# Software Design Specifications

# for

# <Campus Critics: Website to rate professors 1.1>

Prepared by: Team 13

Kuruva Bhuvana Vijaya
Madarapu Sri Harshitha
Laxmi Sowmya Niharika Peddireddi

**Document Information**

| | |
|---|---|
| **Title: Campus Critics** | |
| **Project Manager: *Murali Krishna Bukkasamudram*** | **Document Version No:1.1** |
| | **Document Version Date:08-04-2025** |
| **Prepared By:Laxmi Sowmya Niharika Peddireddi** | **Preparation Date:08-04-2025** |

**Version History**

| Ver. No. | Ver. Date | Revised By | Description | Filename |
|---|---|---|---|---|
| 1.0 | 01-03-2025 | Team | Initial draft of the Software Design Document. | Team13_SRS Document |
| 1.1 | 08-04-2025 | Laxmi Somwya Niharika Peddireddi | Software Design Document | Team13_SoftwareDesignDocument |

## Table of Contents

# 1 Introduction

*This document explains the design of the **Rate My Professor** web application. It includes how the system works, what features it has, and how different parts connect. The goal is to help developers, testers, and future team members understand how the system is built and how to maintain or improve it.*

## 1.1 Purpose

*The purpose of this document is to describe the design of the Rate My Professor system. It helps guide the development process and makes sure everyone working on the project is on the same page. It is mainly for developers, testers, and project stakeholders.*

## 1.2 Scope

*This document covers the design of the main features of the website:*

1. *User registration and login with OTP*
2. *Professor rating system with a detailed questionnaire*
3. *User dashboard to view personal info and past reviews*
4. *Community section for students to talk and share*
5. *Backend using Flask and SQLite*
6. *Frontend using HTML, CSS, and JavaScript (Bootstrap)*

*It does not include large-scale deployment or advanced performance tuning*

## 1.3 Definitions, Acronyms, and Abbreviations

| *TERM :* | *DEFINITION:* |
|---|---|
| *OTP* | *one time password* |
| *DB* | *database* |
| *FLASK* | *framework used for backend* |
| *SQLite* | *Lightweight Database* |
| *HTML/CSS/JAVASCRIPT* | *Programming Language used for Frontend* |

## 1.4 References

*SRS Document:*
*Team13_SRS Document - Google Docs*

# 2 Use Case View

*[Based on the requirements document, this section lists use cases or scenarios from the use-case model if they represent some significant, central functionality of the final system, or if they have a large design coverage - they exercise many design elements, or if they stress or illustrate a specific, delicate point of the design. Provide a use case diagram for use cases that pertains to the software design]*

## 2.1 Use Case

*Each use case describes a single feature from the user's point of view.*

### *U1: Login*

- ***Author**: Kuruva Bhuvana Vijaya*
- ***Purpose**: Allows students to log in with student ID and password.*
- ***Actors**: Student*
- ***Includes**: Verify Password*
- ***Extends**: Login Error*
- ***Preconditions**: Must be registered*
- ***Postconditions**: User is logged in, session created*
- ***Notes**: Consider adding multi-factor authentication*

---

### *U2: Register*

- ***Author**: Laxmi Sowmya Niharika Peddireddi*
- ***Purpose**: New students can register using student ID and email*
- ***Actors**: Student, Email Service*
- ***Includes**: Verify Password*
- ***Preconditions**: Valid ID and email*
- ***Postconditions**: Student account created*
- ***Notes**: Add CAPTCHA to avoid bots*

---

### *U3: Sort & Filter Professors*

- ***Author***: *Kuruva Bhuvana Vijaya*
- ***Purpose***: *Lets students find professors using filters*
- ***Actors***: *Student*
- ***Preconditions***: *Must be logged in*
- ***Postconditions***: *Filtered list shown*

---

### *U4: Rate a Professor*

- ***Author***: *Madarapu Sri Harshitha*
- ***Purpose***: *Submit ratings and feedback*
- ***Actors***: *Student*
- ***Preconditions***: *Must be logged in*
- ***Postconditions***: *Rating saved*

