Software Design Specifications

for

Website to Rate Professors – Version 1.0

Prepared by:

Jampani Srihitha
Frontend and Database developer

Document Information

Title: Software Design Specifications for "Website to Rate Professors"	
Project Manager: Akanksha, Hasini, Srihitha.J	**Document Version No:1.0**
	Document Version Date:09/04/2025
Prepared By: Srihitha.J	**Preparation Date:07-04-2025**

Version History

Ver. No.	**Ver. Date**	**Revised By**	**Description**	**Filename**
1.0	09-04-2025	Srihitha.J	Initial Draft	SDD_RateProf

Table of Contents

1 Introduction

This document presents the software design for the "Website to Rate Professors" project. It translates the Software Requirements Specification (SRS) into a detailed technical framework for implementation. It is intended for developers, testers, and stakeholders to understand the system's structure, interactions, and functionality for effective development and maintenance.

1.1 Purpose

The purpose of this Software Design Specification (SDS) document is to provide a comprehensive technical design for the "Website to Rate Professors" application. It outlines the system's architecture, data flow, use case realizations, design models, and quality attributes to ensure consistent implementation aligned with the requirements. Each section is structured to provide clarity on how components of the system interact and function. The intended audience includes software developers responsible for implementing the system, testers verifying system behaviour, and academic stakeholders who require an overview of the system's internal design.

1.2 Scope

This design specification applies to the full-stack web-based system named "Website to Rate Professors." The system provides a platform where students can rate and review professors and universities based on several attributes such as teaching quality, difficulty, and overall experience. Users can view reviews, compare professors and universities, and make informed academic decisions.
The document covers all key components: front-end user interface, backend logic and APIs, data storage structures, external interfaces, exception handling, and quality-of-service considerations. The scope includes both student and professor-facing features and assumes a modular, extensible architecture for future upgrades.

1.3 Definitions, Acronyms, and Abbreviations

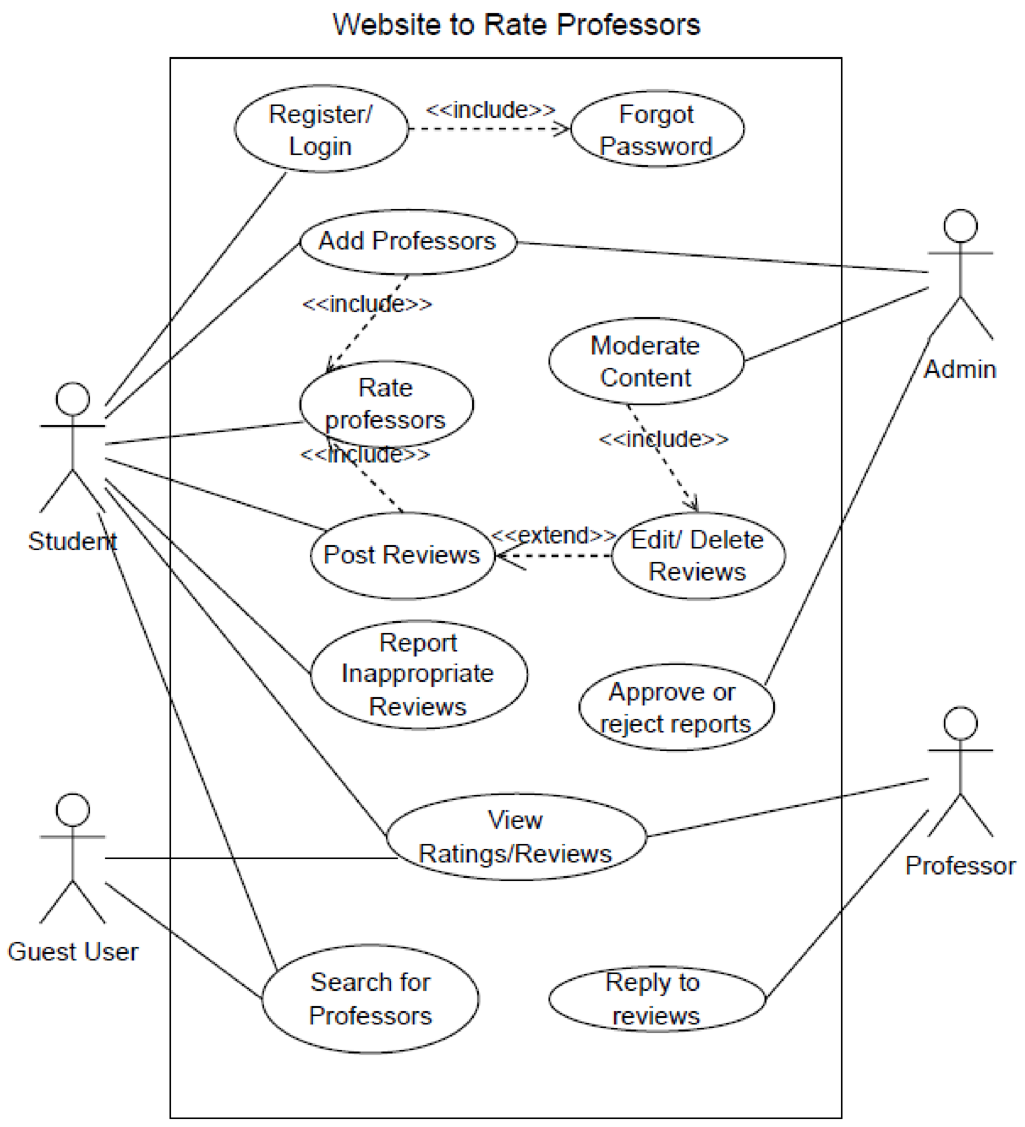
Term/Acronym	**Definition**
UI	User Interface
DB	Database
CRUD	Create, Read, Update, Delete
SRS	Software Requirements Specification
SDS	Software Design Specification
API	Application Programming Interface
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
CDN	Content Delivery Network
REST	Representational State Transfer

