Software Design Specifications

for

“ScoreLens” - A Visualisation Tool for Student Scores

Prepared by:

Bhavana G, se22ucse052
Ramya Kanneganti, SE22UCSE218
Sriya Surisetty, SE22UCSE261
Stuti Garg, SE22UCSE263
Sravya Tadeparti, SE22UCSE255
Vahini Ramadhenu, SE22UCSE281

Software Engineering Course

Document Information

Title: Software Design Specifications	
Project Manager: Swapna Ma'am	**Document Version No: 1**
	Document Version Date:April 9, 2025
Prepared By: Team 13	**Preparation Date: April 7, 2025**

Version History

Ver. No.	Ver. Date	Revised By	Description	Filename
1	April 9, 2025	-	Use case view, design view, logical view, exception handling of ScoreLens – A Visualization Tool for Students	Software Design Specification of Team 13

Table of Contents

1 Introduction

1.1 Purpose

This Software Design Specification describes the architecture and design of the Elective Course Registration System (ECRS), a web-based platform that allows students to register for available electives and project-based courses based on eligibility and availability.

1.2 Scope

This document focuses on the software design of **ScoreLens**, which is a web application for visualizing student academic performance.

The scope of this document includes:

-The **user interface design** (upload forms, dashboards)

-The **backend logic** (data validation, API endpoints, file processing)

-The **database design** (schema and data relationships)

-The **integration of visualization tools** such as Chart.js or Power BI

This SDS will directly affect:

-Development decisions, by guiding how features are built and connected

-Testing procedures, by highlighting what parts of the system need verification

-Project planning, by outlining the components and estimating their complexity

-Future updates, by serving as a reference for how the current system works

It does **not** include design for a mobile version, multi-user collaboration, or direct student access — these are outside the current project scope.

1.3 Definitions, Acronyms, and Abbreviations

Term/Acronym	Definition
SDS	Software Design Specification
UI	User Interface
API	Application Programming Interface
DB	Database
Power BI	Business Analytics Service by Microsoft

JWT JSON Web Token (for user authentication)

1.4 References

Flask Documentation: https://flask.palletsprojects.com/

SQLAlchemy: https://docs.sqlalchemy.org/en/20/

JWT: https://jwt.io/introduction

2 Use Case View

This use case allows students and professors to view academic performance using graphical visualizations such as bar charts, line graphs, or pie charts. It enhances user understanding of academic trends, comparisons across subjects, and class-level statistics.
The goal is to present data in an intuitive and user-friendly format to support academic insights.

2.1 User Authentication

The user logs into the system using their credentials. Access is role-based to ensure the correct permissions are applied.

2.2 Access Visualization Module

From the dashboard, the user selects the "Visualize Scores" option. This option is prominently placed for quick and easy access.

2.3 Select Data Scope

The user chooses filters such as subject, semester, exam type, or student group.
Filter flexibility allows users to tailor the data view to their needs.

2.4 View Generated Chart

The system processes the selected data and displays a visualization (e.g., bar chart, line graph).
The chart dynamically updates based on the user's selected filters.

2.5 Interact with Visualization

Users can hover, click, or toggle to view detailed information or switch chart types.
Interactive features help users explore trends and pinpoint specific data.

3 Design Overview

The Student Score Visualization website is designed to streamline the process of uploading, storing, and analyzing student marks for different subjects. The system follows a modular, layered architecture comprising three primary layers: the user interface layer, the processing layer (backend), and the data storage layer.

This design complies with the functional and non-functional requirements outlined in the Software Requirements Specification (SRS) document. It enables teachers to upload Excel files containing student names and marks, which are then stored in a structured SQLite database. Each subject corresponds to a separate table, and new entries are appended to ensure historical data retention.

The system adheres to interface contracts by ensuring clear separation of concerns between data input, processing logic, and visualization output. The frontend is developed using HTML and JavaScript to allow dynamic interaction and visualization of scores, while the backend handles data parsing, database operations, and feedback management.

Modules are decomposed based on their core responsibilities:

- **Frontend Module:** Manages the user interface and interactions.
- **Backend Module:** Handles file processing, data validation, and storage logic.
- **Database Module:** Manages subject-wise tables and student records in SQLite.

This modular design enhances maintainability, scalability, and allows future integrations such as user authentication or cloud storage support.

