Software Requirements Specification

for

isualisation of Student Scores-ScoreLens

Version <1.1>

Prepared by

Group Name: ScoreLens

Sravya Tadeparti	SE22UCSE255	se22ucse255@mahindrauniversity.edu.in
Vahini Ramadhenu	SE22UCSE281	se22ucse281@mahindrauniversity.edu.in
Ramya Kanneganti	SE22UCSE218	se22ucse218@mahindrauniversity.edu.in
Bhavana Gundeti	SE22UCSE052	se22ucse052@mahindrauniversity.edu.in
Stuti Garg	SE22UCSE263	Se22ucse263@mahindrauniversity.edu.in
		se22ucse261@mahindrauniversity.edu.in
Sriya Surisetty	SE22UCSE261	

Instructor:	*Swapna S*
Course:	Software Engineering
Lab Section:	*CSE (Monday 10:30 AM – 12:30 PM)*
Teaching Assistant:	*Swapna Ma'am*

Date: **10-03-25**

Contents

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.1	Sravya Tadeparti, Bhavana Gundeti, Vahini Ramadhenu, Ramya Kanneganti, Stuti Garg, Sriya	This is the initial version containing the basic functionality such as logging in as a professor and student, storing data, and generating graphs.	10/03/25

1 Introduction

1.1 Document Purpose

This Software Requirements Specification (SRS) document specifies the goals, functionality, constraints, and design considerations for ScoreLens (Version 1.0), a web-based application that allows professors and students to examine academic performance using interactive visualizations. Professors enter student scores, and the system provides insights like average scores, distribution of performance, and rankings. This enables professors to track class performance, detect trends, and make informed decisions to enhance learning outcomes.

This document plays several roles: it gives a clear vision of the system to all stakeholders, is a reference for developers and testers, specifies system constraints and dependencies, and ensures consistency throughout the software development process. It also forms the foundation for validation and verification, ensuring that the end product is as expected by the users and meets functional requirements.

1.2 Product Scope

ScoreLens is a web app designed to facilitate professors and students in analyzing academic performance using interactive graphs and insights. Professors provide the scores of their students, and the web app produces visualizations like average scores, distribution of performance, and rankings of students. This enables professors to monitor class performance, observe trends, and make data-driven decisions to improve learning outcomes.

Students are able to view their scores and compare them with class averages, having a better idea of their academic performance. By translating raw numerical information into simple-to-understand graphs, the platform makes performance analysis easier, more transparent, and more beneficial to both professors and students.

1.3 Intended Audience and Document Overview

Intended Audience
This report is meant for the following stakeholders:

Professor (Reviewer of this Document) – The professor who is guiding this project will utilize this SRS to review the clarity, completeness, and feasibility of the suggested system. The report will enable them to know the product's goals, requirements, and desired results.

Clients (Educational Institutions or Administrators) – Institutions that are likely to implement the system can utilize this report to learn about its features and advantages.

Professors (End Users Giving Input Data) – As direct users, professors can refer to this document to understand how they will use the system to enter student scores and create visualizations.

Developers – This document gives the specifications required to design, develop, and deploy the system.

Testers – The functional and non-functional requirements listed here will be used by testers to verify the behavior of the system and confirm its correctness.

Project Managers – This document assists in development timelines management, progress tracking, and project goal alignment.

Document Overview
This SRS is organized into the following sections:

Introduction – Presents an overview of the system, its purpose, scope, and major stakeholders. (Recommended for all readers, particularly the professor who will be reviewing the document.)

Overall Description – Presents the high-level functionality, constraints, and dependencies of the system. (Beneficial for developers, testers, and project managers.)

Specific Requirements – Specifies the functional and external interface requirements of the system in detail. (Critical for developers and testers.)

Non-Functional Requirements – Addresses performance, security, and software quality characteristics. (Critical for developers, testers, and project managers.)

Other Requirements – Any other constraints or considerations. (Can be referred to as necessary.)
Suggested Reading Sequence

The document reviewer, being a professor, must begin with the Overall Description and Introduction to comprehend the product vision first, then proceed to Specific Requirements to complete it.
Developers must concentrate on Specific Requirements and Non-Functional Requirements in order to direct system design and implementation.
Testers must consult Specific Requirements and Non-Functional Requirements to develop test cases.
Project Managers must read Overall Description and Non-Functional Requirements to monitor feasibility and constraints.