---

### *U5: View Professor Ratings*

- ***Author***: *Laxmi Sowmya Niharika Peddireddi*
- ***Purpose***: *View existing ratings for a professor*
- ***Actors***: *Student*
- ***Preconditions***: *Logged in*
- ***Postconditions***: *Ratings displayed*

---

### *U6: Create a Post*

- ***Author***: *Madarapu Sri Harshitha*
- ***Purpose***: *Write community posts*
- ***Actors***: *Student*
- ***Preconditions***: *Logged in*
- ***Postconditions***: *Post saved*

---

### *U7: Comment on Posts*

- ***Author**: Kuruva Bhuvana Vijaya*
- ***Purpose**: Add comments to posts*
- ***Actors**: Student*
- ***Extends**: Reply to Comments*
- ***Preconditions**: Logged in*
- ***Postconditions**: Comment saved*

---

### *U8: Reply to Comments*

- ***Author**: Laxmi Sowmya Niharika Peddireddi*
- ***Purpose**: Reply to other students' comments*
- ***Actors**: Student*
- ***Preconditions**: Logged in*
- ***Postconditions**: Reply posted*

---

### *U9: Like/Upvote Posts*

- ***Author**: Madarapu Sri Harshitha*
- ***Purpose**: Like or upvote posts*
- ***Actors**: Student*
- ***Preconditions**: Logged in*
- ***Postconditions**: Like recorded*

---

### *U10: Manage Professors*

- ***Author**: Kuruva Bhuvana Vijaya*
- ***Purpose**: Admin adds/edits/deletes professor info*

- ***Actors***: *Admin*
- ***Preconditions***: *Logged in as admin*
- ***Postconditions***: *Professor list updated*

---

### *U11: Moderate Reviews*

- ***Author***: *Laxmi Sowmya Niharika Peddireddi*
- ***Purpose***: *Admin approves or removes reviews*
- ***Actors***: *Admin*
- ***Preconditions***: *Logged in as admin*
- ***Postconditions***: *Review moderated*

---

### *U12: Moderate Community*

- ***Author***: *Madarapu Sri Harshitha*
- ***Purpose***: *Admin controls community content*
- ***Actors***: *Admin*
- ***Preconditions***: *Logged in as admin*
- ***Postconditions***: *Posts/comments moderated*

# 3 Design Overview

*The Campus Critics system is designed using the Flask framework, following the Model-View-Controller (MVC) architecture. It provides a platform where students can log in via OTP, rate professors, view feedback, and participate in a community. The design supports modular development, easy integration with external services (like email), and a lightweight database for small-scale deployment.*

## 3.1 Design Goals and Constraints

***Design Goals:***

- *Build an intuitive, easy-to-navigate system for students.*
- *Ensure data persistence for professor details and reviews.*
- *Allow secure user authentication via OTP without traditional passwords.*
- *Maintain modularity for scalability and maintainability.*

***Constraints:***

- *The system uses SQLite, which limits scalability and concurrent access.*
- *OTPs are managed via email; requires internet and functioning SMTP.*
- *Email IDs are used as identity; no advanced user verification is implemented.*
- *Deployment environment is assumed to be local or basic cloud (like Heroku Free).*

## 3.2 Design Assumptions

- *All users are students with valid institutional email addresses.*
- *Users access the system through a modern web browser.*
- *Internet connection is available for OTP email delivery.*
- *Admin functionalities are handled manually or through hardcoded credentials.*
- *Users are assumed to be honest; no CAPTCHA or anti-bot logic is yet in place.*