3.1 Design Goals and Constraints

The primary goal of the Student Score Visualization website is to provide a simple, efficient, and user-friendly platform for managing and visualizing student performance data. The following objectives and constraints significantly influence the design:

Design Goals

- **Simplicity and Usability**: Ensure that teachers and users with minimal technical background can easily upload Excel files and view visualizations of student performance.
- **Modularity**: Separate frontend, backend, and database responsibilities to enhance maintainability and future scalability.
- **Data Integrity**: Preserve existing records while allowing new data to be appended, ensuring consistent historical tracking.
- **Visualization and Feedback**: Present data in a visually meaningful way (charts, graphs, etc.) and enable feedback collection through a simple portal.
- **Responsiveness**: Ensure the UI works well across devices of various screen sizes.

Constraints

- **Technology Stack**:
  - **Frontend**: HTML, JavaScript
  - **Backend**: Python (if confirmed), with integration for Excel file processing
  - **Database**: SQLite (due to its lightweight, serverless design)
- **Development Tools**: The project uses lightweight tools to keep setup simple and minimize dependencies, suitable for a student-level environment.
- **Schedule Constraint**: The project is developed within a limited academic timeline, requiring iterative, focused development cycles.
- **Team Size**: The backend development is handled solely by one member, which influences the project scope and complexity.
- **File Format Constraint**: Only Excel files (.xlsx or .xls) in a fixed structure (Name, Marks) are accepted for upload and processing.
- **No Legacy Code**: The system is developed from scratch, with no dependencies on previous codebases.

3.2 Design Assumptions

The following assumptions have been made during the design of the Student Score Visualization website. These assumptions significantly influence the structure and functionality of the system:

1. **Fixed Excel Structure**: All uploaded Excel files follow a consistent format with two columns: "Name" and "Marks". No header mismatch or structural variation is expected.
2. **Single Subject per File**: Each Excel file corresponds to one subject only and is handled independently. Teachers will upload one file per subject.
3. **Unique Student Names**: Student names are assumed to be unique identifiers within each subject, with no duplicate entries for the same name in a single file.
4. **Limited Concurrent Usage**: The system is assumed to be used by a small number of users simultaneously, so performance optimizations for large-scale concurrency are not prioritized.
5. **Local/Offline Usage**: The application is expected to run locally or within a closed academic network; hence, cloud support and real-time collaboration features are out of scope.

6. **No Login System**: There is currently no user authentication or role-based access, as the scope is focused on core functionality (upload, store, visualize, feedback).
7. **Browser Compatibility**: Users are expected to access the system on modern browsers that support JavaScript and standard HTML5 features.

3.3 Significant Design Packages

The system follows a layered architecture and is logically divided into several design packages to ensure separation of concerns, maintainability, and scalability. Each package (or module) handles a distinct responsibility within the application.

1. User Interface Package (UI Layer)

- **Description**: This package handles all visual elements and interactions.
- **Technologies**: HTML, CSS, JavaScript
- **Responsibilities**:
  - Display forms for file upload and feedback input.
  - Render visualizations (charts/graphs) using student score data.
  - Provide alerts/messages for upload status and errors.

2. Data Processing & Backend Package (Logic Layer)

- **Description**: Manages the logic for handling file uploads, data validation, and backend communication.
- **Technologies**: Python (with libraries like `pandas`, `openpyxl`)
- **Responsibilities**:
  - Parse Excel files and extract student data.
  - Validate data before inserting it into the database.
  - Handle form submissions for feedback.

3. Database Management Package (Data Layer)

- **Description**: Handles all interactions with the SQLite database.
- **Technologies**: SQLite
- **Responsibilities**:
  - Store student scores in subject-specific tables.
  - Append new data while preserving existing records.
  - Retrieve data for visualizations and reporting.

Package Hierarchy & Dependencies

- The **UI Package** interacts directly with the **Backend Package** via form submissions and JavaScript calls (e.g., AJAX or form POST).
- The **Backend Package** serves as the intermediary, receiving data from the UI and processing it before interacting with the **Database Package**.
- There are no direct interactions between the UI and Database layers, maintaining proper abstraction.

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
Excel File Parser (e.g., `pandas`, `openpyxl`)	Backend / File Processing Module	Used to read and extract student data (Name and Marks) from uploaded Excel files. It ensures correct parsing and prepares the data for database insertion.

SQLite (Database Interface) JavaScript DOM Interface	Database Management Module Frontend / UI Module	Used to insert, update, and retrieve student scores. This interface supports subject-wise table handling and appending data without overwriting existing records. Used to update UI elements dynamically based on user actions (like file upload, form submission, and data visualization) and to enhance interactivity.

