`

Software Design Specifications

for

Visualization tool for analysing Student Scores

Prepared by: **Group Name: 17**

M. Neha Reddy	**SE22UCSE156**	**se22ucse156@mahindrauniversity.edu.in**
Krithi Pavagada	**SE22UCSE144**	**se22ucse144@mahindrauniversity.edu.in**
Niharika D	**SE22UCSE087**	**se22ucse087@mahindrauniversity.edu.in**
Mythri Sri Bhavani Dantu	**SE22UCSE076**	**se22ucse076@mahindrauniversity.edu.in**
Kolla Sai Pramitha	**SE22UCSE135**	**se22ucse135@mahindrauniversity.edu.in**
Gayatri kalsanka Nayak	**SE22UCSE100**	**se22ucse100@mahindrauniversity.edu.in**

Document Information

Title: Visualization tool for analyzing student scores	
Project Manager:-	**Document Version No:1.0**
	Document Version Date:07-04-2025
Prepared By: Group 17	**Preparation Date:07-04-2025**

Version History

Ver. No.	Ver. Date	Revised By	Description	Filename

Table of Contents

1 Introduction

This Software Design Specification (SDS/SDD) outlines the detailed software design for the **Student Performance Visualization and Community Platform**. The system aims to empower students with personalized academic insights and foster collaborative learning through visual data analytics and a social community. It enables students to visualize their performance trends, compare with peers, receive improvement suggestions, and connect with mentors or professors. Professors can also track the overall academic performance of their classes and interact with students.

This document serves as a foundation for developers, stakeholders, and testers to understand the software's internal structure, data flow, module interaction, and user interface components.

1.1 Purpose

The purpose of this document is to define the software design specifications for the Student Performance Visualization and Community Platform. It outlines the architecture, components, data models, and system behaviour needed to build the application.

The intended audience for this document includes:

- **Developers:** to understand the structure and logic for implementing features.
- **Testers:** to design tests based on the architecture and module interactions.
- **Project Managers and Stakeholders:** to evaluate the feasibility and consistency of the design.
- **Professors and Academic Staff:** to provide feedback on the design based on educational requirements.

1.2 Scope

The objective of the Student Score Visualization Tool is to use visual analytics to help academic staff, instructors, and students better understand academic trends and results. Personalized dashboards that show semester-by-semester results, subject-by-subject breakdowns, and specific test components like Minor-1, Minor-2, End-Semester exams, and Lab evaluations are available to students. Although they are not allowed to view student data on an individual basis, faculty members can track trends in average performance over time. At the class or subject level, this supports academic planning, curriculum review, and performance analysis.

Among the essential features are:

- Dashboards with score breakdowns tailored to individual students
- Data visualization using graphs and charts
- Visualizes individual and class-level academic performance.
- Supports various data breakdowns (exam-wise, course-wise, semester-wise, year-wise).
- Compares a student's performance with peers and visualizes GPA trends.
- Uses machine learning to recommend peer mentors based on subject proficiency.
- Provides a community forum for anonymous academic discussions and Q&A.
- Allows professors to upload scores, track class statistics, and engage with students via direct messaging.
- Enables students to privately message professors for doubts or guidance.

The design of user-facing elements and interface logic are the main topics of this document. This specification does not cover administrative modules like data entry, student registration, authentication, and backend implementation.

1.3 Definitions, Acronyms, and Abbreviations

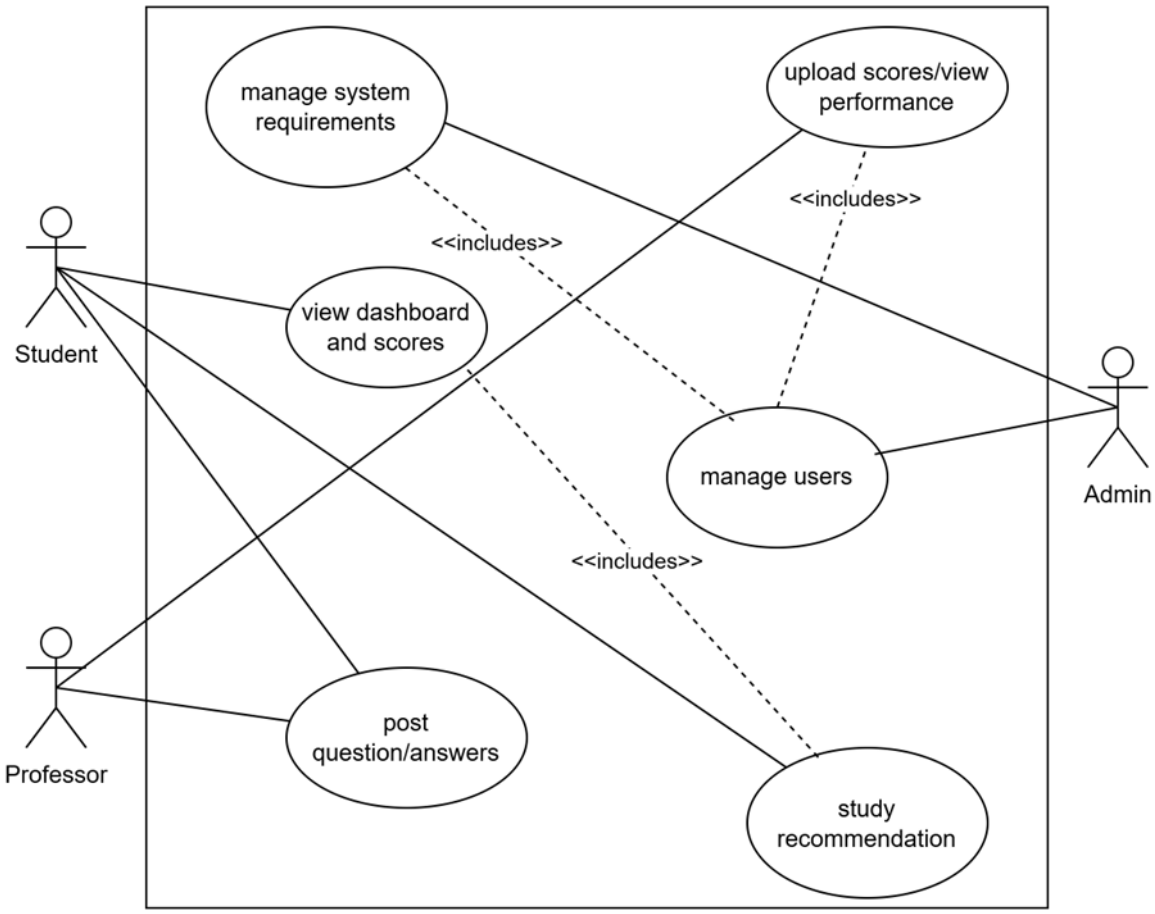
Table 1: List of Key terms, Acronyms, and Abbreviations