1.4 References

- Software Requirements Specification for "Website to Rate Professors"
- IEEE Standard 1016-2009 - IEEE Recommended Practice for Software Design Descriptions
- Firebase Authentication Documentation: https://firebase.google.com/docs/auth
- Google Maps API Reference: https://developers.google.com/maps/documentation

2 Use Case View

This section outlines key use cases that define the core interactions between users and the system. These use cases were identified from the Software Requirements Specification and form the basis of the system design. Each use case highlights major functionality, system interaction, and user roles.



2.1 Use Case

1. User Registration

- **Actors:** Student, Professor
- **Description:** Users can register by providing their email, name, password, and role (student/professor).
- **Steps:**
    1. Navigate to registration page
    2. Fill form and submit

3. Firebase validates and creates user account

2. Login/Logout

- **Actors:** All Users
- **Description:** Allows users to log into and out of the platform securely.
- **Steps:**
    1. Enter email and password
    2. Authentication through Firebase
    3. Redirect to home/profile

3. Submit Rating and Review

- **Actors:** Student
- **Description:** Students can submit a rating and review for professors they've taken.
- **Steps:**
    1. Navigate to professor profile
    2. Select rating criteria
    3. Submit review

4. View Professor and University Profiles

- **Actors:** All Users
- **Description:** Users can view detailed profiles including average ratings, reviews, and stats.
- **Steps:**
    1. Search/select a professor or university
    2. View information dashboard

5. Compare Professors/Universities

- **Actors:** All Users
- **Description:** Allows side-by-side comparison of two professors or universities.
- **Steps:**
    1. Select two entities
    2. System displays comparison table

3 Design Overview

This section presents a high-level overview of the website's software design. It outlines the architecture, design goals, assumptions, and how the system is decomposed into packages and interfaces. The system adheres to the design principles mentioned in the requirements and complies with modularity, maintainability, and extensibility.