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**
File Upload Endpoint	Backend / File Processing Module	Accepts Excel files from the frontend, processes them using pandas or openpyxl, and stores the data in subject-specific tables in the SQLite database.
Feedback Form Submission Handler	Backend / Feedback Handling Module	Receives feedback input from users and stores it in the database or local storage for later analysis or admin access.
Data Retrieval API (for Charts)	Backend / Data Access Module	Provides score data to the frontend for rendering visualizations like charts or tables. Returns data in JSON format when called via JavaScript.

4 Logical View

4.1 Design Model

The system is designed using a layered architecture with three primary layers:

- **Presentation Layer**: Web interface for user interaction (students, professors, admin).
- **Application Layer**: Core logic and workflows (authentication, grade processing, analytics).
- **Data Layer**: Handles storage of users, grades, requests, and class information.

Each of these layers is further divided into modules that handle specific tasks.

Modules Overview

- **User Authentication Module:** Manages login/signup, roles (student, professor, admin), and session handling.
- **Grade Management Module:** Allows professors to upload grades and students to view their performance.
- **Visualization Module:** Generates interactive charts and graphs based on grades and performance data.
- **Admin Module:** Lets professors and admins manage class memberships and grade updates.
- **Notification Module:** Handles communication, including alerts for grade changes and request statuses.

Design Summary:

- Each module contains relevant classes like User, Grade, Class, and Chart that interact to provide specific functionality.
- Class relationships follow a modular approach to maintain flexibility and scalability.
- A class diagram (optional) would show how main entities like User, Grade, and Chart interact in the system.

4.2 Use Case Realization

Each key use case from Section 2 is implemented by coordinating interactions between the frontend, backend modules, and database.

Use Case 1: Student Views Grades and Statistics

- The student logs in and navigates to the dashboard.
- The system retrieves their grade data and class statistics from the backend.
- The frontend displays this data using interactive charts for better understanding.

Use Case 2: Professor Uploads Grades and Views Trends

- The professor logs in and uploads a grade sheet.
- The system processes and stores the data securely.
- Class averages and performance trends are calculated and visualized on the dashboard.

Use Case 3: Student Requests Grade Change

- A student submits a grade change request.
- The request is forwarded to the respective professor/admin.
- After review, the professor can approve or reject the request.
- The system updates the grade if needed and refreshes the statistics.

Each use case involves communication between modules like authentication, grade management, and visualization, ensuring a seamless user experience.

Each key use case from Section 2 is implemented by coordinating interactions between the frontend, backend modules, and database.

Use Case 1: Student Views Grades and Statistics

- The student logs in and navigates to the dashboard.
- The system retrieves their grade data and class statistics from the backend.
- The frontend displays this data using interactive charts for better understanding.

Use Case 2: Professor Uploads Grades and Views Trends

- The professor logs in and uploads a grade sheet.
- The system processes and stores the data securely.
- Class averages and performance trends are calculated and visualized on the dashboard.

Use Case 3: Student Requests Grade Change

- A student submits a grade change request.
- The request is forwarded to the respective professor/admin.
- After review, the professor can approve or reject the request.
- The system updates the grade if needed and refreshes the statistics.

Each use case involves communication between modules like authentication, grade management, and visualization, ensuring a seamless user experience.