Term	Definition
API	Application Programming Interface
JSON	JavaScript Object Notation
DM	Direct Message
GUI	Graphical User Interface
GPA	Grade Point Average
ML	Machine Learning
UI	User Interface
UX	User Experience
DBMS	Database Management System
Faculty Dashboard	The interface for professors to analyze overall student performance trends.
Student Dashboard	The personalized interface where students view their semester-wise scores, grades, and performance insights.
User Authentication	The process of verifying a user's identity before granting access to the system.
Visualization	The representation of data through graphs, charts, and other graphical elements to enhance understanding.

1.4 References

- **IEEE Standard 1016-2009** – IEEE Standard for Information Technology (Software Design Descriptions) https://ieeexplore.ieee.org/document/5290780
- **System Requirements Specification (SRS) Document** – Defines functional and non-functional requirements for the Student Score Visualization Tool
- **High-Level Architecture Diagram** – Illustrates the structural design and component relationships in the system
- **Recharts Documentation** – A composable charting library used for rendering interactive data visualizations in React https://recharts.org/en-US
- **Material UI Documentation** – A popular React component library following Google's Material Design https://mui.com/material-ui/
- **React Documentation** – A JavaScript library for building user interfaces https://react.dev/
- **MongoDB Documentation** – NoSQL database used for storing and retrieving student performance data https://www.mongodb.com/docs/
- **Tailwind CSS Documentation** – A utility-first CSS framework for rapid UI development https://tailwindcss.com/

2 Use Case View



2.1 Use Case

2.1.1 Use Case #1

Author: Dantu Mythri
Purpose: Enables students to view their academic performance on a detailed dashboard.
Requirements Traceability: F2
Priority: High
Actors: Student

2.1.2 Use Case #2

Author: Gayatri K Nayak
Purpose: Allows faculty to update and view performance trends of student groups.
Requirements Traceability: F3
Priority: Medium
Actors: Faculty

2.1.3 Use Case #3: ML-Based Collaboration Recommendation (U3)

Author: Krithi Pavagada
Purpose: Matches weak-performing students with stronger-performing peers for guidance.
Requirements Traceability: F4
Priority: High
Actors: System, Student

2.1.4 Use Case #4: Forum Management (U4)

Author: Neha Reddy
Purpose: Enables students to post and respond to questions in a forum with optional anonymity.
Requirements Traceability: F5
Priority: Medium
Actors: Student

2.1.5 Use Case #5: User Account Management (U5)

Author: Niharika D
Purpose: Allows administrators to manage student and faculty accounts.
Requirements Traceability: F5
Priority: Medium
Actors: Administrator

2.1.6 Use Case #6: System Settings Management (U6)

Author: Kolla Sai Pramitha
Purpose: Allows administrators to manage system settings for customization and updates.
Requirements Traceability: F6
Priority: Medium
Actors: Administrator

3 Design Overview

The *Visualization of Student Scores* system adopts a microservices-based, three-tier architecture divided into the Frontend, Backend, and Database layers. This modular structure supports independent service deployment and maintenance. The Frontend, built using React.js and Recharts, handles dynamic data visualizations. The Backend, split between Django and Express.js (for APIs and user logic), manages business logic, real-time communication, and role-based access. Data is stored in MongoDB Atlas, ensuring scalability and flexible schema handling.

3.1 Design Goals and Constraints

Design Goals:

- **User-Centered Visualization**: The system must enable students to view their performance from multiple perspectives—semester-wise, subject-wise, exam-wise (Minor-1, Minor-2, End-Semester, Lab)—using clear and dynamic charts.
- **Faculty Support**: Faculty must be able to upload scores and access data trends for the classes and subjects they teach.
- **ML-Based Recommendations**: The system should intelligently identify students who are underperforming and match them with peers who excel in specific areas for collaborative improvement.

- **Interactive Communication**: A student forum and messaging feature should foster communication between users while respecting privacy and allowing anonymity.
- **Administrative Oversight**: Admins should be able to manage users, roles, and application configurations.

Design Constraints:

- **Technology Stack**:
    - Frontend: React.js with Tailwind CSi0o0S, Recharts, and Chart.js for visualizations.
    - Backend: Node.js, Express.js, Mongoose
    - Database: MongoDB
- **Role-Based Access Control (RBAC)** is mandatory to differentiate permissions and data visibility among students, faculty, and admins.
- **Performance**: Dashboards should load within 2 seconds; data queries should be optimized for latency.
- **Deployment**: Must support deployment to cloud platforms.
- **Maintainability**: The design must support modular updates—especially for ML and visualization components—without affecting other subsystems.
- **Security**: All sensitive data must be encrypted in transit and at rest. Authentication must be secure, with user passwords hashed.

