Software Design Specifications

for

<TEACHTRACK 1>

Prepared by:

Shreya Seshadri, se22uari159, Project Manager & UI/UX Designer

Abhiram Ayla, se22uari005, Frontend Developer

Aarnav Tandava, se22uari002, Backend Developer& Database Manager

Kandala Mahitha Siri, se22uari066, Security & Moderation Specialist

P Jotsna Sree, se22uari123, Testing &Quality Assurance (QA) Engineer

Document Information

Title: Professor Rating Website	
Project Manager: Mahitha Sir	**Document Version No:**
	Document Version Date:
Prepared By : Team Profyze	**Preparation Date:**

Version History

Ver. No.	Ver. Date	Revised By	Description	Filename

Table of Contents

1. Introduction

This Software Design Specification (SDS) outlines the structure and design of the **Professor Rating Website**, a web-based application that enables students to rate and review their professors based on parameters such as teaching quality, clarity, grading style, and approachability. The SDS provides detailed information about the architecture, data design, and system components required to implement the proposed system.

The document is intended to be used by software developers, designers, testers, and stakeholders involved in the project. It ensures that all involved parties share a common understanding of the design strategy and software components, thus enabling effective implementation and future maintenance.

1.1 Purpose

The purpose of this Software Design Specification is to provide a comprehensive description of the design of the Professor Rating Website. This document serves as a bridge between the software requirements and the actual implementation, ensuring that the system is built according to the needs identified during requirements analysis.

The intended audience includes:

- **Developers**, for understanding system components and implementation logic.
- **Testers**, to develop test cases based on design specifications.
- **Project Managers**, to monitor development progress and evaluate design choices.
- **Instructors and Reviewers**, for academic evaluation and feedback.

1.2 Scope

This document covers the design of a full-stack web application titled **Professor Rating Website**. It includes the front-end UI design, back-end server structure, database schema, and integration points. The system allows users (students) to sign up, log in, search for professors by name or institution, view ratings and reviews, and submit their own feedback. It also includes an admin interface for moderating content.

The document affects:

- The design and architecture of the application
- The data models and storage
- The logic for rating calculations and filtering
- The interaction between user interface and backend services

1.3 Definitions, Acronyms, and Abbreviations

Term / Acronym	Definition
SDS	Software Design Specification
UI	User Interface
UX	User Experience
DB	Database
API	Application Programming Interface
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
DBMS	Database Management System
Admin	Administrator of the platform

1.4 References

- W3Schools - HTML – Reference for HTML elements and structure
- W3Schools - CSS – Reference for styling with CSS
- W3Schools - JavaScript – Reference for client-side scripting
- PHP Manual – Official PHP documentation for server-side scripting
- MySQL Documentation – Official MySQL documentation for database design and queries

2. Use Case View

The **Professor Rating Website** is centered around the core interactions between students and the system, as well as administrative maintenance operations. The system enables students to register,

search for professors, provide feedback, and view professor ratings. It also empowers administrators to manage data and moderate content.

The following use cases cover major functional requirements and are central to the design of the system.

2.1 Use Case: Log In

Actors: Student, Professor
Description:

Users (students or professors) log into the system using credentials.

Includes:

- Verify Record (checks against database)
- Manage Professor Records (if actor is a professor)
- Manages Leaderboard (admin-level or automated action)
- Prepare Professor Database (ensures searchable professor info is ready)

Extends:

- Display Login Error (if credentials are incorrect)

Steps:

1. User enters username and password.
2. System verifies credentials.
3. If valid, user is directed to their dashboard.
4. If invalid, system shows an error message.

2.2 Use Case: Search College/Professor

Actors: Student
Description:

Students can search for specific professors or colleges.

Steps:

1. Student enters search query.
2. System fetches matching results from the professor database.

3. Student selects a result to view detailed profile or ratings.

Use Case: Check Rating/Leaderboard

Actors: Student

Description:

Students can view professors' average ratings and rankings.

Steps:

1. Student navigates to the Leaderboard.
2. System displays a list of professors ranked by ratings.

2.3 2.4Use Case: Submit Anonymous Comments

Actors: Student

Description:

Students can post comments about a professor anonymously.

Includes:

- Rules a Professor (validates comment)
- Add to Database (stores the comment)

Steps:

1. Student selects a professor profile.
2. Fills in an anonymous comment form.
3. System validates and saves the comment.

2.4 Use Case: Professor Register

Actors: Professor

Description:

Professors can create an account to manage their profiles.