5 Data View

5.1 Domain Model

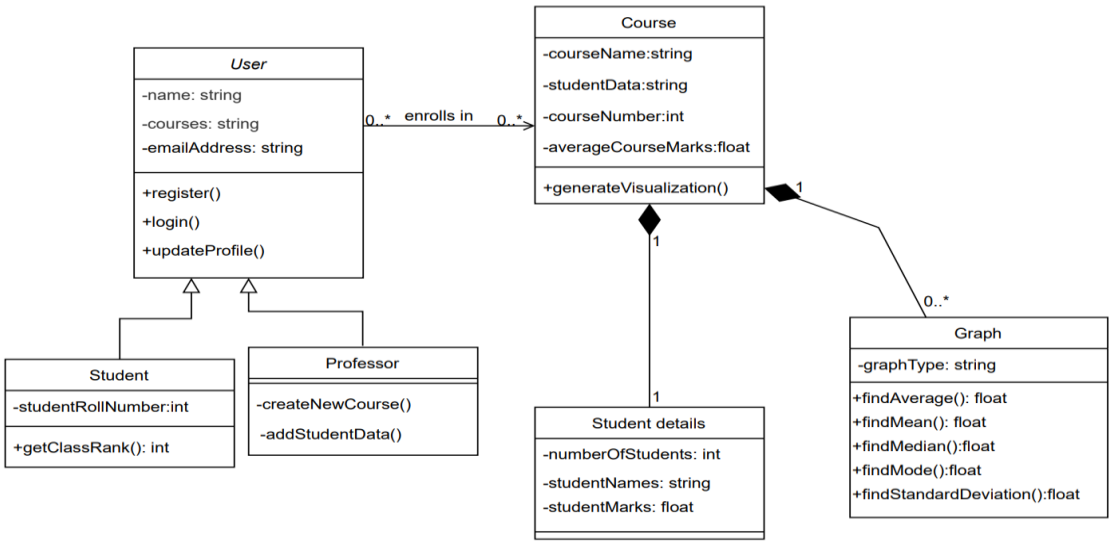

Figure: Domain Model (ERD) — This diagram illustrates the main entities of the system, their attributes, behaviors, and the relationships between them. It helps map the domain logic to data structures and database design.

The domain model represents the core entities of the system and the relationships between them. It provides a conceptual framework for how the system's objects interact, focusing on persistent data relevant to the ScoreLens platform.

The primary entities in the domain are:

- **User**: Represents any individual using the platform, either as a professor or a student. Each user has a name, email address, and associated courses. Users can register, log in, and update their profiles.
- **Student**: Inherits from `User`. Students are associated with a roll number and have the ability to view their class rank based on marks data.
- **Professor**: Inherits from `User`. Professors can create new courses and upload student score data.
- **Course**: Represents a course created by a professor. Each course contains associated student data, a course number, and an average mark. It can generate visualizations based on student performance.
- **StudentDetails**: Contains student-specific information for a course, including student names, marks, and the number of students.
- **Graph**: Represents different types of visualizations (bar, line, scatter, etc.) that can be generated based on student performance data. Each graph can calculate average, median, mode, mean, and standard deviation.

The relationships between these entities are shown in the diagram below. Users can enroll in multiple courses. Each course has student details and can generate multiple graphs for performance analysis.

5.2 Data Model (persistent data view)

The persistent data of the ScoreLens system is stored using **SQLite**, a lightweight relational database. This section outlines the structure of the database schema, representing how domain entities are stored in tables along with their relationships.

The key database tables are:

Table Name	Attributes	Description

Users	id (PK), name, email, role, courses	Stores user info (both students and professors). The role field distinguishes between them.
Students	user_id (PK, FK), roll_number	Stores student-specific data linked to a user.
Professors	user_id (PK, FK)	Stores professor-specific data linked to a user.
Courses	id (PK), name, course_number, professor_id (FK), average_marks	Represents courses created by professors.
Enrollments	user_id (FK), course_id (FK)	Many-to-many mapping between users and courses.
StudentDetails	id (PK), course_id (FK), student_name, student_marks	Stores marks and names for each student in a course.
Graphs	id (PK), course_id (FK), graph_type	Stores the type of visualization generated for a course.

5.2.1 Data Dictionary

Users Table

Attribute	Data Type	Description	Constraints
id	INTEGER	Unique ID for the user	Primary Key
name	TEXT	Name of the user	Not Null
email	TEXT	Email address	Unique, Not Null
role	TEXT	Role of user (student/professor)	Not Null
courses	TEXT / JSON	List of associated course IDs (if any)	Optional

Students Table

Attribute	Data Type	Description	Constraints
user_id	INTEGER	Linked ID from Users table	Primary Key, Foreign Key (Users.id)
roll_number	TEXT	Unique roll number of student	Not Null, Unique

Professors Table

Attribute	Data Type	Description	Constraints
user_id	INTEGER	Linked ID from Users table	Primary Key, Foreign Key (Users.id)

Courses Table

Attribute	Data Type	Description	Constraints
id	INTEGER	Unique course ID	Primary Key

name	TEXT	Course name	Not Null
course_number	TEXT	Code for the course (e.g., CS101)	Unique
professor_id	INTEGER	Linked to professor creating the course	Foreign Key (Professors.user_id)
average_marks	REAL	Average marks across enrolled students	Optional

Enrollments Table

Attribute	Data Type	Description	Constraints
user_id	INTEGER	User enrolled (student or professor)	Foreign Key (Users.id)
course_id	INTEGER	Course enrolled in	Foreign Key (Courses.id)
			Composite Primary Key (user_id, course_id)

StudentDetails Table

Attribute	Data Type	Description	Constraints
id	INTEGER	Unique record ID	Primary Key
course_id	INTEGER	Associated course	Foreign Key (Courses.id)
student_name	TEXT	Name of the student	Not Null
student_marks	REAL	Marks scored by the student	Not Null

Graphs Table

Attribute	Data Type	Description	Constraints
id	INTEGER	Unique graph ID	Primary Key
course_id	INTEGER	Associated course	Foreign Key (Courses.id)
graph_type	TEXT	Type of graph (e.g., bar, line, etc.)	Not Null

6 Exception Handling

This section describes the exceptions that may occur within the application, the conditions under which they arise, how they are handled and logged, and any necessary follow-up actions. Proper exception handling ensures the system remains robust, user-friendly, and secure during unexpected events.