3.2 Design Assumptions

This section outlines the assumptions considered during the design phase that may affect implementation and integration:

- All users (students, faculty, admins) will access the platform via modern browsers that support JavaScript.
- Each user has only one role at a time (Student, Faculty, or Administrator), and roles do not overlap.
- Data uploaded by faculty will follow a standardized format (e.g., CSV with pre-defined headers for student ID, subject code, marks, etc.).
- Academic data (e.g., marks, attendance) uploaded is assumed to be accurate and in the correct format.
- The user base is expected to scale linearly across semesters, with periodic data archiving to reduce load.
- Peer recommendation systems will update periodically.
- No offline mode is required functionality will be dependent on consistent internet connectivity.
- Users have access to stable network connections, ensuring minimal delays in visualization rendering or communication.
- Hosting services such as MongoDB Atlas will be available and reliable for deployment.

3.3 Significant Design Packages

The system is organized using a layered and modular architecture, facilitating separation of concerns, scalability, and maintainability. The major layers and packages are described below, reflecting the core responsibilities and interdependencies of the system components.

1. Presentation Layer (Frontend)

Tech Stack: React.js, Tailwind CSS, Material-UI, Recharts, Chart.js

Packages & Responsibilities:

- Dashboard Package: Visualizes individual student performance using line graphs, pie charts, radar charts, and bar charts across semesters, subjects, and exams.
- Comparison & Analytics Package: Displays class-level comparisons and historical performance trends.
- Forum & DM Package: Provides an interface for anonymous question posting and real-time messaging between users.
- Faculty UI Package: Tailored views for faculty to upload scores, see class-level analytics, and track performance.
- Admin Panel Package: Enables user management, system configuration, and access control.

These components interact with the backend via REST APIs connections for real-time updates and data rendering.

2. Application Layer (Backend)

Tech Stack: Node.js, Express.js, Mongoose

Packages & Responsibilities:

- Authentication & Authorization Module: Handles user login/registration, role-based access (RBAC), and secure session management.
- Score Upload & Validation Module: Facilitates bulk file upload (e.g., CSV), validates data formats, and stores results in the database.
- Recommendation Engine Module: Uses ML to identify struggling students and match them with peers performing well in the same subjects.
- Messaging Service Module: Enables secure real-time communication between students and professors.
- Admin Control Module: Provides endpoints for account creation, role updates, and system version management.

3. Data Layer (Database)

Tech Stack: MongoDB (via MongoDB Atlas), Mongoose ODM

Packages & Responsibilities:

- User Data Package: Manages collections for students, faculty, and admins.
- Academic Records Package: Stores score data, exam details, and GPA calculations.
- Communication Data Package: Manages forum posts, replies, and direct messages.
- Audit & Logs Package: Keeps track of data changes, uploads, and system logs for admin review.

Inter-Package Dependencies:

- The Frontend Dashboard consumes APIs exposed by the Backend Score & Analytics Module.
- The Recommendation Module operates asynchronously but updates the academic records and notifies the frontend upon processing.
- All user actions pass through the Authentication Package to enforce RBAC.

3.4 Dependent External Interfaces

The system relies on several external applications and services to support core functionalities such as data storage, authentication, deployment, and visualization. These dependencies are accessed by specific internal modules to fulfill business requirements.

External Application & Interface	Module Using the Interface	Functionality / Description
MongoDB Atlas	Database Access Module	Stores user data, scores, messages, and forum content.
Recharts/Chart.js/D3 (Visualization Libraries)	Student Dashboard, Faculty View	Used for rendering academic data as visual graphs and charts.
OAuth2 (tentative)	Authentication Module	May be used for secure login and session handling if third-party login is implemented.

3.5 Implemented Application External Interfaces (and SOA web services)

The system exposes a set of RESTful APIs that are used by the frontend to interact with backend services. These interfaces are implemented by different backend modules and support operations such as authentication, score uploads, data retrieval, and ML recommendations.

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/auth/login	Authentication Module	Logs in a user and returns a session token
/api/scores/upload	Faculty Module	Allows faculty to upload student scores
/api/scores/student	Student Dashboard Module	Sends student's score data for visualization
/api/forum/post	Forum Module	Posts or fetches questions and replies
/api/ml/recommendations	ML Recommendation Module	Returns professor-student suggestions
/api/admin/users	Admin Module	Lets admin manage users and roles

4 Logical View

This section presents the detailed design of the system, broken down into modules and the relationships between their respective components. The system follows a layered architecture to separate concerns and promote modularity, scalability, and ease of maintenance.

The system is decomposed into the following major layers:

1. **Presentation Layer (Frontend)** – Handles user interface and experience.
2. **Application Layer (Backend)** – Coordinates logic between frontend and data layers.
3. **Business Logic Layer** – Implements core features like performance analysis, comparison, matchmaking, etc.
4. **Data Access Layer** – Manages database operations.
5. **Machine Learning Layer** – Handles peer-matching and performance insights

4.1 Design Model

1. User Module

- Classes: Student, Professor, Admin, UserProfile, AuthenticationService
- Responsibilities: Handle login/signup, profile management, role-based access
- Key Attributes: user_id, name, role, email, password_hash

2. Score Module

- Classes: Score, Course, Exam, Semester, ScoreService
- Responsibilities: Store, update, and retrieve student marks; calculate GPA
- Relationships: Score belongs to Student, Course, Exam, and Semester

3. Visualization Module

- Classes: AnalyticsDashboard, ComparisonEngine, ChartGenerator
- Responsibilities: Display score graphs, peer comparison, performance trends
- Uses: Chart.js / D3.js in frontend for rendering visualizations

4. Recommendation Module (ML)

- Classes: Matchmaker, MLModel, StudentPerformanceProfile
- Responsibilities: Analyze students who need help and match them with top performers
- Algorithms: KNN / Similarity score using normalized course-wise GPA

