| Draft Type and Number | Full Name | Information about the revision. This table does not need to be filled in whenever a document is touched, only when the version is being upgraded. | 00/00/00 |

# 

# Introduction

This document describes the software requirements for **Website to Rate Professors,** a **professor rating website** that allows students to rate and review faculty members based on multiple factors and lets the professors to respond to the reviews. The document provides an overview of the system, including functionalities, constraints, and assumptions.

## Document Purpose

This Software Requirements Specification document outlines the functional and non-functional requirements for the "Website to Rate Professors" system. The purpose of this document is to provide a detailed description of the system’s objectives, features, and constraints for stakeholders, including developers, testers, and clients.

This document covers the complete system, including both front-end and back-end components, as well as integration with databases and external authentication mechanisms. It is intended to guide the development process from design to deployment while ensuring that all requirements align with software engineering principles.

## Product Scope

The "Website to Rate Professors" system is a web-based platform that allows students to rate and review their professors based on various criteria such as teaching quality, course difficulty, and student engagement. The primary goal of the system is to provide an insightful, data-driven platform for students to make informed academic decisions while also enabling professors to receive constructive feedback and respond back.

The system offers the following benefits:

* Provides a user-friendly interface for students to submit reviews and browse professors.
* Implements authentication and authorization to ensure data integrity and prevent spam.
* Uses analytical tools to generate statistical insights on professor ratings.
* Enhances transparency and accountability in academia by fostering student-professor engagement.
* Ensures anonymity and moderation to prevent biased or inappropriate reviews.

## Intended Audience and Document Overview

This document is intended for the following stakeholders:

* **Developers & testers:** To understand functional and technical requirements.
* **Professors:** To review the website features and scope.
* **End Users (Students & Professors):** To understand the purpose and features of the system.

The document is organized as follows:

* **Section 1:** Introduction, including purpose, scope, and conventions.
* **Section 2:** System description, including overall architecture.
* **Section 3:** Functional requirements and system features.
* **Section 4:** Non-functional requirements such as security and performance.
* **Section 5:** References and supporting documentation.

It is recommended that readers begin with Section 1 for an overview before proceeding to the detailed requirements in Sections 2 and 3.

## Definitions, Acronyms and Abbreviations

API – Application Programming Interface

CSS – Cascading Style Sheets

DBMS – Database Management Systems

GUI – Graphical User Interface

HTTP – Hypertext Markup Language

JWT – JSON Web Token

SQL – Structured Query Language

UI – User Interface

UX – User Experience

## Document Conventions

This document follows the IEEE formatting requirements with the following conventions:

* Font: Arial, size 11 or 12, single-spaced.
* Section and subsection titles are bolded.
* Italics are used for comments or placeholders.
* Code snippets and technical terms appear in monospaced font.
* Margins are maintained at 1 inch for readability.

## References and Acknowledgments

* IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications.
* University documentation on academic policies and professor evaluation criteria.
* Online style guides for UI/UX best practices.
* Open-source projects and API documentation for authentication and security best practices.
* Research papers and articles on rating systems and sentiment analysis in online platforms.

# Overall Description

## Product Overview

The "Website to Rate Professors" is a self-contained product designed to provide students with a platform to rate and review their professors based on various criteria such as teaching effectiveness, course difficulty, and engagement. The system aims to help students make informed decisions when selecting courses while providing constructive feedback to educators. This product also lets professors receive reviews and respond back. This product is a standalone web application with potential integration into university systems for authentication and data validation.

The system consists of the following major components:

* **User Interface:** A web-based portal accessible to students and professors.
* **Database Management System (DBMS):** Stores professor ratings, reviews, and user information.
* **Authentication Module:** Ensures secure access for students and faculty.
* **Review Moderation System:** Filters and moderates content to prevent misuse.
* **Reporting and Analytics:** Provides summarized insights on professor performance.

## Product Functionality

The system provides the following major functionalities:

* **User Registration and Authentication:** Secure login and sign-up process.
* **Professor Search:** Users can search for professors by name, department, or university.
* **Rating Submission:** Students can submit reviews and rate professors based on multiple parameters.
* **Review Moderation:** Filters inappropriate content and prevents spam.
* **Professor response:** Professors can respond to the reviews.
* **Data Analytics and Reporting:** Generates insights on professor performance.
* **Professors and universities’ comparison:** Users can compare universities and professors.
* **User Profile Management:** Allows users to update account details.
* **Admin Panel:** Enables moderators to manage users, reviews, and flagged content.