3.1 Design Goals and Constraints

- **Goals:**
    - Provide an intuitive, user-friendly platform for rating and reviewing professors.
    - Ensure secure authentication and authorization for students, professors, and admins.
    - Support moderation workflows for content and reviews.
    - Enable scalable search and view functionality for users (students/guests).
- **Constraints:**
    - The project must be completed within a semester timeline.
    - Developed using web technologies (HTML/CSS/JavaScript, with optional backend technologies like Node.js or Python Flask).
    - Must be compatible with desktop and mobile browsers.
    - Admin rights are hardcoded and not role-configurable in version 1.0.

3.2 Design Assumptions

- All users will access the system through modern web browsers.
- The database schema is predefined and normalized for ease of use.
- Admins are trusted users and assumed to act responsibly.
- Users (students/professors) must register to post reviews or reply.
- Internet access is assumed during use (no offline mode).

3.3 Significant Design Packages

Package Name	**Responsibility**
UserAuth	Handles registration, login, and password recovery for all user types.
ProfessorManagement	Allows users to add/view/search professors.
RatingSystem	Manages rating submission, calculation, and view logic.
ReviewSystem	Supports writing, editing, deleting, and replying to reviews.
ModerationSystem	Admin-only module to moderate reviews, handle reports, and take corrective actions.
UIComponents	Reusable UI modules for forms, modals, and search filters.
DatabaseConnector	Interface layer to interact with the backend database.

3.4 Dependent External Interfaces

The table below lists the public interfaces this design requires from other modules or applications.

External Application and Interface Name	**Module Using the Interface**	**Functionality/ Description**
SMTP Server (Email)	UserAuth	Sends password recovery mails to registered users
MySQL	DatabaseConnector	Stores and retrieves data related to users, ratings, reviews
Web Browser	All frontend modules	User interacts via web UI

3.5 Implemented Application External Interfaces (and SOA web services)

The table below lists the implementation of public interfaces this design makes available for other applications.

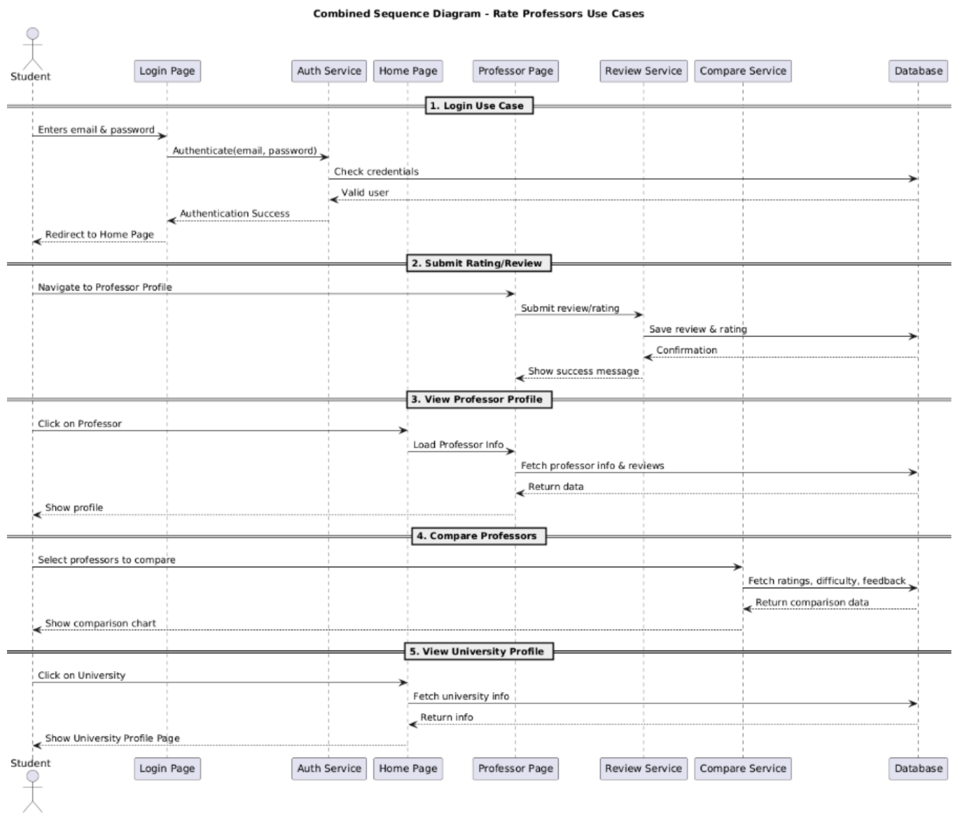
Interface Name	**Module Implementing the Interface**	**Functionality/ Description**