5. Community Module

- Classes: Post, Comment, Thread, ForumService, AnonymousHandler
- Responsibilities: Enable anonymous Q&A and discussions among students

6. Messaging Module

- Classes: Message, ChatSession, Notification, DMService
- Responsibilities: Handle private messages between students and professors

7. Admin & Monitoring Module

- Classes: AdminDashboard, AuditLog, SystemMonitor
- Responsibilities: View system health, manage users, logs, and view analytics

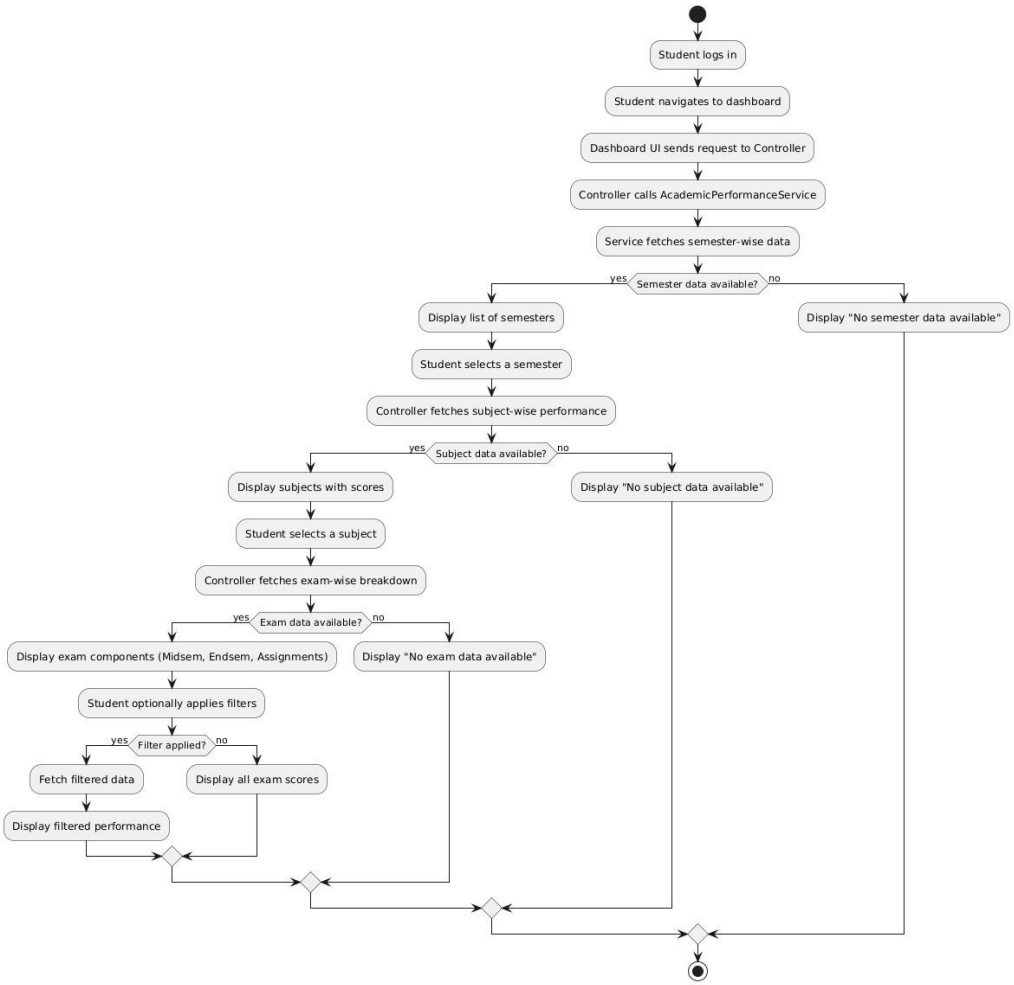
4.2 Use Case Realization

4.2.1 Use Case #1

Author: Dantu Mythri
Purpose: Enables students to view their academic performance on a detailed dashboard.
Requirements Traceability: F2
Priority: High
Preconditions: Student must be logged into their account.
Postconditions: Dashboard displays requested academic data.
Actors: Student
Flow of Events:

1. Student logs in.
2. Student navigates to the dashboard.
3. System displays semester-wise performance, subject breakdowns, and exam component scores.
4. Student applies filters for detailed insights.

Alternative Flow: If data is unavailable, the system notifies the student with an appropriate message.