Includes:

- Rules a Professor (validates comment)
- Add to Database (stores the comment)

Steps:

1. Student selects a professor profile.
2. Fills in an anonymous comment form.
3. System validates and saves the comment.

2.5 Use Case: Professor Register

Actors: Professor
Description:
Professors can create an account to manage their profiles.

Includes:

- Fill up Registration Form
- Add Details to Database
- Prepare Professor Database

Steps:

1. Professor fills out a registration form.
2. System stores their details.
3. Profile is made available in the professor database.

2.6 Use Case: Manage Professor Records

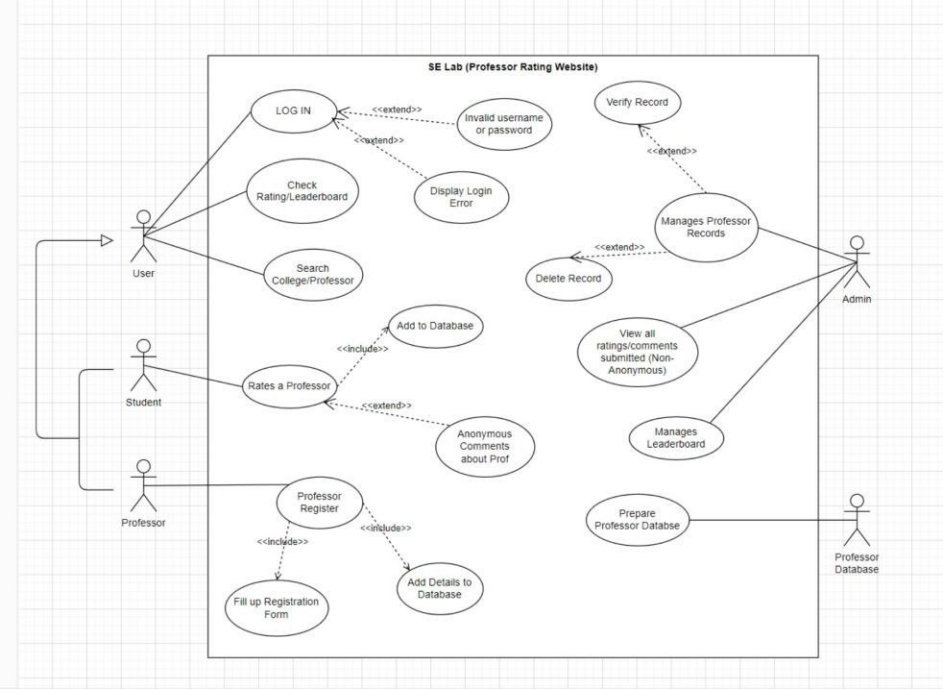
Actors: Professor
Description:
Professors can edit their information or view feedback.

Includes:

- View All Ratings/Comments (only non-anonymous ones)
- Delete Record (if needed)

Steps:

1. Professor logs in.



2. Accesses their profile management page.

3. Views, edits, or deletes records as necessary.

3.Design Overview

The Professor Rating System is a web-based application designed for students to rate and review professors based on specific parameters such as teaching quality, clarity, and approachability. The frontend is built using HTML, CSS, and JavaScript. The backend is developed in PHP, and SQL is used for structured data storage.

The system follows a modular, component-based design. It includes user authentication, professor management, review submission, review display, and admin moderation. Communication between frontend and backend is handled via AJAX. Firebase Authentication is integrated for secure login using college-issued email addresses.

This design supports clarity, maintainability, and ease of feature enhancement.

3.1 Design Goals and Constraints

Goals:

- Allow students to submit professor reviews in a simple and intuitive interface.
- Display average ratings and feedback for each professor.
- Ensure only verified students (via college email) can post reviews.
- Provide admin moderation to manage flagged or inappropriate content.

Constraints:

- Technology stack is limited to HTML, CSS, JavaScript (frontend), PHP (backend), and SQL (database).
- Firebase is used only for user authentication.
- Project timeline is limited to the academic semester.
- No external frameworks or libraries beyond basic frontend scripting and Firebase integration.

3.2 Design Assumptions

- Users will have access to modern browsers and stable internet.
- College email authentication via Firebase ensures genuine user base.
- Professor data will be preloaded into the system before deployment.
- Each student is allowed to post one review per professor.
- Admins are responsible for maintaining quality and decorum of the review content.

3.3 Significant Design Packages

Package	Description
Authentication Module	Handles sign-up and login using Firebase email authentication.
Review Module	Manages creation, editing, deletion, and display of reviews.
Professor Module	Maintains professor data like name, department, and aggregates rating scores.