## 3.3 Significant Design Packages

- ***Frontend Package**: HTML, CSS, JavaScript (Bootstrap) for UI components.*
- ***Backend Package**: Flask routes, authentication, and logic.*
- ***Database Package**: SQLite schema for users, professors, ratings, and posts.*
- ***Utilities**: OTP generation, session handling, and email service.*

| Package/Module | Description |
|---|---|
| *main.py* | *Core application logic including routing, OTP generation/validation, data fetching, and page rendering.* |
| *create_db.py* | *Initializes the SQLite database and seeds initial data (professors, reviews).* |
| *templates/* | *HTML files rendered by Flask for various pages (index, login, dashboard, professor view, etc.).* |
| *professors.db* | *SQLite database storing professors and reviews.* |
| *session (Flask)* | *Temporarily stores OTPs and email information during the login process.* |

### 3.4 Dependent External Interfaces

*The table below lists the public interfaces this design depends on:*

| External Application and Interface Name | Module Using the Interface | Functionality/ Description |
|---|---|---|
| Email Service (SMTP) send_otp(email, otp) | main.py | Sends OTP to user's email for authentication |
| Web Browser HTML Template Rendering | main.py, templates/ | Displays the UI and sends input to the backend |
| SQLite professors.db | main.py , create_db.py | Used for storing professors and reviews |

### 3.5 Implemented Application External Interfaces (and SOA web services)

The table below lists the internal modules responsible for implementing public-facing routes and features:

| Interface Name | Module Implementing the Interface | Functionality/Description |
|---|---|---|

| | | |
|---|---|---|
| / | main.py | Loads homepage with navigation options |
| /register, /login | main.py | Accepts email ID and initiates OTP generation |
| /send_otp | main.py | Sends OTP to user and stores it in session |
| /verify_otp | main.py | Validates OTP and redirects to dashboard |
| /api/professors | main.py | Returns a list of professors and their ratings |
| /professor/<id> | main.py | Displays professor details and reviews |
| /rate_professor (assumed) | main.py | Accepts new ratings from users |

# 4 Logical View

*The Rate My Professor application uses a modular structure built around the Flask web framework. It follows the Model-View-Controller (MVC) pattern, where routes (controllers) handle logic and communication between the data model and HTML templates (views). Each module is responsible for completing specific user interactions such as OTP-based login, professor rating, and displaying reviews.*

## 4.1 Design Model

| Module | Responsibilities |
|---|---|
| main.py | Handles user interactions through HTTP routes (login, OTP, professor view, rating submission). |
| create_db.py | Initializes SQLite database with sample data (professors, reviews). |
| templates/ | Contains all HTML templates rendered dynamically by Flask. |
| professors.db | Stores data for professors and student reviews. |
| session (Flask) | Temporarily stores OTPs and login state per user session. |

Functions (Simulating Classes):

*Flask routes act like methods inside a controller. Some key functions include:*

- *home() → renders the homepage (/)*
- *register() / login() → initiates user session*

- *send_otp(email) → generates and sends OTP via email*
- *verify_otp() → validates entered OTP*
- *get_professors() → returns professor data as JSON*
- *professor_detail(id) → loads individual professor page with reviews*
- *rate_professor() → submits a new rating and updates average*

## 4.2 Use Case Realization

**Use Case: Register/Login with OTP**

***Goal**: Securely log users in using OTP verification.*

***Flow**:*

1. *User enters email on /register or /login.*
2. *System generates a 4/6-digit OTP and sends it via SMTP.*
3. *OTP is stored in session['otp'].*
4. *User inputs OTP at /verify_otp.*
5. *If OTP is valid → redirect to dashboard.*

***Modules involved**: main.py, session, email, templates/login.html, templates/otp.html*

**Use Case: View Professors**

***Goal**: Show all professors ordered by average rating.*

***Flow**:*

1. *User navigates to the homepage.*
2. */api/professors route queries the professors table.*
3. *Results are sent as JSON and rendered into Bootstrap cards on the frontend.*

***Modules involved**: main.py, professors.db, templates/index.html*

**Use Case: Rate a Professor**

***Goal**: Submit a new review and update average rating.*

***Flow**:*

1. *User clicks a professor card and lands on /professor/<id>.*
2. *Submits a form with rating (1-5) and comment.*
3. *Flask inserts review into reviews table.*
4. *Recalculates average rating in professors table.*
5. *Page refreshes with updated rating.*