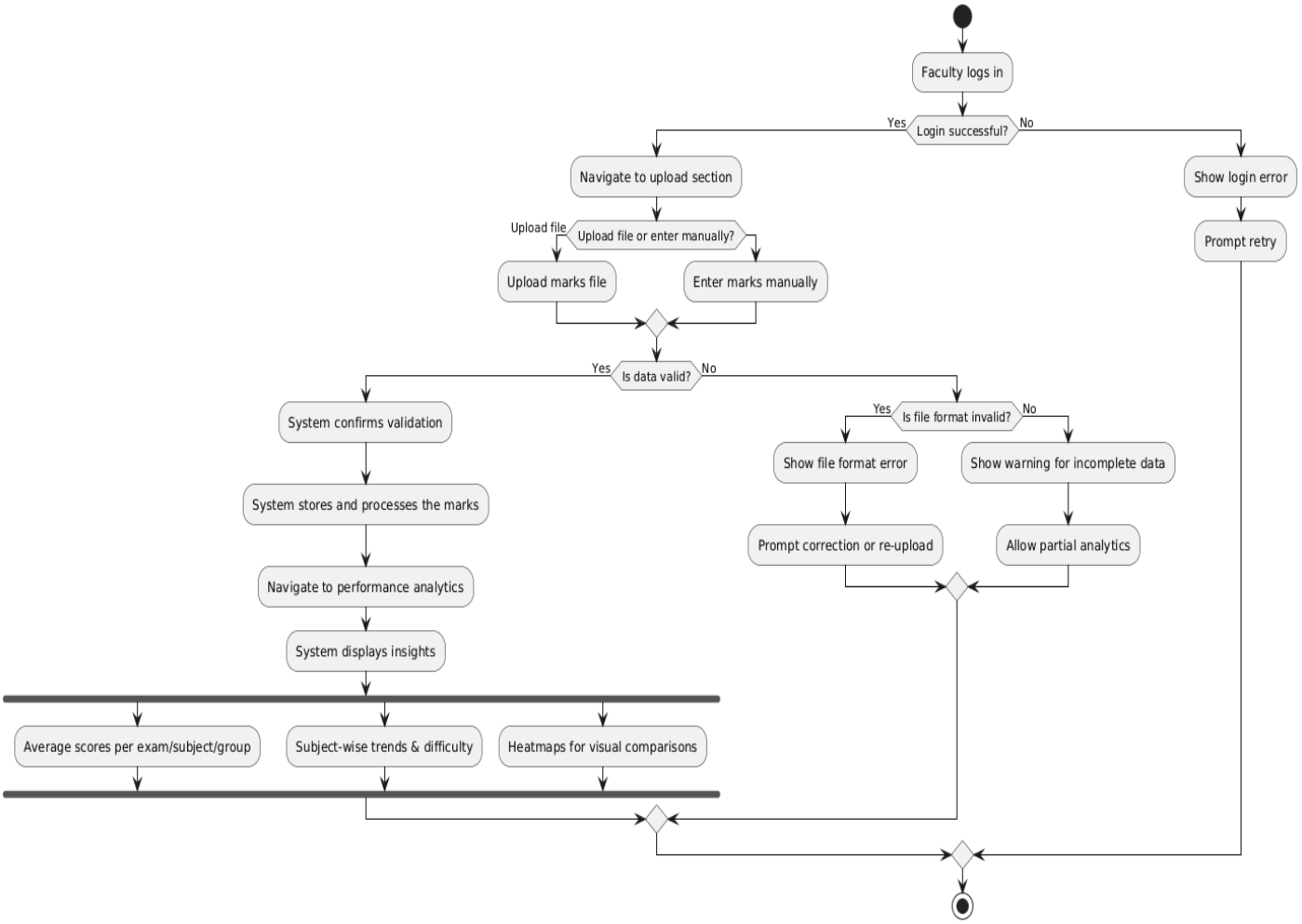
4.2.2 Use Case #2

Author: Gayatri K Nayak
Purpose: Allows faculty to update and view performance trends of student groups.
Requirements Traceability: F3
Priority: Medium
Preconditions: Faculty must be authenticated.
Postconditions: Faculty receives insights on performance trends.
Actors: Faculty
Flow of Events:

1. Faculty logs in.
2. Faculty navigates to upload section and uploads marks
3. Faculty navigates to the performance analytics section.
4. System displays average scores, subject trends, and heatmaps.

Alternative Flow:

1. Invalid Login: If login fails, the system shows an error and prompts retry.
2. Invalid File Format: If the uploaded file is malformed, the system prompts for correction or re-upload.
3. Incomplete Data: If marks are missing for certain students or exams, the system shows a warning and allows partial analytics.



4.2.3 Use Case #3: ML-Based Collaboration Recommendation (U3)

Author: Krithi Pavagada
Purpose: Matches weak-performing students with stronger-performing peers for guidance.
Requirements Traceability: F4
Priority: High
Preconditions: Student scores are available.
Postconditions: Recommended pairing is displayed to the student.
Actors: System, Student
Flow of Events:

1. Student logs in
2. Student navigates to subjects where they have poor performance.
3. Student clicks on the button ‘start pairing’ to get suggestions on strong peers
4. Student chooses and is taken to a personal forum to communicate with the consenting peers.

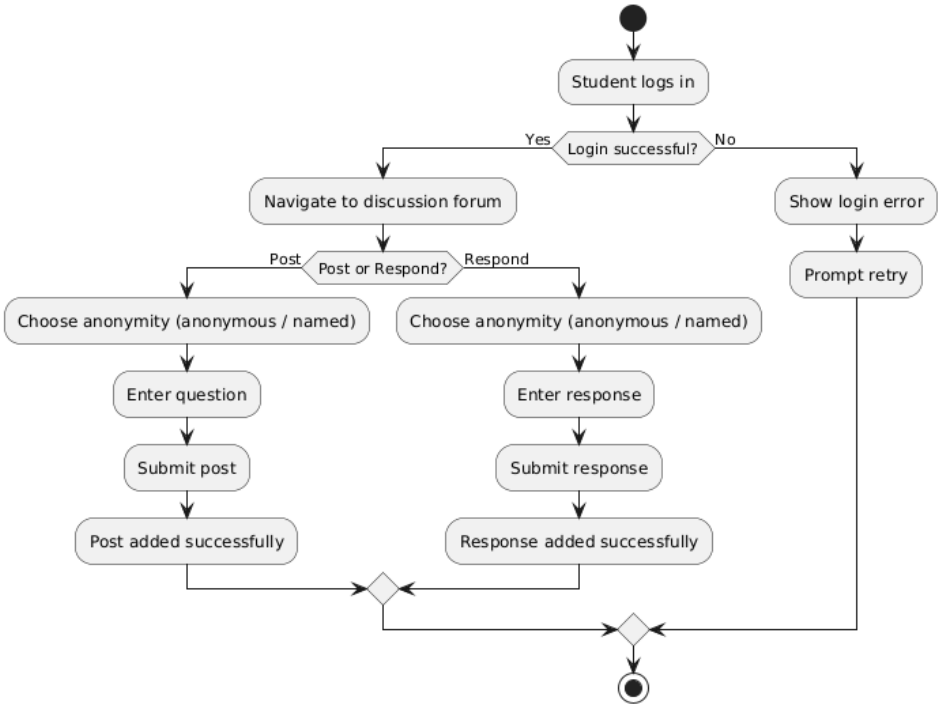
Alternative Flow:
Student who passed a subject can nominate themselves to be matched as a mentor

4.2.4 Use Case #4: Forum Management (U4)

Author: Neha Reddy
Purpose: Enables students to post and respond to questions in a forum with optional anonymity.
Requirements Traceability: F5
Priority: Medium
Preconditions: Student must be authenticated.
Postconditions: Forum post or response is successfully added.
Actors: Student
Flow of Events:

1. Student logs in
2. Student navigates to the discussion forum.
3. Student chooses anonymity status (anonymous or non-anonymous) and posts a question

Alternative Flow:
Student navigates to the discussion forum.
Student chooses anonymity status and answers the question.