/api/register	UserAuth	Accepts new user registration data
/api/login	UserAuth	Authenticates users based on credentials
/api/add-professor	ProfessorManagement	Allows students to add new professors to the database.
/api/post-review	ReviewSystem	Enables students to post reviews and ratings.
/api/search-professor	ProfessorManagement	Returns professor records based on search queries.
/api/moderate	ModerationSystem	Admin actions for deleting or editing flagged content.

4 Logical View

This section describes the detailed object-oriented design of the system, including the breakdown of modules into classes, their relationships, and collaboration to implement the system's functionality.

4.1 Design Model



4.2 Use Case Realization

Use Case: Register/Login

- **Actors Involved:** Student, Professor, Admin
- **Sequence:**
    1. User fills form → User.register() or User.login()
    2. System validates credentials → Creates session
    3. Redirects to respective dashboard based on role

Use Case: Post Review

- **Actors Involved:** Student
- **Modules Involved:** UserAuth → ReviewSystem
- **Sequence Diagram (in text):**
    1. Student selects a professor → SearchEngine.searchProfessors()
    2. Fills out review form → Review.post()
    3. Review saved in database → ProfessorProfile.updateRating()

Use Case: View Ratings/Reviews

- **Actors Involved:** Student, Professor, Guest
- **Modules:** ProfessorManagement, ReviewSystem
- **Sequence:**
    1. User searches professor → SearchEngine.searchProfessors()
    2. Result links to ProfessorProfile.getProfile()
    3. Reviews fetched and displayed via Review.getReviewsByProfessor()

Use Case: Report Inappropriate Review

- **Actor:** Student
- **Modules:** ReviewSystem, ModerationSystem
- **Sequence:**
    1. Student.reportReview() → sends to ModerationQueue
    2. Admin accesses ModerationQueue.viewReports()
    3. Admin.approve() or reject() the report

Use Case: Moderate Content

- **Actor:** Admin
- **Modules:** ModerationSystem
- **Sequence:**
    1. Admin logs in → accesses flagged reviews
    2. Reviews content → ContentModerator.moderateReview()
    3. Deletes or edits content as necessary

5 Data View

This section provides a detailed view of the system's persistent data architecture. It outlines the domain model that represents core entities and their relationships in the system, followed by the actual data model that defines how this information is stored and managed in the database. The data view ensures that the design aligns with the application's functional requirements and supports efficient data access, integrity, and scalability.

5.1 Domain Model

The domain model consists of the main entities that represent the data involved in the system, along with their relationships:

- **User**
  - Attributes: id, name, email, password, user_type
  - Relationships: A user can submit many reviews and ratings
- **Professor**
  - Attributes: id, name, university_id, department
  - Relationships: A professor can have many reviews
- **University**
  - Attributes: id, name, location
  - Relationships: A university can have many professors
- **Review**
  - Attributes: id, user_id, professor_id, review_text, date
  - Relationships: Belongs to one user and one professor
- **Rating**
  - Attributes: id, review_id, difficulty, quality, would_take_again, for_credit, attendance
  - Relationships: Tied to a single review

5.2 Data Model (persistent data view)

The persistent data model for the Rate Professors platform captures key entities such as users, universities, departments, professors, ratings, comparisons, and reported reviews. The system uses a relational database with foreign key constraints to ensure data integrity and to maintain associations between users, professors, universities, and their respective ratings.

The core entities include Users, Professors, Universities, Departments, ProfessorRatings, and UniversityRatings. Supplementary tables such as LoginCredentials, ProfessorComparisons, UniversityComparisons, and ReportedReviews provide support for authentication, comparisons, and moderation features.