1.4 Definitions, Acronyms and Abbreviations

The following abbreviations and acronyms are used throughout this document:

- **API** – Application Programming Interface
- **CSV** – Comma-Separated Values (a file format used for importing/exporting data)
- **HTTP** – Hypertext Transfer Protocol
- **SRS** – Software Requirements Specification
- **UI** – User Interface
- **UX** – User Experience
- **Visualization** – Graphical representation of data (e.g., charts, graphs)
- **Web App** – Web Application

1.5 Document Conventions

This document adheres to the IEEE standards for Software Requirements Specifications. The following conventions have been employed to maintain consistency and readability:

Formatting Conventions

The document employs Arial font, size 11 for all text.
Section and subsection headings adhere to the standard IEEE template format.
Italics are employed for comments or further explanatory notes.
The text is single-spaced with 1-inch margins on all sides.

Naming Conventions

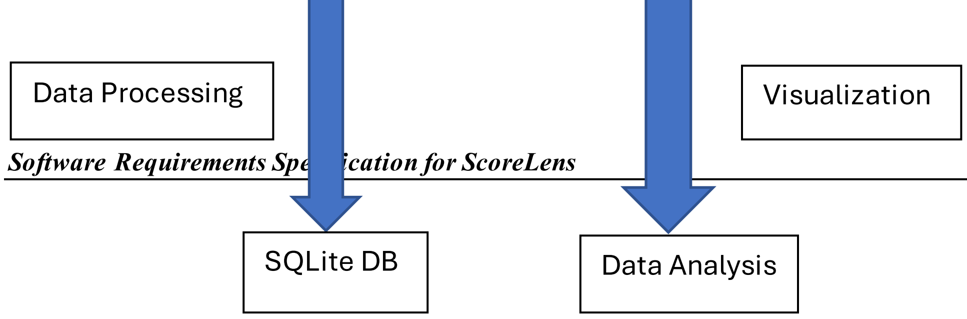
System modules and components are in PascalCase.
Function and variable names in code samples are camelCase.
Acronyms and abbreviations are in uppercase (e.g., API, UI, CSV).
These rules provide clarity and keep a professional, organized format throughout the document.

2 Overall Description

2.1 Product Overview

The system is a **new, self-contained web application** designed to provide an efficient and user-friendly platform for student score analytics. It integrates a **React.js frontend, Node.js backend, and SQL Lite database**, ensuring smooth interaction between users and the system. The application will support **role-based access**, secure authentication, and real-time data updates to enhance usability and efficiency.

It will interact with external APIs for [example: data retrieval or authentication] and ensure a smooth communication between different system components. The diagram below provides a high-level view of how users, the frontend, backend, and database interact.



2.2 Product Functionality

Key Features:

1. **User authentication** (Professors & Students)
2. **Uploading student grades** by professors
3. **Interactive data visualization** with charts & graphs
4. **Class comparison metrics** (average score, top/bottom performers)
5. **Request grade modifications** (students can request changes, professors approve/reject)
6. **Secure access** with role-based permissions
7. **Admin features** (manage users, classes, and grades)

2.3 Design and Implementation Constraints

The development of **ScoreLens** will have some limitations and rules that guide how it is built. These constraints will help make sure the system is **efficient, secure, and easy to use**, while following good software design practices.

The system will follow the **COMET** method for structured software design, ensuring modularity and maintainability. **UML diagrams** (class, sequence, and use case) will be used to model system interactions before implementation.

Technology Stack & Integration

- **Frontend:** React.js for an interactive UI
- **Backend:** Node.js with Express.js for handling requests
- **Database:** MongoDB for secure data storage
- **API Communication:** RESTful APIs for frontend-backend interaction

- **Authentication:** OAuth 2.0 or JWT for secure login

Performance & Security

- Optimized for **modern web browsers** and responsive across devices
- Supports **multiple concurrent users** with efficient database queries
- Implements **data encryption and secure API communication**

These constraints ensure the system remains **scalable, secure, and easy to maintain** while meeting project requirements.

By following these rules, **ScoreLens** will be a **secure, efficient, and user-friendly** tool for **visualizing student performance** while being easy to scale and maintain.