***Modules involved**: main.py, professors.db, templates/professor.html*

# 5 Data View

*This section provides a detailed look at the system's persistent data. The Rate My Professor application uses SQLite as its backend database for storing information about professors and their ratings. OTPs are handled through session storage but can optionally be stored in a separate table for audit or logging purposes.*

## 5.1 Domain Model

*The domain model for the application includes two core entities:*

**Entities and Relationships:**

- ***Professor***
  - *Stores professor details like name, rating, and profile image.*
  - *Has a one-to-many relationship with reviews.*
- ***Review***
  - *Stores student ratings and feedback associated with professors.*
  - *Each review belongs to a single professor.*
- ***OTP_Log***
  - *If persisted, this table would log OTPs sent for verification.*
  - *Helps track whether users verified successfully.*

***Entity Relationship Summary:***

- *One **Professor** → Many **Reviews***
- *One **User Email** → Many **OTP_Log** records*

## 5.2 Data Model (persistent data view)

*The database consists of the following tables:*

*Table: professors*

| Column | Type | Description |
|---|---|---|
| id | INTEGER | Primary key, unique professor ID |
| name | TEXT | Full name of the professor |
| average_rating | REAL | Auto-updated average score |
| image_url | TEXT | URL for professor profile picture |

*Table: Reviews*

| Column | Type | Description |
|---|---|---|
| id | INTEGER | Primary key |
| professor_id | INTEGER | Foreign key referencing professors.id |
| rating | INTEGER | Rating score from student (1 to 5) |
| comment | TEXT | Feedback message |

*Table: otp_logs*

| Column | Type | Description |
|---|---|---|
| id | INTEGER | Primary key |
| email | TEXT | Email to which OTP was sent |
| otp | TEXT | OTP code generated |
| timestamp | DATETIME | When the OTP was created |
| is_verified | BOOLEAN | Whether OTP was successfully validated |

### 5.2.1 Data Dictionary

| Table | Attribute | Description |
|---|---|---|
| professors | id | Unique ID for each professor |

| | | |
|---|---|---|
| | name | Professor's full name |
| | average_rating | Average of all submitted ratings |
| | image_url | URL to profile image |
| reviews | id | Unique ID for each review |
| | professor_id | ID of the reviewed professor |
| | rating | Rating value (1–5) |
| | comment | User feedback on the professor |
| otp_logs | email | Email address used to send OTP |
| | otp | One-time password sent to user |
| | timestamp | Time when OTP was generated |
| | is_verified | Boolean indicating successful OTP verification |

# 6 Exception Handling

The *Rate My Professor* application includes basic exception handling to ensure stability and proper user feedback during errors. Exceptions are mostly handled within the Flask application code (`main.py`) and include database access issues, invalid OTP entries, and form input errors.

*Types of Exceptions and Handling Strategy:*

| *Exception* | *Cause* | *Handling Strategy* | *Logging/Feedback* | *Follow-Up Action* |
|---|---|---|---|---|
| *sqlite3.Operational Error* | *Database is inaccessible or malformed query* | *Wrapped in try-except blocks during DB operations* | *Error message flashed to user* | *Retry after fixing DB or query* |
| *KeyError* | *Missing keys in session (e.g., OTP expired)* | *Check session dictionary before access* | *Redirect to OTP entry page with error message* | *Prompt user to re-enter info* |
| *ValueError* | *Invalid data type in form submission (e.g., rating not an int)* | *Use input validation in route handlers* | *Show validation message on form* | *Re-enter correct input* |

| | | | | |
|---|---|---|---|---|
| *smtplib.SMTPException* | *Email server connection failed during OTP send* | *Catch and log error during send_otp* | *Display "Could not send OTP" alert* | *Retry or contact admin* |
| *Exception (Generic)* | *Any other unhandled error* | *Caught using global error handler or generic blocks* | *Show "Something went wrong" page* | *Log for admin review* |