## Design and Implementation Constraints

The development of this project is subject to several constraints, including:

* **Technology Stack:** The project will use **React.js** for the frontend, **Node.js with Express.js** for the backend, and **MySQL** for the database.
* **Software Development Methodology:** The **COMET** method will be followed for structured software design.
* **Modeling Language:** The project will use **Unified Modeling Language (UML)** to design system architecture.
* **Hardware Constraints:** Must run on cloud servers with scalable backend support.
* **Security Measures:** Must comply with data protection laws and prevent unauthorized access.

## Assumptions and Dependencies

The following assumptions and dependencies are critical to the successful execution of the project:

* **Users have valid university credentials** for authentication.
* **Internet access is required** for accessing the platform.
* **Third-party libraries** will be used.
* **Hosting on a cloud service (e.g., AWS, Azure, or Firebase)** is assumed for deployment.
* **The project depends on MySQL for data storage** and structured querying.

Any deviations from these assumptions may impact the project’s functionality, requiring modifications to the design and implementation.

# Specific Requirements

## External Interface Requirements

### User Interfaces

The system will have a web-based user interface that allows students and faculty to interact with the platform. The main user interface will consist of:

* **Homepage:** Displays a search bar to find universities and professors, login/signup options, and featured top-rated professors.
* **University Page:** Provides university details along with a list of professors, their ratings, and options to view or compare them.
* **Search and Filter Page:** Users can search for professors by name, department, or university and filter results based on ratings and reviews.
* **Professor Profile Page:** Displays professor details, department, university affiliation, overall ratings, individual reviews, and interactive buttons to rate or compare professors.
* **Review Submission Page:** Allows users to submit feedback, including textual reviews and ratings on predefined criteria such as teaching quality, clarity, and responsiveness.
* **Comparison Page:** Enables users to compare multiple professors’ side by side based on ratings, reviews, and university affiliation.
* **Admin Dashboard:** Accessible only to moderators for review approval, user management, professor additions, and report generation.

The user interface will feature intuitive navigation, a clean layout, and a responsive design to ensure accessibility on different devices, including desktops, tablets, and mobile phones.

### Hardware Interfaces

The system will be deployed on cloud infrastructure and will interact with the following hardware interfaces:

* **Web Servers:** Hosts the website and processes user requests.
* **Database Servers:** Stores user data, reviews, and ratings.
* **User Devices:** The system will be accessible via desktops, laptops, tablets, and smartphones through a web browser.
* **Authentication System:** May integrate with university login systems for authentication.

### Software Interfaces

The Website to Rate Professors interacts with various software components:

* **University Authentication System:** Verifies students via **OAuth SSO** (if integrated).
* **Database (MySQL):** Stores user accounts, professor details, ratings, and reviews.
* **Frontend-Backend Communication:** The **React.js frontend** interacts with the **Node.js/Express.js backend** via **RESTful APIs**.
* **Email Service:** Sends account verification and notifications via **AWS SES**.
* **Cloud Hosting:** Deployed on **AWS, Azure, or Firebase** for scalability.

These interfaces ensure smooth data flow, authentication, and content moderation.

## Functional Requirements

*This section details the specific functional requirements of the* ***Website to Rate Professors*** *system.*

### User Registration and Authentication

 The system shall allow users to register using their university email.

 The system shall require email verification before account activation.

 The system shall provide secure login and logout functionality.

### Professor Search

 The system shall allow users to search for professors by name or university.

 The system shall display search results with relevant professor details.

### Review & Rating Submission

 The system shall allow students to rate professors based on parameters such as teaching effectiveness, clarity, and engagement.

 The system shall allow users to write detailed reviews with optional comments.

### Professor Search

 The system shall allow users to **search for professors** by name, or university.

 The system shall display **search results** with relevant professor details.

### Professor Response

 The system shall allow professors to respond to student reviews.

 The system shall ensure responses are visible under the respective review.

### Professors and Universities Comparison

 The system shall allow users to compare professors based on ratings and feedback.

 The system shall allow users to compare universities based on professor reviews.

### User Profile Management

 The system shall allow users to update their account details, including password changes.

