Software Design Document

Student Score Visualization Tool

Version 1.0

Prepared by: Penumetsa Abhiram Varma

Group Number: 10

Member	ID	Email
Penumetsa Abhiram Varma	SE22UARI120	se22uari120@mahindrauniversity.edu.in
Arnav Reddy Padamati	SE22UARI021	se22uari021@mahindrauniversity.edu.in
Amarnath Reddy N	SE22UARI017	se22uari017@mahindrauniversity.edu.in
Yeruva Suprith Reddy	SE22UARI189	se22uari189@mahindrauniversity.edu.in
Jaya Siddu Karthikeya	SE22UARI191	se22uari191@mahindrauniversity.edu.in

Document Information

Title:	Student Score Visualization Tool - Software Design Document
Project Manager:	Penumetsa Abhiram Varma
Document Version No:	1.0
Document Version Date:	April 9, 2025
Prepared By:	Group 10
Preparation Date:	April 5-9, 2025

Version History

Ver. No.	Ver. Date	Revised By	Description	Filename
0.1	April 5, 2025	Penumetsa Abhiram Varma	Initial draft	SDD_v0.1.md

Table of Contents

1 INTRODUCTION

1.1 PURPOSE

This Software Design Document (SDD) document details the architectural and design specifications for the Student Score Visualization Tool. It translates the requirements defined in the Software Requirements Specification (SRS) into a comprehensive blueprint for implementation. This document serves as the primary reference for developers, testers, and stakeholders involved in building the system.

The document describes the decomposition of the system into design packages, the dependencies between packages, and the interfaces between them. It is intended for use by the development team, project managers, and other technical stakeholders involved in the implementation of the Student Score Visualization Tool.

1.2 SCOPE

This SDD covers the design of a web-based Student Score Visualization Tool that provides:

Interactive dashboards for students to visualize and analyze their academic performance

Tools for educators to review and analyze class-wide and individual student performance

Administrative features for user management and system configuration

Data visualization components including charts, graphs, and statistical summaries

The design incorporates frontend components, backend services, data storage solutions, and the relationships between these components to create a cohesive system.

The SDS describes the system from multiple perspectives, including logical organization, data structure, and system behavior. It also addresses the non-functional aspects such as security, availability, and performance.

1.3 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

UI/UX: User Interface/User Experience

API: Application Programming Interface

JWT: JSON Web Token

CRUD: Create, Read, Update, Delete

REST: Representational State Transfer

GPA: Grade Point Average

DBMS: Database Management System

RBAC: Role-Based Access Control

MERN: MongoDB, Express.js, React.js, Node.js

MUI: Material-UI (React component library)

CGPA: Cumulative Grade Point Average

SDS: Software Design Specifications

SRS: Software Requirements Specification

1.4 REFERENCES

1. IEEE Std 1016-2009, IEEE Standard for Information Technology—Systems Design—Software Design Descriptions

2. Software Requirements Specification (SRS) for the Student Score Visualization Tool, Version 1.0, April 1, 2025

2 USE CASE VIEW

2.1 USE CASES

The following use cases represent the key interactions between users and the Student Score Visualization Tool:

Student-Related Use Cases:

1. Login and Authentication Secure access to the system
2. View Personal Scores Students view their own academic scores
3. Analyze Performance Trends Visualize trends in personal academic performance
4. Compare with Batch Averages Compare personal scores with anonymized batch statistics
5. Export Reports Generate and download personalized academic reports

Educator-Related Use Cases:

6. View Class Performance Access aggregated class performance data
7. Analyze Individual Student Data Access detailed performance data for specific students

Admin-Related Use Cases:

8. Manage User Roles Assign and modify user access levels
9. Allows administrators to add, edit, and manage courses for different batches and assign educators to them
10. Configure System Settings Modify system-wide configuration parameters

Each use case is detailed with sequence diagrams showing data flow and component interactions in the Use Case Realization section.

3 DESIGN OVERVIEW

3.1 DESIGN GOALS AND CONSTRAINTS

Design Goals:

1. Modularity Create loosely coupled components that can be developed, tested, and maintained independently.
2. Scalability Design the system to handle increasing numbers of users and data volumes.
3. Security Implement robust authentication, authorization, and data privacy measures.
4. Responsiveness Ensure the system provides a fluid user experience with minimal loading times.
5. Maintainability Develop clear, well-documented code that can be easily maintained and extended.
6. Usability: Create an intuitive interface that requires minimal training for users.