4.2.5 Use Case #5: User Account Management (U5)

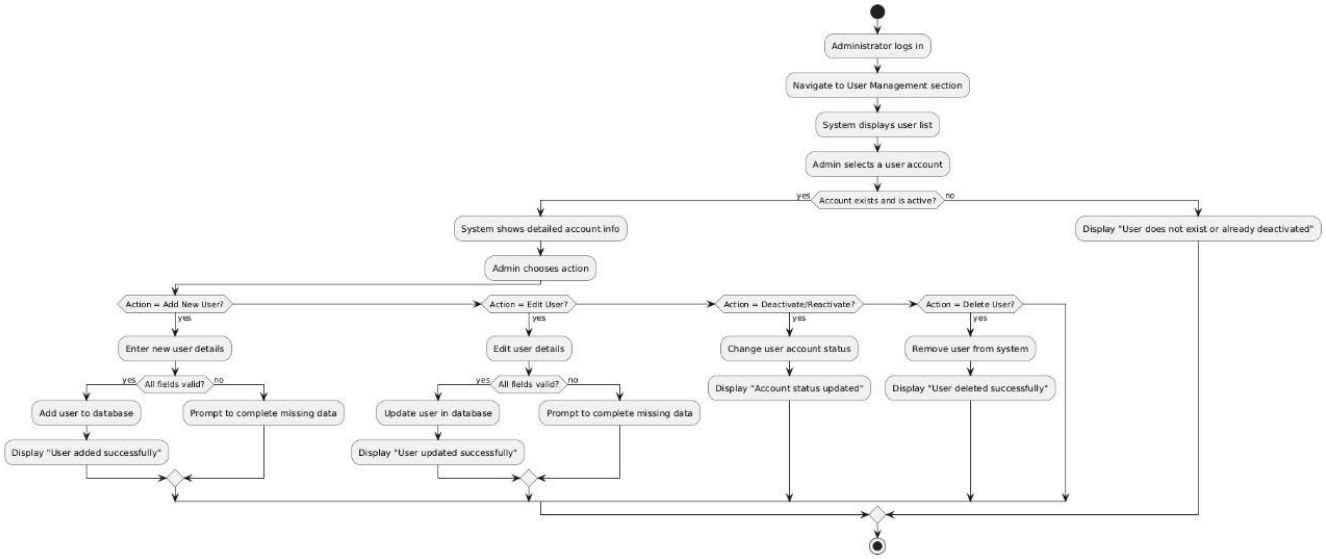
Author: Niharika D
Purpose: Allows administrators to manage student and faculty accounts.
Requirements Traceability: F5
Priority: Medium
Preconditions: Administrator must be authenticated.
Postconditions: System updates account information.
Actors: Administrator

Flow of Events:

1. Administrator logs in.
2. Administrator navigates to the User Management section.
3. The system displays a list of all registered users with roles, statuses, and details.
4. Administrator selects a user account to manage.
5. The system displays detailed account information.
6. Administrator performs one of the following actions:
   Add New User: Enters user details and assigns a role (Student/Faculty).
   Edit User Information: Modifies user details (e.g., name, email, role).
   Deactivate/Reactivate Account: Changes account status accordingly.
   Delete User Account: Permanently removes the account from the system.
7. The system verifies the changes, applies updates, and notifies the administrator of successful changes.

Alternative Flow:

If the selected account does not exist or has already been deactivated, the system displays an appropriate error message.
If required fields are incomplete during user creation or editing, the system prompts the administrator to fill in missing data.