Admin Module	Allows review moderation and user management for flagged or abusive content.
Search Module	Enables searching and filtering professors by name, department, or rating.

Each module is structured for reusability and minimal coupling with others.

3.4 Dependent External Interfaces

External Module	Interface Name	Description
Firebase Authentication	Email Auth API	Used to authenticate users via college-issued email.
MySQL (or equivalent SQL DB)	SQL Queries	Used for storing and retrieving professor data, reviews, and user activity.
Email Notification (optional)	SMTP (PHP Mailer)	Can be used for sending admin alerts or confirmation messages.

These interfaces are used at the controller level in PHP to support core system functions.

3.5 Implemented Application External Interfaces

Interface Name	Module Implementing the Interface	Description
`login.php`	Authentication Module	Handles user login using Firebase and initializes session.
`submit_review.php`	Review Module	Accepts review data and stores it in the SQL database after validation.
`get_reviews.php`	Review Module	Retrieves and displays reviews for a selected professor.
`search_professors.php`	Search Module	Returns professor list based on search input or selected filters.

`moderate_review.php`	Admin Module	Allows admin to delete or approve flagged reviews.

These PHP scripts act as API endpoints for asynchronous AJAX requests from the frontend.

4 Logical View

The **Logical View** represents the static structure of the TeachTrack system from an object-oriented perspective. It captures the key components and their relationships using class abstractions that model real-world entities such as users, professors, reviews, and admin reports.

Key Modules and Classes:

- **User Module**
    - User: Base class with attributes like userID, name, email, password, and role.
    - Responsibilities: Handles login, registration, and authentication.
- **Professor Module**
    - Professor: Represents teaching staff with attributes like professorID, name, department, and averageRating.
    - Responsibilities: Maintains information about professors and allows for aggregation of reviews.
- **Review Module**
    - Review: Contains reviewID, studentID, professorID, rating, comment, timestamp, isFlagged.
    - Responsibilities: Stores feedback submitted by students, linked to users and professors.
- **Report Module**
    - Report: Tracks reports filed against reviews with reportID, reviewID, reportedBy, reason, and status.
    - Responsibilities: Supports review moderation workflow handled by admin users.

Relationships:

- User has a one-to-many relationship with Review (students can submit multiple reviews).
- Review has a many-to-one relationship with Professor.
- Report is linked to both Review and User (reporting student and targeted review).

4.1 Design ModelThe **TeachTrack** system is designed using an object-oriented approach, with the software decomposed into three main modules: **User Management**, **Professor & Review Handling**, and **Analytics**. Each module contains key classes that fulfill specific responsibilities.

- The **User** class serves as a base for both **Student** and **Admin**, handling authentication and role-based access.
- The **Professor** class stores faculty information and maintains an updated average rating based on associated reviews.
- The **Review** class links students and professors, enabling feedback submission, editing, and moderation.
- The **AnalyticsEngine** class is responsible for processing review data and generating meaningful insights for admin users.

4.2 Use Case Realization

The key use cases in the **TeachTrack** system—such as user login, submitting a professor review, and admin moderation—are realized through well-defined interactions between the system's modules and classes.

- For **User Login**, the system interacts with the authentication service to validate credentials and initiate a session based on the user's role.
- In **Submit Review**, the student inputs feedback which is handled by the review module, then linked to the corresponding professor and reflected in the updated average rating.
- During **Admin Moderation**, flagged reviews are retrieved by the admin, who can then approve or delete them based on platform policies.

5 Data View

The **TeachTrack** system maintains a structured and persistent data storage model to support its core functionalities, including user management, professor details, reviews, and admin actions.

The system uses a **relational database** to store and manage the following key entities:

- **Users Table**: Stores user credentials and profile information, including role distinctions (student or admin).
- **Professors Table**: Contains faculty details such as name, department, and aggregated rating.
- **Reviews Table**: Records student feedback linked to both user and professor, including ratings, comments, timestamps, and report status.
- **Reports Table**: Tracks reviews flagged by users for admin moderation.

5.1 Domain Model

The **TeachTrack** domain model represents the key entities involved in persistent data storage and their relationships. These entities reflect real-world components like users, professors, and reviews.

- **User**
  Attributes: userID, name, email, password, role (Student/Admin)
  Relationships:
    - Can **submit** multiple reviews
    - Can **report** inappropriate reviews
- **Professor**
  Attributes: professorID, name, department, averageRating
  Relationships:
    - Can **receive** multiple reviews
- **Review**
  Attributes: reviewID, studentID, professorID, rating, comment, timestamp, isFlagged
  Relationships:
    - **Belongs to** a student
    - **Targets** a professor
- **Report**
  Attributes: reportID, reviewID, reportedBy, reason, status
  Relationships:
    - **Linked to** a specific review
    - **Handled by** an admin

This domain model helps bridge the design and data layers, ensuring accurate mapping of objects to the database and smooth data transfer across components.