2.4 Assumptions and Dependencies

Major Assumptions:

- Professors and students **will have internet access** to use the platform.
- The **university will provide necessary datasets** (student scores).
- Users **will have basic tech literacy** to navigate the platform.
- **Third-party tools like PowerBI** will be available for integration.
- Hosting and database services **will remain stable and accessible**.

Dependencies:

- ScoreLens depends on **PowerBI** for visualization.
- Uses **third-party authentication libraries** for login/signup.
- Requires **a stable database backend** (e.g., PostgreSQL, Firebase).
- Relies on **React/Node.js (or chosen tech stack)** for development.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

Flowchart of ScoreLens:

ScoreLens provides an interactive and intuitive interface for users to explore and analyze data visually.
-Landing page: Contains Features page, Analytics page, About Us page, Contact page and Login page
-Main page: Contains student data, visual display of scores for students, and an excel template and an upload option for professors.

*Users will interact with the interface through **mouse clicks** and **keyboard inputs**.*
Dropdown menus, buttons, and sliders allow for easy selection and customization.

3.1.2 Hardware Interfaces

-Supported device types: Laptops, desktops, tablets, and mobile devices that access the visualization tool via a web browser.
-Servers: The backend database and API services running on a cloud server
-External storage: If users upload Excel files, the system interacts with local or cloud-based storage for temporary file processing.

3.1.3 Software Interfaces

-Connection with database: the backend stores student grade data in an SQLite database and the data is retrieved via Flask API endpoints and formatted in JSON

-Frontend: the frontend (HTML + JavaScript) makes AJAX calls to fetch data and receives it through Flask API endpoints

3.2 Functional Requirements

F1: Data upload and processing

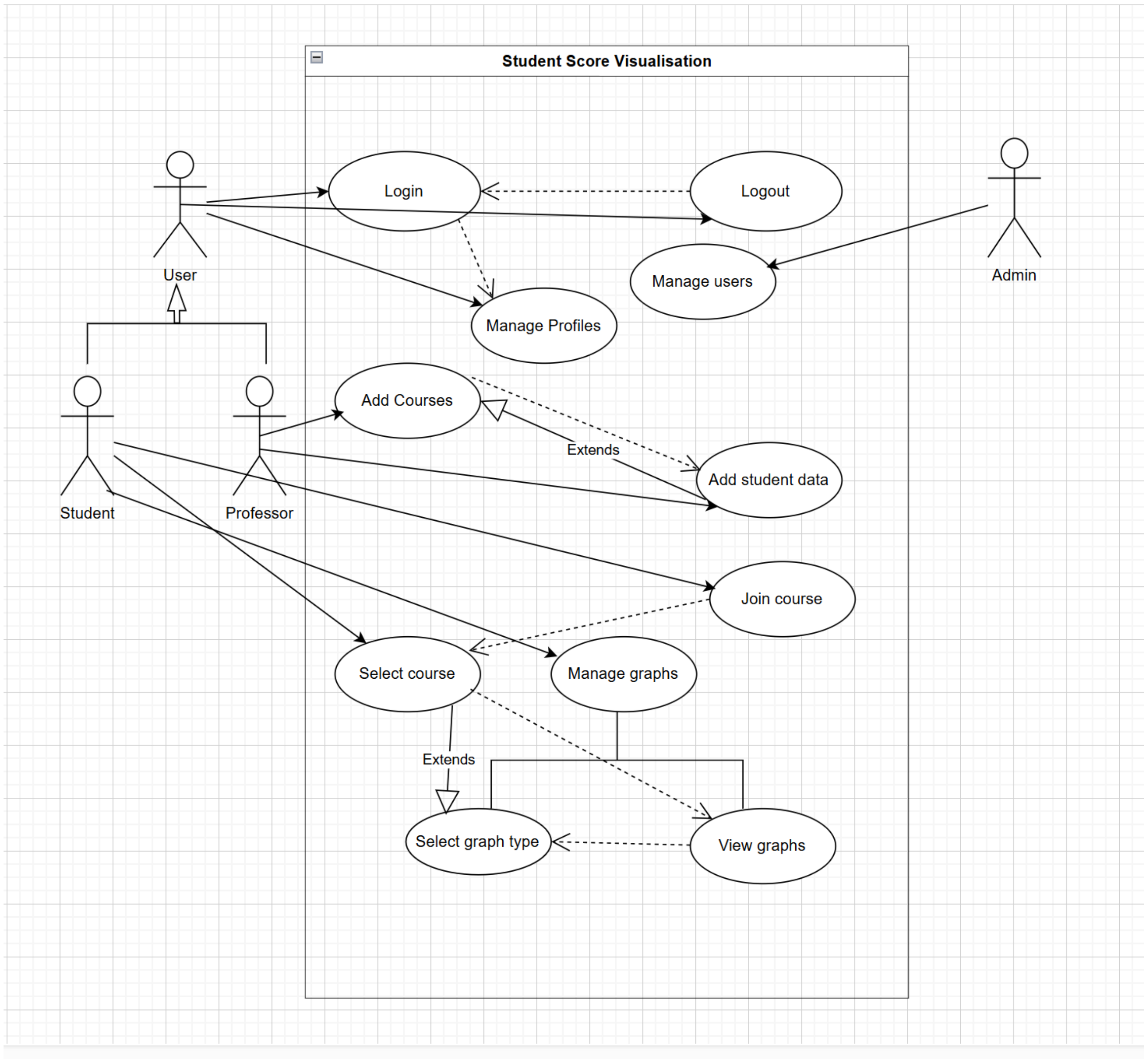
The system shall allow users to upload datasets in Excel format and validate the uploaded data to ensure it follows the expected format. It stores the uploaded data securely in a database.