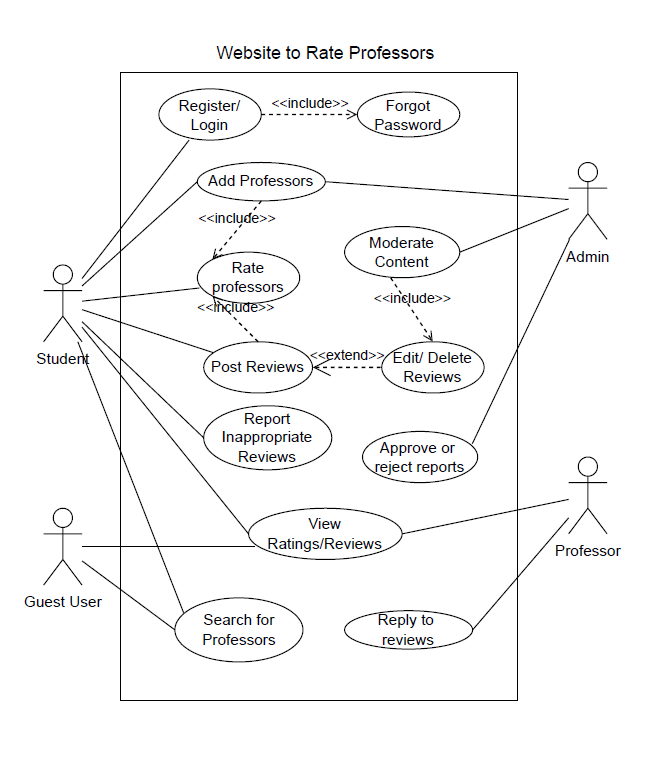
 The system shall allow users to view their submitted reviews and ratings.

### Admin Panel

 The system shall provide an admin dashboard to manage users, reviews, and flagged content.

 The system shall allow admins to ban users or remove inappropriate reviews.

## Use Case Model



### Use Case #1 (Register/Login - U1)

**Author –**

**Purpose** - Allow users (students, professors) to create an account and log into the system.

**Requirements Traceability –** This use case is related to the requirement that users must be able to register and log in to access system functionalities. The system should verify user credentials and provide authentication to ensure security.

**Priority** - High

**Preconditions** - The user must have access to the website, and the system must be online and available. If the user is logging in, they must already have an account.

**Post conditions** - After successful execution, the user is authenticated and redirected to their dashboard. If the login fails, an error message is displayed, and the user is prompted to re-enter their credentials or reset their password.

**Actors** – Students, Professors

**Extends –** -

**Flow of Events -**

* 1. Basic Flow - The user enters their login credentials (email and password). The system verifies the credentials against the database. If the credentials are correct, the user is redirected to their respective dashboard.
  2. Alternative Flow - If the user forgets their password, they can use the "Forgot Password" feature to reset it.
  3. Exceptions - If the credentials are incorrect, the system displays an error message and prompts the user to try again.

**Includes -** None

**Notes/Issues** - The system should ensure secure authentication, and additional security measures like multi-factor authentication or captcha should be considered.

### Use Case #2 (Update Profile Information – U2)

**Author -**

**Purpose -** This use case allows users to update their personal profile details, ensuring that their account information remains accurate and up to date.

**Requirements Traceability -** The system must provide an option for users to modify their profile information, including their name, email address, and profile picture. Changes should be validated and securely stored in the database.

**Priority -** Medium

**Preconditions -** The user must be logged into the system to access and update their profile information.

**Postconditions -** After successfully updating their profile, the user’s new details are saved in the system, and any visible profile information is updated accordingly.

**Actors -** Student, Professor

**Extends - -**

**Flow of Events -**

1. Basic Flow - The user navigates to the profile settings page. The system displays the user's current profile details. The user updates their information and submits the changes. The system validates the input and saves the updated details. The system displays a confirmation message that the profile has been successfully updated.
2. Alternative Flow - If the user changes their email, the system may require them to verify the new email before updating it.
3. Exceptions - If the database is unavailable, the system displays an error message and prevents profile updates. If invalid data is entered (e.g., an incorrectly formatted email), the system prompts the user to correct the information.

**Includes –** None

**Notes/Issues:** The system should implement validation checks to prevent incorrect or malicious data entry. A history of changes may also be useful for audit purposes.

### Use Case #3 (Rate Professors – U3)

**Author –**

**Purpose -** This use case allows students to rate professors based on their teaching quality.

**Requirements Traceability -** The system must allow students to provide ratings for professors, and these ratings should be stored and updated in the database. The average rating of a professor should be dynamically calculated and displayed to users.

**Priority –** High

**Preconditions -** The student must be logged into the system to rate a professor.

**Postconditions -** After the rating is submitted, the professor's overall rating is updated in the system, and the student receives confirmation that their rating has been recorded.