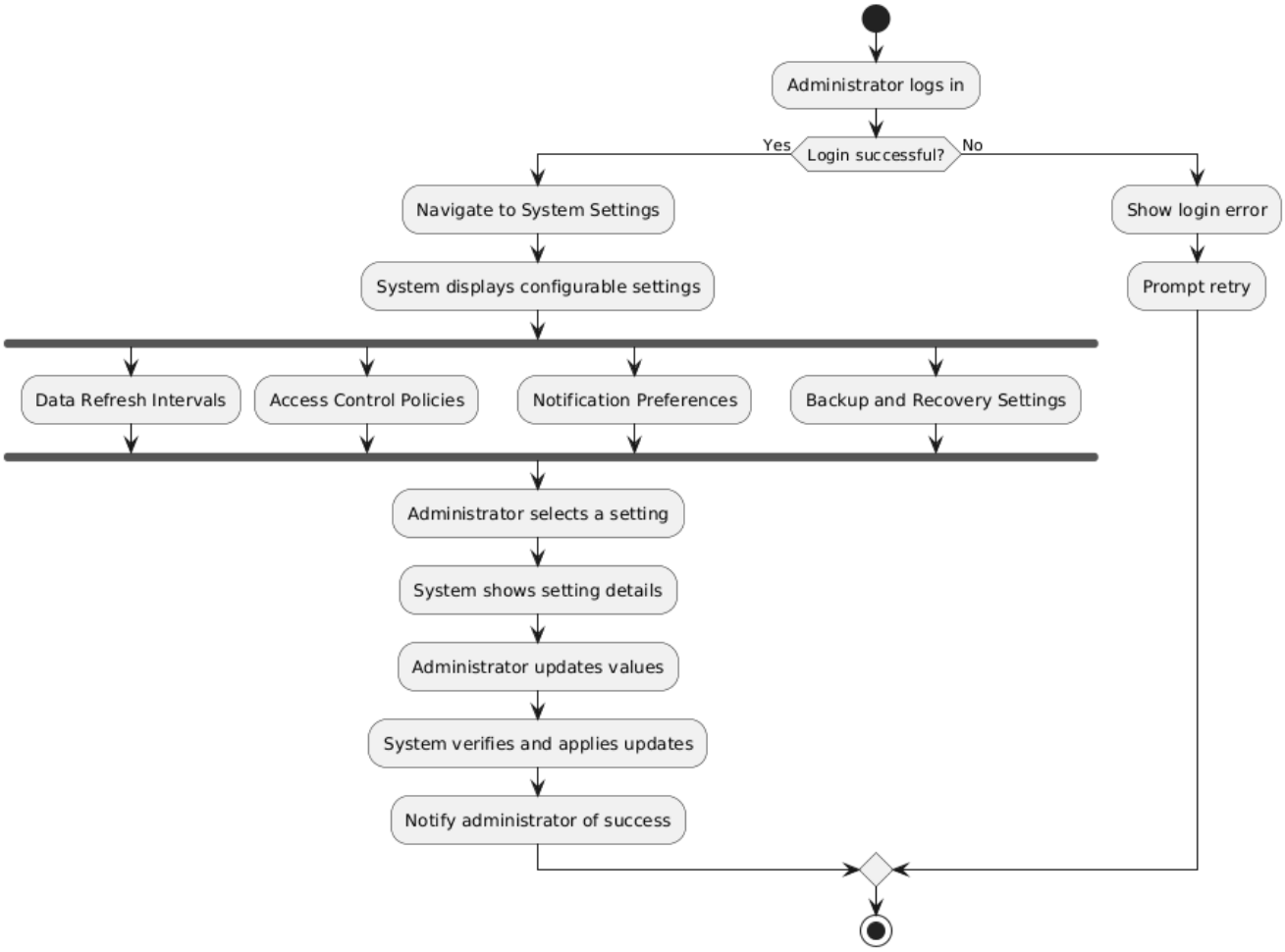
4.2.6 Use Case #6: System Settings Management (U6)

Author: Kolla Sai Pramitha
Purpose: Allows administrators to manage system settings for customization and updates.
Requirements Traceability: F6
Priority: Medium
Preconditions: Administrator must be authenticated.
Postconditions: System settings are updated successfully.
Actors: Administrator
Flow of Events:

1. Administrator logs in.
2. Administrator navigates to the System Settings section.
3. The system displays a list of configurable settings, such as:
   Data Refresh Intervals

Access Control Policies
Notification Preferences
Backup and Recovery Settings

4. Administrator selects a setting to modify.
5. The system displays the selected setting's details.
6. Administrator updates the desired values.
7. The system verifies the changes, applies updates, and notifies the administrator of successful changes.



5 Data View

This section outlines the persistent data storage perspective of the system. It includes a high-level domain model, a data model describing the database structure, and a data dictionary explaining each field used.

5.1 Domain Model

Entities and Descriptions

1. **User**
    a. Represents a person using the platform (student, instructor, admin)
    b. Attributes: user_id, name, email, password, role, profile_info, joined_date
2. **Course**

a. Represents a learning module or training unit
b. Attributes: course_id, title, description, category, created_by, created_on

3. **Enrolment**
   a. Represents the enrolment of a user in a course
   b. Attributes: enrollment_id, user_id, course_id, enrolled_on, progress, status
4. **Assessment**
   a. Represents quizzes, tests, or assignments
   b. Attributes: assessment_id, course_id, title, type, max_score, created_on
5. **Submission**
   a. Represents a user's attempt or answer to an assessment
   b. Attributes: submission_id, assessment_id, user_id, submitted_on, score, feedback
6. **AI Feedback**
   a. Represents automated insights or performance suggestions
   b. Attributes: feedback_id, user_id, course_id, feedback_text, generated_on
7. **Interaction Log**
   a. Represents logs of user actions (for analytics or recommendations)
   b. Attributes: log_id, user_id, action_type, timestamp, details
8. **Performance Trend**
   a. Captures historical performance metrics for visualization
   b. Attributes: trend_id, user_id, course_id, date, score, improvement_metric, comment
9. **Mentorship Connection**
   a. Links students with mentors or professors
   b. Attributes: connection_id, student_id, mentor_id, status, created_on, last_interaction

5.2 Data Model (persistent data view)

5.2.1 Data Dictionary

Field Name	Data Type	Description	Entity
user_id	ObjectId	Unique ID for user	User, Enrollment, Submission
name	String	Full name of user	User
email	String	Unique user email	User
password	String (hashed)	User login password	User
role	String	Role of user	User
profile_info	Object	Additional profile data (bio, pic, etc.)	User
joined_date	Date	Account creation timestamp	User
course_id	ObjectId	Unique ID of course	Course, Enrollment, Assessment
title	String	Course or assessment title	Course, Assessment
description	String	Course summary	Course
category	String	Subject category (e.g., Math, CS)	Course
created_by	ObjectId	Faculty who created the course	Course
created_on	Date	Timestamp of creation	Course, Assessment
enrollment_id	ObjectId	Unique enrollment identifier	Enrollment
enrolled_on	Date	Enrollment date	Enrollment
progress	Number	% completed in course	Enrollment