F2: Data Visualization

This system shall generate visual representations of data, including bar charts, line graphs, pie charts, and scatter plots. It allows users to select different visualization types for the same dataset.

F4: Authentication
This system requires users to login before accessing the dashboard.
It supports different user roles (e.g., student, instructor, admin)
It securely stores user credentials and use encrypted authentication.

3.3 Use Case Model



3.3.1 Use Case #1 (use case name and unique identifier – e.g. U1)

Author – Sravya Tadeparti (se22ucse255), Vahini Ramadhenu (se22ucse281)

Purpose – To provide a visual aid in understanding the working of the website

Requirements Traceability – This use case traces to: User authentication, Profile and user management, course management, data visualization
Priority – High, as it involves essential functionalities like login, course selection, and graph visualization. The system would be incomplete without it.

Preconditions – The user must have an account and valid credentials to login, the professor must have access to add courses, and students must be enrolled in a course to view scores.

Post conditions – The user successfully logs in or out, students can view their scores, professors can upload scores, graphs are generated based on selected courses.

Actors – User initiates login, student selects course and views scores, professor selects a course and uploads scores, admin manages users and profiles

Extends – Add courses extends to Add student data and manage graphs extends to select graph type and view graphs

Flow of Events

1. Basic Flow – User logs in as either student or professor, selects a course and either views or uploads scores respectively. The user logs out when finished
2. Alternative Flow – invalid login credentials leads to error message displayed, student not enrolled in any courses is prompted to enroll before viewing scores
3. Exceptions – System downtime, unauthorized access, data upload failure

Notes/Issues - Improve security system to prevent unauthorized access

4 Other Non-functional Requirements

4.1 Performance Requirements

P1. The system will load and display student score visualizations within **1 second** for individual student queries and **3 seconds** for class-wide analytics to ensure a smooth and interactive experience.

P2. The interactive charts and graphs shall respond to user actions (hover, click) within **500 milliseconds**, ensuring dynamic engagement.

P3. The platform shall support simultaneous access by **at least 100 concurrent users** without response time degradation beyond **10%**, ensuring scalability for a growing user base.

P4. Grade uploads by professors shall be processed and displayed in real-time, with individual updates reflected within **5 seconds** and bulk uploads within **30 seconds**.

P5. The system will ensure real-time comparison of student scores with class averages, detecting and displaying anomalies (highest and lowest scores) within **2 seconds** of user request.

4.2 Safety and Security Requirements

S1. The login/signup portal shall require **administrator approval** before granting users access to prevent unauthorized registrations.

S2. The system shall enforce **two-factor authentication (2FA)** for professors and admins when modifying grades to prevent unauthorized changes.

S3. All user data, including student scores and personal information, shall be **encrypted using Argon2** both in transit (via HTTPS) and at rest (within the database).