6.1 Authentication Failure

Circumstance: User enters invalid credentials during login.
Handling: The system displays an error message indicating incorrect username or password.
Logging: Failed login attempts are recorded with a timestamp and IP address.
Follow-up Action: User is prompted to retry or reset their password after a limited number of failed attempts.

6.2 Data Retrieval Error

Circumstance: Failure to fetch student scores due to database issues or network problems.
Handling: An error message is shown indicating that data could not be loaded.
Logging: Error details, including query failure or timeout messages, are logged.
Follow-up Action: System retries once or suggests the user to refresh the page.

6.3 Visualization Rendering Failure

Circumstance: Chart rendering fails due to invalid or missing data.
Handling: The system notifies the user with a message like "Unable to generate chart. Please check selected filters."
Logging: The exception is logged with the dataset and visualization parameters used.
Follow-up Action: User is advised to modify filters or contact support if the issue persists.

6.4 Access Denied

Circumstance: A student tries to access admin-only features or unauthorized data.
Handling: The system redirects to an error page with a “Permission Denied” message.
Logging: The unauthorized access attempt is logged with user details.
Follow-up Action: Admin may be notified if repeated violations are detected.

6.5 File Export Error

Circumstance: Error occurs while downloading chart or report as PDF/image.
Handling: User receives a notification that the export failed.
Logging: Logs include the export format, time of attempt, and user role.
Follow-up Action: User is advised to try again or contact support.

7 Configurable Parameters

This section describes the parameters used by the application that are configurable. These parameters allow the system behavior to be adjusted without changing the core application logic. Some parameters are dynamic and can be modified without restarting the application, while others require a restart.

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

Configuration Parameter Name	Definition and Usage	Dynamic?
maxLoginAttempts	Maximum number of failed login attempts before temporary lockout.	Yes
chartDefaultType	Default chart type used for score visualization (e.g., bar, line).	Yes

sessionTimeoutMinutes	Duration (in minutes) before an inactive user is logged out.	No
dataRefreshInterval	Interval (in seconds) for auto-refresh of score data.	Yes
databaseRetryCount	Number of retry attempts for failed database queries.	No

8 Quality of Service

This section describes aspects of the design related to application availability, security, performance, and monitoring and control in production.

8.1 Availability

ScoreLens is designed for high availability, aligning with the business requirement of 99.9% uptime. By default, Create React App supports hot-reloading and development-time stability. For production, it is developed and maintained using GitHub for version control and collaboration. It can be deployed by serving the compiled files through a local based web server. The frontend is built with React and the backend connects to a database, with JSON used for data exchange.

To reduce downtime, we will schedule maintenance windows that will be minimal and pre-communicated. Also, data loading and mass updates will be handled via background scripts or in off-peak hours. Housekeeping tasks are isolated from user-interface to maintain responsiveness.

8.2 Security and Authorization

Our website, prioritizes data privacy and secure access to features:

- Role-based access control is in place to differentiate admin users from general users.
- Sensitive user data (like scores, personal information- student/faculty details) is secured via HTTPS protocols and token-based authentication.
- Only authenticated users can create or edit content, unauthorized access is strictly restricted and monitored.

User management:

- Admin dashboard includes controls for enabling or disabling accounts.
- Password recovery and user verification is to be implemented.
- OAuth and biometrics integration can be considered for future scalability

8.3 Load and Performance Implications

ScoreLens could experience usage spikes during exam result releases or batch uploads:

- Load testing is simulated using tools like Postman for API endpoints.
- React's lazy loading and code-splitting techniques are employed to minimize the initial bundle size and improve loading times
- Expected traffic is approximately 100 concurrent users during peak, with rates up to 10 requests/sec.
- Data growth is manageable under current architecture, but horizontal scaling options are thought for future demand.

8.4 Monitoring and Control

Our portal includes internal logging and monitoring:

- Use of third-party tools like Sentry (if integrated) we will capture front-end exceptions.
- Basic health APIs can be set up if backend is deployed.
- Static log files and performance metrics (render times, user load) could be collected using Google Analytics in future.
- CI/CD pipelines (for example- GitHub Actions) help validate deployment health before going live.