5.2 Data Model (persistent data view)

The **TeachTrack** data model outlines how persistent data is structured and stored in the underlying database system. It aligns closely with the domain model and is designed using a **relational database** approach to ensure consistency, integrity, and scalability.

Core Tables and Relationships:

- **Users Table**
  Columns: user_id (PK), name, email, password, role
  Purpose: Stores information for all system users, including students and admins.
- **Professors Table**
  Columns: professor_id (PK), name, department, average_rating
  Purpose: Stores data about professors, updated dynamically based on reviews.
- **Reviews Table**
  Columns: review_id (PK), student_id (FK), professor_id (FK), rating, comment, timestamp, is_flagged
  Purpose: Captures student feedback and links it to both users and professors.

- **Reports Table**
  Columns: report_id (PK), review_id (FK), reported_by (FK), reason, status
  Purpose: Tracks reviews flagged for moderation and their resolution status.

5.2.1 Data Dictionary

Field Name	Data Type	Description
user_id	*INTEGER (PK)*	*Unique identifier for each user*
name	*VARCHAR*	*Full name of the user or professor*
email	*VARCHAR*	*User's email address (used for login)*
password	*VARCHAR*	*Hashed password for secure authentication*
role	*VARCHAR*	*Role of the user: 'student' or 'admin'*
professor_id	*INTEGER (PK)*	*Unique identifier for each professor*
department	*VARCHAR*	*Department the professor belongs to*
average_rating	*FLOAT*	*Average rating based on student reviews*
review_id	*INTEGER (PK)*	*Unique identifier for each review*
student_id	*INTEGER (FK)*	*Foreign key referencing the student who submitted review*
professor_id	*INTEGER (FK)*	*Foreign key referencing the reviewed professor*
rating	*INTEGER*	*Numeric score provided by student*
comment	*TEXT*	*Optional feedback written by student*
timestamp	*DATETIME*	*Date and time the review was submitted*
is_flagged	*BOOLEAN*	*Flag to indicate if the review has been reported*
report_id	*INTEGER (PK)*	*Unique identifier for each report*
review_id	*INTEGER (FK)*	*Foreign key referencing the review being reported*
reported_by	*INTEGER (FK)*	*Foreign key referencing the user who reported the review*

reason	*TEXT*	*Justification or reason for reporting the review*
status	*VARCHAR*	*Current state of report: 'pending', 'approved', 'rejected'*

6 Exception Handling

Defined Exceptions and Handling

1. **InvalidLoginException**
    a. **When**: User enters incorrect email or password.
    b. **Handling**: Displays a user-friendly error message (“Invalid credentials”).
    c. **Logging**: Logged with timestamp and IP address.
2. **UnauthorizedAccessException**
    a. **When**: A user tries to access restricted admin features without permission.
    b. **Handling**: Redirects to homepage with an alert.
    c. **Logging**: Logs user ID and attempted endpoint.
3. **ProfessorNotFoundException**
    a. **When**: A professor ID requested does not exist in the database.
    b. **Handling**: Displays “Professor not found” message.
    c. **Logging**: Logs as a warning.
4. **RatingSubmissionException**
    a. **When**: There is an error saving the rating due to backend or validation failure.
    b. **Handling**: Shows a message asking the user to retry.
    c. **Logging**: Logs payload and error for debugging.
5. **DatabaseConnectionException**
    a. **When**: Failure to connect to the database.
    b. **Handling**: Shows a fallback error message.
    c. **Logging**: Logs critical error and alerts the development team.

Logging and Follow-Up

- All exceptions are logged using a centralized logger with details like timestamp, user info, and error type.
- Repeated or critical exceptions trigger internal alerts for quick resolution.
- User-facing messages remain simple and do not expose technical details.

7 Configurable Parameters

This section describes the configurable parameters used in the TeachTrack application. These parameters can be modified to adjust the behavior of the system without altering the core source code. Some parameters are dynamic and can be changed without restarting the application, while others require a restart.