S4. The platform shall implement **role-based access control to ensure that students can only view their own scores, while professors and admins have appropriate permissions to manage class-wide data.**

4.3 Software Quality Attributes

4.3.1 Reliability

R1. The system shall maintain **99% uptime**, ensuring consistent access for students and professors throughout the academic term.

R2. Automated **hourly backups** shall be conducted to prevent data loss, with a retention period of **30 days** for rollback purposes.

4.3.2 Maintainability

M1. The system shall follow a **modular and scalable architecture**, allowing seamless updates to grading criteria, visualizations, and analytics tools without major downtime.

M2. The codebase will adhere to **industry-standard best practices**, ensuring that new developers can onboard and contribute effectively.

4.3.3 Usability

U1. The UI will provide **tooltips and contextual help** for interactive graphs to guide users in understanding the data representations.

U2. The platform will maintain a **responsive design**, ensuring optimal functionality on desktops, tablets, and mobile devices.

U3. The navigation flow will be designed such that any primary function (view grades, request changes, analyze class performance) can be accessed within **three clicks** from the homepage.

4.3.4 Scalability

S1. The system will be designed to **support at least 500 users concurrently**, with an ability to scale dynamically based on usage spikes.

S2. The database will be optimized for handling large volumes of grade records, ensuring minimal latency during retrieval and analysis.

S3. The system will support **future expansion to additional visualization tools like Power BI**, allowing integration with more advanced analytics.

4.3.5 Security and Privacy

SP1. The system will ensure that **only verified institutional users** (students and faculty) can access the platform, preventing external users from exploiting grade data.

SP3. Student profiles and scores willl be **visible only to authorized users**, with clear distinctions between admin, professor, and student privileges.

Name	**Type**	**Description**	**Possible values/ Format**	**Operations/ Requirements**
UserType	*Enum*	Defines the type of user in the system	`{Professor, Student, Admin}`	Controls access to functionalities
`CourseID`	String	Unique identifier for a	`CSE101`, `MTH202`, etc.	Assigned when a course is

		course		created
`StudentID`	String	Unique identifier for a student	`STU12345, STU67890`	Assigned during user registration
`ProfessorID`	String	Unique identifier for a professor	`PROF001, PROF002`	Assigned during user registration
`Score`	Float (0-100)	Student's score in a particular course	`0 - 100`	Entered by professor, used for graphs
`ClassAverage`	Float (0-100)	Average score of all students in a course	`0 - 100`	Calculated from student scores
`GraphType`	Enum	Type of graph for score visualization	`{Bar, Line, Pie, Histogram}`	Selected by user to view scores
`LoginStatus`	Boolean	Tracks if a user is logged in	`{true, false}`	Required to access most features
`JoinedCourses`	List of Strings	Courses a student has joined	`[CSE101, MTH202]`	Allows student to view scores
`AddedScores`	Boolean	Tracks if scores have been entered for a course	`{true, false}`	Required to generate graphs
`ManageGraphs`	Function	Controls graph selection and visualization	Depends on `GraphType`	Used by students and professors
`AddStudentData`	Function	Allows professors to input scores	Requires `ProfessorID` & `CourseID`	

Appendix B - Group Log

4.3.1 Meeting 1 – Project Planning (February 17, 2025)

- Decided to work on **ScoreLens**, a tool to visualize student scores.
- Discussed **how the website should look** and what features it should have.
- Assigned tasks to team members:
    - **Feature selection** – Deciding what functions the website will include.
    - **Login/signup page** – Assigning who will create the user authentication system.
    - **Backend setup** – Assigning database and server integration work.
    - Completed final documentation and prepared the **Statement of Work**.

4.3.2 Meeting 2 – Development Progress (March 3, 2025)

- Reviewed progress on login/signup page.
- Worked on integrating the **backend with the frontend**
- Working on database – SQL Lite

4.3.3 Meeting 3 – Development Progress (March 10, 2025)

- Working on all major features:
    - **Login/signup system**
    - **Grade uploading and visualization**
    - **Interactive graphs and comparisons**

4.3.4 Group Activities

- Researched **data visualization** tools for better graph representation.
- Developing ScoreLens individual components.
- Worked together to solve technical issues and improve features.
- Wrote the **Statement of Work** (weekly reports).