**Actors -** Student

**Extends -** Post Reviews (U4)

**Flow of Events -**

1. Basic Flow - The student selects a professor they want to rate. The student provides a rating (on a scale of 1 to 5) and optionally leaves a comment. The system updates the professor's overall rating based on the new input.
2. Alternative Flow - If the student has already rated the professor, they can edit or update their rating.
3. Exceptions -If the system is down, the student is notified that ratings cannot be submitted at the moment.

**Includes –** None

**Notes/Issues -** The system should have spam prevention measures to avoid multiple ratings from the same user in a short time.

### Use Case #4 (Post Reviews – U4)

**Author –**

**Purpose -** This use case allows students to post reviews about professors, sharing feedback on their teaching, behaviour, and overall performance.

**Requirements Traceability -** The system should allow students to submit reviews for professors, store them in a database, and make them visible to other users. There should also be a mechanism to edit or delete reviews if necessary.

**Priority -** High

**Preconditions -** The student must be logged into the system and should have selected a professor to review.

**Postconditions -** Once the review is successfully posted, it becomes visible to other users. If a review violates guidelines, it can be reported or moderated.

**Actors -** Student

**Extends -** Edit/Delete Reviews (U5)

**Flow of Events –**

1. Basic Flow - The student selects a professor to review. The student writes a review and submits it. The system saves the review and updates the professor’s profile. The review is displayed to other users.
2. Alternative Flow - If the student has already reviewed the professor, they may edit or delete their previous review.
3. Exceptions - If the system detects inappropriate language, the review submission is rejected with an error message. If the system is down, the student is notified that the review cannot be posted at the moment.

**Includes -** None

**Notes/Issues -** The system should implement filters to detect offensive language and prevent spam.

### Use Case #5 (Edit/Delete Reviews – U5)

**Author –**

**Purpose -** This use case allows students to edit or delete their previously submitted reviews.

**Requirements Traceability -** The system must allow students to modify or remove their own reviews while preventing unauthorized changes to others’ reviews.

**Priority -** Medium

**Preconditions -** The student must be logged in and have an existing review on a professor’s profile.

**Postconditions -** The updated or deleted review is reflected in the system. If the review is deleted, it is removed from public view.

**Actors -** Student

**Extends -** Post Reviews (U4)

**Flow of Events –**

1. Basic Flow - The student navigates to their submitted review. The student chooses to edit or delete the review. If editing, the student updates the content and saves it. If deleting, the system removes the review from the database.
2. Alternative Flow - If the review was reported for inappropriate content, the system may restrict edits.
3. Exceptions - If the system is down, the student is notified that they cannot edit or delete reviews at the moment.

**Includes -** None

**Notes/Issues -** There should be a history log to track edited reviews for moderation purposes.

### Use Case #6 (Report Inappropriate Reviews – U6)

**Author –**

**Purpose -** This use case allows users to report inappropriate or offensive reviews to be reviewed by an admin.

**Requirements Traceability -** The system must allow users to flag inappropriate content, notify administrators, and provide them with tools to take appropriate action.

**Priority -** High

**Preconditions -** The review must already exist in the system.

**Postconditions -** The reported review is flagged for administrative review, and appropriate action (approval, modification, or deletion) is taken.

**Actors -** Student, Professor

**Extends - -**

**Flow of Events –**

1. Basic Flow - The user finds a review they consider inappropriate. The user clicks "Report" and selects a reason. The system marks the review as "Under Review" and notifies an admin. The admin reviews the report and decides whether to remove, edit, or approve the review.
2. Alternative Flow - If multiple reports are filed for the same review, the system may automatically hide it until an admin makes a decision.
3. Exceptions - If the system fails to notify an admin, the report remains pending until manually reviewed.

**Includes -** None

**Notes/Issues -** A threshold for automatic content removal should be considered if a review receives multiple reports.

### Use Case #7 (View Ratings/Reviews – U7)

**Author -**

**Purpose -** This use case allows users to view ratings and reviews for professors.

**Requirements Traceability -** The system must provide a publicly accessible page where users can browse professor ratings and read student reviews.

**Priority -** High

**Preconditions -** There must be at least one professor with a rating or review in the system.

**Postconditions -** The user successfully views the rating and reviews of the selected professor.

**Actors -** Student, Professor, Guest User

**Extends - -**

**Flow of Events –**

1. Basic Flow - The user searches for a professor. The system displays the professor’s profile, including their ratings and reviews. The user browses through the reviews.
2. Alternative Flow - If no reviews exist, the system displays a message stating that there are no reviews yet.
3. Exceptions - If the system fails to retrieve data, an error message is displayed.