status	String	Enrollment or connection status	Enrollment, MentorshipConnection
assessment_id	ObjectId	Unique assessment ID	Assessment, Submission
type	String	Type of assessment	Assessment
max_score	Number	Maximum possible score	Assessment
submission_id	ObjectId	Unique ID per submission	Submission
submitted_on	Date	Submission timestamp	Submission
score	Number	Marks scored by user	Submission
feedback	String	Instructor's or system feedback	Submission
feedback_id	ObjectId	Unique feedback ID	AI_Feedback
feedback_text	String	ML-generated suggestion	AI_Feedback
generated_on	Date	When feedback was generated	AI_Feedback
log_id	ObjectId	Unique log ID	InteractionLog
action_type	String	e.g., login, viewCourse, submitTest	InteractionLog
timestamp	DateTime	When action occurred	InteractionLog
details	Object	Additional context about action	InteractionLog
trend_id	ObjectId	Unique trend record	PerformanceTrend
date	Date	Date of performance metric	PerformanceTrend
improvement_metric	Number	Relative score gain/loss	PerformanceTrend
comment	String	Instructor or ML annotation	PerformanceTrend
connection_id	ObjectId	Unique mentorship link ID	MentorshipConnection
student_id	ObjectId	ID of mentee	MentorshipConnection
mentor_id	ObjectId	ID of mentor (peer or faculty)	MentorshipConnection
last_interaction	DateTime	Most recent interaction timestamp	MentorshipConnection

6 Exception Handling

Exception Type	**Module Affected**	**Circumstances**	**Handling strategy**	**Follow-Up Action**
AuthenticationError	Authentication Module	Invalid login credentials, expired session	Return HTTP 401 with error message	Prompt user to re-login
ValidationError	Score Upload / Form Submissions	Incorrect or malformed data input (e.g., CSV format issues)	Return HTTP 400 with error details	Instruct user to fix the input format

FileUploadError	Score Upload Module	File missing, unsupported type, or too large	Return HTTP 422 with guidance	Show file requirements
MessageDeliveryError	Messaging Module	Chat server timeout or socket disconnection	Retry message send, fallback to polling	Inform user of message failure
UnknownError	Global Error Handler	Any uncaught or unexpected runtime exception	Return HTTP 500 with general message	Notify system admin and log stack trace

7 Configurable Parameters

This section defines the system's runtime-configurable parameters, useful for tuning performance, managing environments, or updating thresholds without code changes.

Configuration Parameter Name	Definition and Usage	Dynamic?
SESSION_TIMEOUT_MINUTES	Controls session expiration time in minutes for authenticated users	Yes
MAX_UPLOAD_FILE_SIZE_MB	Restricts the size of files (e.g., CSVs) that can be uploaded	Yes
PASSWORD_MIN_LENGTH	Minimum character requirement for passwords during registration	Yes

8 Quality of Service

This section outlines the critical aspects of system design that impact the platform's reliability, security, performance, and maintainability in a production environment. The goal is to ensure the system remains available, secure, performant, and observable under varying load conditions and user activity levels.

8.1 Availability

The application is expected to be available 24/7, with a target uptime of **99.5%** to support students and professors during peak academic periods, such as exam result days and semester-end reviews.

To meet this requirement, the design includes:

- **Redundant servers** hosted on a cloud-based infrastructure (e.g., AWS/GCP/Azure) with autoscaling enabled.
- **Database replication and automated backups** to reduce risk of data loss and ensure fast recovery in case of system failure.
- **Failover mechanisms** for critical services like score upload, messaging, and ML matchmaking.
- **Scheduled maintenance** will be communicated in advance and will be conducted during off-peak hours (e.g., midnight to 4 a.m.).
- Heavy operations like **mass data uploads** and **batch analytics computations** will be handled asynchronously using job queues to avoid affecting frontend availability.

8.2 Security and Authorization

Security is a core requirement to protect academic records and ensure safe user interaction.

Design considerations include:

- **Role-Based Access Control (RBAC):**
  - Students can view their own scores and analytics.
  - Professors can access class-wise analytics and upload scores for their respective courses only.
  - Admins can manage users and community forum moderation.
- **Secure Authentication** using JWT (JSON Web Tokens) for login and session handling.
- **Data Protection:** All sensitive data (e.g., scores, DMs) is encrypted in transit (HTTPS) and at rest.
- **Community Forum Anonymity:** Posts are stored with hashed IDs to preserve user privacy while preventing misuse.
- **Audit Logs** track all score uploads, data edits, and access attempts for accountability.
- Access control policies will be managed via a secure Admin Panel, with permissions customizable by system administrators.

8.3 Load and Performance Implications

The system is designed to support:

- **Up to 10,000 concurrent users**, especially during exam result announcements.
- **Score processing rate:** ~1,000 records per second during bulk uploads.
- **ML matchmaking latency:** Under 2 seconds for peer-matching suggestions.
- **Forum activity:** Up to 100 new posts/comments per minute.

Key performance strategies:

- **Indexing** on critical database columns (e.g., user_id, course_id, semester) for fast queries.
- **Caching** frequently accessed data (e.g., GPA trends, peer recommendations) using Redis or similar tools.
- **Horizontal scalability** of backend services using container orchestration tools like Kubernetes.
- Load testing will be conducted using tools like **Locust** or **Apache JMeter** to simulate exam week usage patterns.

8.4 Monitoring and Control

The application will include the following for operational monitoring:

- **Health check endpoints** for all microservices to ensure uptime and self-healing.
- **Logging and Error Tracking** using tools like ELK stack (Elasticsearch, Logstash, Kibana) or Sentry.
- **Performance metrics** (API response time, request volume, DB latency) exposed via Prometheus/Grafana dashboards.
- **Alerts and Notifications** for service failures, error spikes, or abnormal usage patterns.

Background jobs (e.g., peer recommendation engine, score batch processors) will be implemented as **controllable daemons**, monitored via scheduled logs and fail-safe restart policies.