The ER model follows a normalized structure to reduce redundancy and improve consistency. Relationships are enforced using foreign keys, particularly for user-generated content linked to the respective student, professor, or university entity.

5.2.1 Data Dictionary

Table Name	Column Name	Data Type	Description
Users	user_id	INT (PK)	Unique ID for each user
	full_name	VARCHAR(100)	Full name of the user
	email	VARCHAR(100)	Email address, must be unique
	user_type	ENUM	Indicates if user is 'student' or 'professor'
	date_joined	DATE	Date when the user joined
LoginCredentials	login_id	INT (PK)	Unique ID for login entry
	user_id	INT (FK)	Foreign key referencing Users

Table Name	Column Name	Data Type	Description
	username	VARCHAR(50)	Unique username
	password_hash	VARCHAR(255)	Encrypted password
Universities	university_id	INT (PK)	Unique ID for each university
	university_name	VARCHAR(255)	Unique name of the university
	location	VARCHAR(100)	Location of the university
	established_year	YEAR	Year the university was founded
Departments	department_id	INT (PK)	Unique ID for each department
	university_id	INT (FK)	Foreign key referencing Universities
	department_name	VARCHAR(100)	Name of the department
Professors	professor_id	INT (PK)	Unique ID for each professor
	user_id	INT (FK)	Links professor to their user account
	department_id	INT (FK)	Foreign key referencing Departments
	professor_name	VARCHAR(100)	Name of the professor
ProfessorRatings	rating_id	INT (PK)	Unique ID for each rating
	professor_id	INT (FK)	Foreign key referencing Professors
	student_id	INT (FK)	Foreign key referencing Users
	rating	DECIMAL(2,1)	Overall rating (1.0 to 5.0)
	would_take_again	BOOLEAN	Indicates whether student would take the professor again
	difficulty	TINYINT	Difficulty level (1–5)
	quality	TINYINT	Quality rating (1–5)
	for_credits	BOOLEAN	Whether course was taken for credits
	attendance	BOOLEAN	Whether attendance was mandatory
	review	TEXT	Optional written review
	rated_on	DATE	Date the rating was submitted
UniversityRatings	rating_id	INT (PK)	Unique ID for each university rating
	university_id	INT (FK)	Foreign key referencing Universities
	student_id	INT (FK)	Foreign key referencing Users
	rating	DECIMAL(2,1)	Overall university rating
	location_rating	TINYINT	Rating for location
	reputation	TINYINT	Reputation score
	opportunity	TINYINT	Opportunity score
	happiness	TINYINT	Happiness index
	internet	TINYINT	Internet facility rating
	faculty	TINYINT	Faculty rating
	clubs	TINYINT	Clubs and extracurriculars rating
	social	TINYINT	Social life rating
	food	TINYINT	Food quality
	safety	TINYINT	Safety level
	review	TEXT	Optional written review

Table Name	Column Name	Data Type	Description
	rated_on	DATE	Date the rating was submitted
ProfessorComparisons	comparison_id	INT (PK)	Unique ID for each professor comparison
	student_id	INT (FK)	Foreign key referencing Users
	prof1_id	INT (FK)	First professor being compared
	prof2_id	INT (FK)	Second professor being compared
	compared_on	TIMESTAMP	Timestamp when comparison was made
UniversityComparisons	comparison_id	INT (PK)	Unique ID for each university comparison
	student_id	INT (FK)	Foreign key referencing Users
	uni1_id	INT (FK)	First university being compared
	uni2_id	INT (FK)	Second university being compared
	compared_on	TIMESTAMP	Timestamp when comparison was made
ReportedReviews	report_id	INT (PK)	Unique ID for each report
	reporter_id	INT (FK)	User who reported the review
	professor_rating_id	INT (FK, nullable)	Optional FK referencing ProfessorRatings
	university_rating_id	INT (FK, nullable)	Optional FK referencing UniversityRatings
	reason	TEXT	Reason for reporting
	reported_on	TIMESTAMP	When the report was submitted
	status	ENUM	Current status of the report ('pending', 'reviewed', 'dismissed')