**Includes -** None

**Notes/Issues -** The system should allow filtering reviews by date or rating for better readability.

# Other Non-functional Requirements

## Performance Requirements

**P1:** The system shall respond to user actions (e.g., searching for professors, submitting reviews) within 2 seconds under normal load.

**P2:** The database shall handle at least 1000 concurrent users without performance degradation.

**P3:** The system shall load the homepage within 3 seconds on a standard broadband connection.

**P4:** The search functionality shall return results within 1 second for common queries.

## Safety and Security Requirements

**S1:** All user passwords shall be stored using

**S2:** User authentication shall be implemented using

**S3:** The system shall prevent

**S4:** Professors and students shall have **role-based access control (RBAC)** to prevent unauthorized actions.

**S5:** User data, including reviews and ratings, shall be **encrypted**

**S6:** Admins shall be able to **monitor and flag suspicious activity** to prevent misuse.

## Software Quality Attributes

**4.3.1 Reliability**

* The system shall be available **99.9% of the time** except for scheduled maintenance.
* Regular **database backups** shall be performed every **12 hours** to prevent data loss.
* The system shall have **error logging and automatic recovery mechanisms** to handle unexpected failures.

**4.3.2 Maintainability**

* The codebase shall follow **modular design principles** to allow easy updates and feature additions.
* The system shall be built using **MVC architecture** to separate concerns and facilitate maintenance.
* Automated unit tests shall cover **at least 80%** of the code to ensure stability.

**4.3.3 Usability**

* The user interface shall follow **responsive design principles** to work seamlessly on desktops and mobile devices.
* The system shall provide **tooltips and inline guidance** for new users.
* Users shall be able to **complete a rating submission within 3 clicks** from the homepage.

**4.3.4 Scalability**

* The system shall support **horizontal scaling** by deploying additional instances on cloud infrastructure.
* The database shall use **indexed queries and caching mechanisms** to handle increasing traffic

# Other Requirements

## Database Requirements:

* Uses **MySQL** with **3NF normalization** and **AES-256 encryption** for security.
* Implements **indexing** for fast retrieval and **12-hour backups** to prevent data loss.

## Legal and Compliance:

* **GDPR & FERPA compliant** for data privacy and user rights.
* Requires **user consent** for terms and privacy policies.

## Reusability & Extensibility

* Built with **modular components** for future expansion.
* Follows **RESTful API principles** for integration with other systems.
* Maintains **loose coupling** between frontend and backend for scalability.

Appendix A – Data Dictionary

**1. User Data:**

* **UserID** (Integer) – Unique identifier for each user, auto-generated.
* **Username** (String) – User’s login identifier, must be unique and alphanumeric.
* **Password** (String) – Encrypted user password, minimum 8 characters, stored securely.
* **UserRole** (Enum) – Defines the role of a user in the system. Possible values:
  + Student
  + Professor
  + Admin
* **AuthToken** (String) – Secure authentication token (JWT) used for session management.

**2. Professor and Course Data:**

* **ProfessorID** (Integer) – Unique identifier for each professor, auto-generated.
* **ProfessorName** (String) – Name of the professor, user-entered.
* **CourseID** (Integer) – Unique identifier for each course, auto-generated.
* **Department** (String) – Department to which the professor belongs, user-defined.

**3. Review and Rating Data:**

* **ReviewID** (Integer) – Unique identifier for each review, auto-generated.
* **Rating** (Integer) – Numeric rating given by students. Possible values: 1 to 5.
* **ReviewText** (String) – Text-based feedback submitted by a student (max 500 characters).
* **ProfessorResponse** (String) – Response from a professor to a review (optional, max 500 characters).

**4. Moderation & Admin Actions:**

* **AdminID** (Integer) – Unique identifier for admin users, auto-generated.
* **AdminAction** (String) – Action taken by an admin
* **FlaggedReviews** (Object) – Stores reviews flagged for moderation, containing a list of review IDs.

**5. System Operations and Logging:**

* **SearchQuery** (String) – Text input for searching professors/universities.
* **ComparisonData** (Object) – Stores comparison data for professors or universities, including rating averages and reviews.
* **Timestamp** (DateTime) – Records the time of user actions, auto-generated for logging purposes.

Appendix B - Group Log

<Please include here all the minutes from your group meetings, your group activities, and any other relevant information that will assist in determining the effort put forth to produce this document>