# 7 Configurable Parameters

*Although Flask applications are typically simpler than enterprise platforms like J2EE, there are still key configuration parameters that can be adjusted to tailor system behavior, especially for development, deployment, and security.*

*The following table lists the configurable parameters used in the system, their purpose, and whether they can be changed dynamically (without restarting the application):*

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

| Configuration Parameter Name | Definition and Usage | Dynamic? |
|---|---|---|
| SECRET_KEY | Used by Flask to manage session security, CSRF protection, and OTP session variables. | No |
| OTP_LENGTH | Number of digits in the generated OTP (e.g., 4 or 6). | Yes |
| SMTP_SERVER | Host address for the email service (e.g., smtp.gmail.com). | No |
| SMTP_PORT | Port number used to send OTP emails (typically 587 for TLS). | No |
| SENDER_EMAIL | Email address from which OTPs are sent. | No |
| DEBUG | Enables Flask debug mode (useful in development only). | Yes |
| DATABASE_PATH | Path to the SQLite database file. | No |
| MAX_RATING_VALUE | Maximum value allowed for a rating submission (default is 5). | Yes |

| | | |
|---|---|---|

# 8 Quality of Service

*This section outlines how the system meets quality expectations related to availability, security, performance, and operational monitoring. Given that the system is lightweight and built using Flask and SQLite, it is best suited for small to medium-scale academic deployments.*

## 8.1 Availability

The system is designed for high availability in a classroom or campus environment:

- **Availability Requirement**: The application should be accessible to users at all times during academic hours, with minimal downtime.
- **Design Features Supporting Availability**:
  - Lightweight Flask server ensures fast startup and quick recovery in case of failure.
  - SQLite provides persistent storage without needing a separate database server.
  - No background cron jobs or heavy batch processing that may block access.
- **Potential Impact Factors**:
  - Maintenance tasks like database resets (if done manually).
  - Email service outages affecting OTP delivery.

Downtime can be minimized by restarting the Flask server quickly or hosting it using platforms like Heroku or PythonAnywhere with auto-restart features.

## 8.2 Security and Authorization

The system implements basic security measures suitable for academic projects:

- **Authorization Features**:
  - Students log in via **email-based OTP**—no passwords are stored, reducing credential theft risk.
  - Session management via Flask ensures that unauthorized users cannot access dashboard pages.
  - Admin access is protected via hardcoded or separate credentials (future upgrade may include role-based access control).
- **Data Access Security**:

- OTPs are stored only temporarily in session.
- Database inputs (like comments and ratings) are validated to prevent injection.
- Future upgrades may include CAPTCHA to prevent spam submissions and additional authorization layers for admin routes.

## 8.3 Load and Performance Implications

Given the academic use case and limited concurrent users, the application is not expected to face high load:

- **Transaction Rate**: Expected to handle tens of requests per minute, which SQLite and Flask can easily support.
- **Performance Considerations**:
  - Data volume is low (a few hundred reviews and professors at most).
  - Use of AJAX for professor data (/api/professors) ensures efficient frontend updates.
  - Email OTP generation may experience delays if SMTP server is down or rate-limited.

**Further Scaling Considerations:**:

- For higher usage, migrate from SQLite to PostgreSQL or MySQL.
- Deploy using a WSGI server (e.g., Gunicorn) and reverse proxy (e.g., Nginx).

## 8.4 Monitoring and Control

*Basic monitoring can be implemented using Flask and external tools:*

- ***Controllable Processes**:*
  - *OTP generation and verification logic in main.py*
  - *Review submission and database updates*
- ***Measurable Values (for future expansion)**:*
  - *Number of OTP requests per session*
  - *Number of ratings submitted per day*
  - *System uptime logs (if hosted on platforms with dashboards)*

*For production, error logging using Python's logging module and health monitoring using uptime monitors (e.g., UptimeRobot) can be introduced.*