Simple Configuration Parameters

Configuration Parameter Name	Definition and Usage	Dynamic?
DATABASE_URL	*Defines the connection string for the primary database used to store all application data.*	*No*
SESSION_TIMEOUT	*Duration (in minutes) of user inactivity after which the session expires.*	*Yes*
DEFAULT_RATING_SCALE	*Sets the maximum rating scale (e.g., 5 or 10) used in the feedback form.*	*Yes*
ENABLE_ANONYMOUS_RATING	*Boolean flag to enable or disable anonymous rating submissions.*	*Yes*
MAX_UPLOAD_SIZE_MB	*Maximum allowed upload size (in MB) for profile pictures.*	*No*
EMAIL_SUPPORT_ADDRESS	*Email address shown for support-related queries.*	*Yes*
MAINTENANCE_MODE	*If set to true, displays a maintenance message and disables all core features.*	*No*

8 Quality of Service

8.1 Availability

Design Requirements:

- ***99.9% uptime*** *(aligned with SRS Section 4.3.1).*
- *Automated failover for critical components.*
- *Maintenance windows during non-peak hours (e.g., weekends).*

Implementation:

- ***Redundant Servers****: Deploy load-balanced web servers (e.g., AWS EC2 instances) to handle traffic spikes.*
- ***Database Replication****: Use MySQL master-slave replication to prevent single points of failure.*
- ***Automated Backups****: Daily encrypted backups (SRS 4.3.1) stored in geographically separate locations.*
- ***Graceful Degradation****: If the database is unavailable, the UI will display cached ratings with a warning message.*

Downtime Scenarios:

- ***Planned****: Database schema updates (max 1 hour/month).*
- ***Unplanned****: Cloud provider outages (mitigated via multi-region deployment in future phases).*

8.2 Security and Authorization

Design Requirements (from SRS 4.2):

- ***Password hashing*** *(bcrypt/PBKDF2).*
- ***Role-Based Access Control (RBAC)*** *for Students, Professors, and Admins.*
- ***SSL/TLS encryption*** *for all data in transit.*

Implementation:

1. ***Authentication:***
   - ***JWT Tokens****: Stateless session management with short-lived tokens (refresh tokens for extended sessions).*
   - ***Rate Limiting****: Lock accounts after 5 failed login attempts (SRS F12).*
2. ***Authorization:***
   - ***RBAC Matrix****:*

Role	***Permissions***
Student	*Submit/edit reviews (within 24h), view leaderboard, search professors.*
Professor	*View own ratings (no edit/delete), update profile details.*
Admin	*Moderate reviews, purge fake accounts, generate analytics reports.*

   -
   - ***API Gateways****: Enforce role checks before processing sensitive requests (e.g., review deletion).*
3. ***Data Protection:***
   - ***XSS/CSRF Prevention****: Sanitize user input (SRS 4.2.S4) using htmlspecialchars() and CSRF tokens.*
   - ***SQL Injection Mitigation****: Prepared statements with PDO (SRS 2.3.2).*

8.3 Load and Performance Implications

Design Requirements (from SRS 4.1):

- *Support* ***1,000 concurrent users*** *(P2).*
- ***<3s page load*** *(P1) under 10 Mbps bandwidth.*
- ***<500ms search response*** *for 100K professors (P3).*

Scalability Strategies:

- ***Frontend***:
  - *Lazy-load images/ratings.*
  - *Cache static assets via CDN (e.g., Cloudflare).*
- ***Backend***:
  - ***Horizontal Scaling****: Auto-scaling PHP-FPM pools based on CPU usage.*
  - ***Database***:
    - *Indexes on Professor_ID, Course_ID, and Rating columns.*
    - *Query optimization (e.g., pagination for leaderboard queries).*
- ***Stress Testing***:
  - *Simulate 1,500 users with Locust to validate auto-scaling triggers.*

Growth Projections:

- ***Database Size****: ~50MB/year (assuming 10K new reviews/year at 5KB each).*

8.4 Monitoring and Control

Key Metrics:

- ***Application****: Response time (per API endpoint), error rates (4xx/5xx).*
- ***Database****: Query latency, connection pool usage.*
- ***Security****: Failed login attempts, flagged reviews.*

Tools & Processes:

1. ***Real-time Monitoring***:
   - ***Prometheus + Grafana****: Track server health and SLA compliance (e.g., uptime).*
   - ***Logging****: Centralized logs (ELK Stack) for audit trails (SRS S7).*
2. ***Alerts***:
   - *Slack/Email notifications for:*
     - *Server CPU >80% for 5+ minutes.*
     - *10 failed login attempts/minute (potential brute-force attacks).*
3. ***Admin Controls***:
   - *Manual review queue for flagged content (SRS F8).*
   - *Emergency "maintenance mode" toggle to disable review submissions.*