6 Exception Handling

The website handles exceptions gracefully to ensure a smooth user experience and secure system behaviour. Common exceptions and their handling strategies are as follows:

- **Authentication Errors:**
  Circumstance: Invalid login credentials or unauthorized access attempts.
  Handling: User is shown an appropriate error message (e.g., “Invalid username or password”). Login attempts are logged for auditing.
- **Input Validation Errors:**
  Circumstance: Missing or incorrectly formatted form inputs (e.g., empty review fields, invalid rating values).
  Handling: Client-side and server-side validation with real-time feedback; error messages are shown without reloading the page.
- **Database Exceptions:**
  Circumstance: Duplicate entries (e.g., existing email or username), failed foreign key constraints, or unavailable database.
  Handling: Errors are caught and logged on the backend; user is informed with a generic failure message.

- **Rating/Review Errors:**
  Circumstance: Submitting ratings for non-existent professors/universities or duplicate submissions.
  Handling: Backend checks and responds with status codes; UI disables already rated entities for the user.
- **Comparison Errors:**
  Circumstance: Invalid professor/university IDs passed for comparison.
  Handling: System checks for valid input and alerts the user if invalid.
- **Unhandled Exceptions:**
  Handling: Logged via a centralized logging system (e.g., console logs, file logs); user is shown a generic error message ("Something went wrong. Please try again.").

7 Configurable Parameters

This section outlines the configurable parameters used in the application that can be modified based on deployment environment or operational requirements.

This table describes the simple configurable parameters (name / value pairs).

Configuration Parameter Name	**Definition and Usage**	**Dynamic?**
session_timeout_minutes	Duration (in minutes) after which an inactive user session expires.	Yes
max_review_length	Maximum number of characters allowed in the review text field	Yes
password_min_length	Minimum number of characters required in a user's password.	Yes
allow_anonymous_ratings	Boolean flag to allow/disallow anonymous ratings submission.	Yes
max_login_attempts	Maximum allowed failed login attempts before locking the account.	No
default_rating_scale	Defines the upper limit of rating scale (e.g., 5 for a 1–5 scale).	Yes
review_flag_threshold	Number of reports required before a review is flagged for admin attention.	Yes
database_backup_path	File path where automated database backups are stored.	No

8 Quality of Service

This section outlines the quality attributes built into the application to ensure reliable, secure, and high-performing operation in both development and production environments.

8.1 Availability

 The application is designed with minimal downtime in mind. Key features like asynchronous data loading and modular front-end components ensure the website remains responsive even during partial failures or background updates.

□ Scheduled database backups are configured to run during off-peak hours to avoid downtime.
□ Maintenance activities such as data cleanup or report generation are handled during low-traffic windows and displayed via banners to users when applicable.

8.2 Security and Authorization

□ **Role-based access control (RBAC)**: Different access privileges are defined for students and professors (e.g., only students can rate, and only professors can view aggregated feedback).
□ Passwords are stored using secure hashing (e.g., bcrypt).
□ Input validations and sanitization prevent SQL injection, XSS, and CSRF attacks.
□ Admin-level users have the ability to manage reported reviews and user accounts.
□ Login lockout after multiple failed attempts ensures brute-force protection.

8.3 Load and Performance Implications

□ The application is expected to handle light-to-moderate load in early stages (~200 concurrent users).
□ Database indexes are used for fast retrieval of popular queries (e.g., top-rated professors, reviews per university).
□ Lazy loading is implemented for long lists of reviews and search results to reduce front-end rendering time.
□ Projected data growth:

- ~5000 user accounts in year one
- ~10,000 ratings
- ~200 universities and ~2000 professors

8.4 Monitoring and Control

□ Logs are maintained for key actions like login attempts, review submissions, and error messages.
□ Admin dashboard includes metrics like:

- Number of new users per day
- Most-rated professors
- Number of reported reviews

□ Server uptime and health are monitored using third-party tools (e.g., UptimeRobot, Pingdom).
□ Email alerts are configured for login anomalies and failed database backups.