Design Constraints:

1. Technology Stack:
   Frontend: React.js with Material-UI
   Backend: Node.js with Express
   Database: MongoDB
   Charts: Recharts for data visualization
   Authentication: JWT

2. Compliance Requirements: GDPR and FERPA for data privacy and protection
3. Development Timeline: Must be completed by May 1, 2025
4. Performance Requirements: Dashboard load time ≤ 3 seconds
5. Security Standard: JWT-based authentication and role-based access control

3.2 DESIGN ASSUMPTIONS

1. Student score data is in a structured format compatible with MongoDB imports.
2. The application will be hosted on university servers with sufficient computational resources.
3. Users will access the system via modern web browsers (Chrome, Firefox, Edge, Safari).
4. The system will not require real-time updates; data refresh at regular intervals will be sufficient.
5. User authentication will be handled through JWT-based authentication within the application.
6. The initial dataset size will be manageable within a MongoDB database.
7. Material-UI components will be sufficient for creating the required user interface elements.
8. Recharts library will provide all necessary visualization capabilities for student performance data.

3.3 SIGNIFICANT DESIGN PACKAGES

The system architecture is organized according to the following project structure:

1. Frontend Package (/frontend)
   Components
   Authentication Components (Login.js, Register.js)
   Student Components (StudentDashboard.js, StudentProfile.js)
   Admin Components (AdminDashboard.js, AdminProfile.js)

- Educator Components (EducatorDashboard.js, EducatorProfile.js)
- Common Components (Navbar.js, Home.js)
- CourseManagement.js
- Utils
  - API Services (api.js)
  - Theme Configuration (theme.js)
- Contexts
  - Authentication Context (AuthContext.js)

2. Backend Package (/backend)
   - Models
     - User Model (User.js)
     - Score Model (Score.js)
     - Course Model (Course.js)
   - Routes
     - Authentication Routes (auth.js)
     - User Management Routes (users.js)
     - Score Management Routes (scores.js)
     - Course Management Routes (courses.js)
   - Config
     - Database Configuration (db.js)
     - Middleware (middleware.js)

3. Data Visualization Package
   - Bar Charts for Course Performance
   - Line Charts for Semester Progress
   - Course Management Dashboards
   - Comparative Analytics Charts

4. Security Package
   - JWT Authentication
   - Role-Based Access Control
   - Data Encryption
   - Input Validation

3.4 DEPENDENT EXTERNAL INTERFACES

External Application/Interface	**Module Using the Interface**	**Description**
University Authentication Provider	Authentication Service	Validates user credentials against university database

3.5 IMPLEMENTED APPLICATION EXTERNAL INTERFACES

Interface Name	**Module Implementing the Interface**	**Description**
Score API	Score Processing Service	Provides CRUD operations for student scores
User API	User Management Service	Handles user creation, update, and role assignment
Analytics API	Score Processing Service	Provides statistical analysis of score data

4 LOGICAL VIEW

4.1 DESIGN MODEL

The Student Score Visualization Tool follows a modern React.js frontend architecture with a Node.js/Express backend and MongoDB database. The design follows component-based architecture principles for the frontend and a RESTful API approach for the backend.

Key Components and Their Relationships

1. Frontend Component Structure:

Authentication Components:

Login.js: Handles user authentication with JWT

Register.js: Manages user registration with role selection

Dashboard Components:

StudentDashboard.js: Central hub for student performance visualization

EducatorDashboard.js: Interface for educators to manage and view class performance

AdminDashboard.js: Admin control center for user and system management

CourseManagement.js :Interface for administrators to create and manage courses

Profile Components:

StudentProfile.js: Student information management

EducatorProfile.js: Educator information management

AdminProfile.js: Admin settings and configuration

Common Components:

Navbar.js: Navigation with role-based menu rendering

Home.js: Landing page with role-specific content

Visualization Components:

Using Recharts library for rendering:

Bar charts for course performance comparison

Line charts for semester-wise progress tracking

Donut charts for grade distribution visualization

2. Backend Structure:

Models:

User.js: Mongoose schema for user data with role-based fields

Score.js: Schema for academic performance records

Course.js: Course information schema

Route Controllers:

auth.js: Authentication endpoints (login, register, validate)

users.js: User management API endpoints

scores.js: Score management and analytics endpoints

courses.js: Course information endpoints

Services:

Authentication service with JWT implementation

Data validation and sanitization services

Analytics computation services

Course management services

3. Context Providers:
   AuthContext.js: Manages authentication state and user information
   Provides user role-based rendering and access control

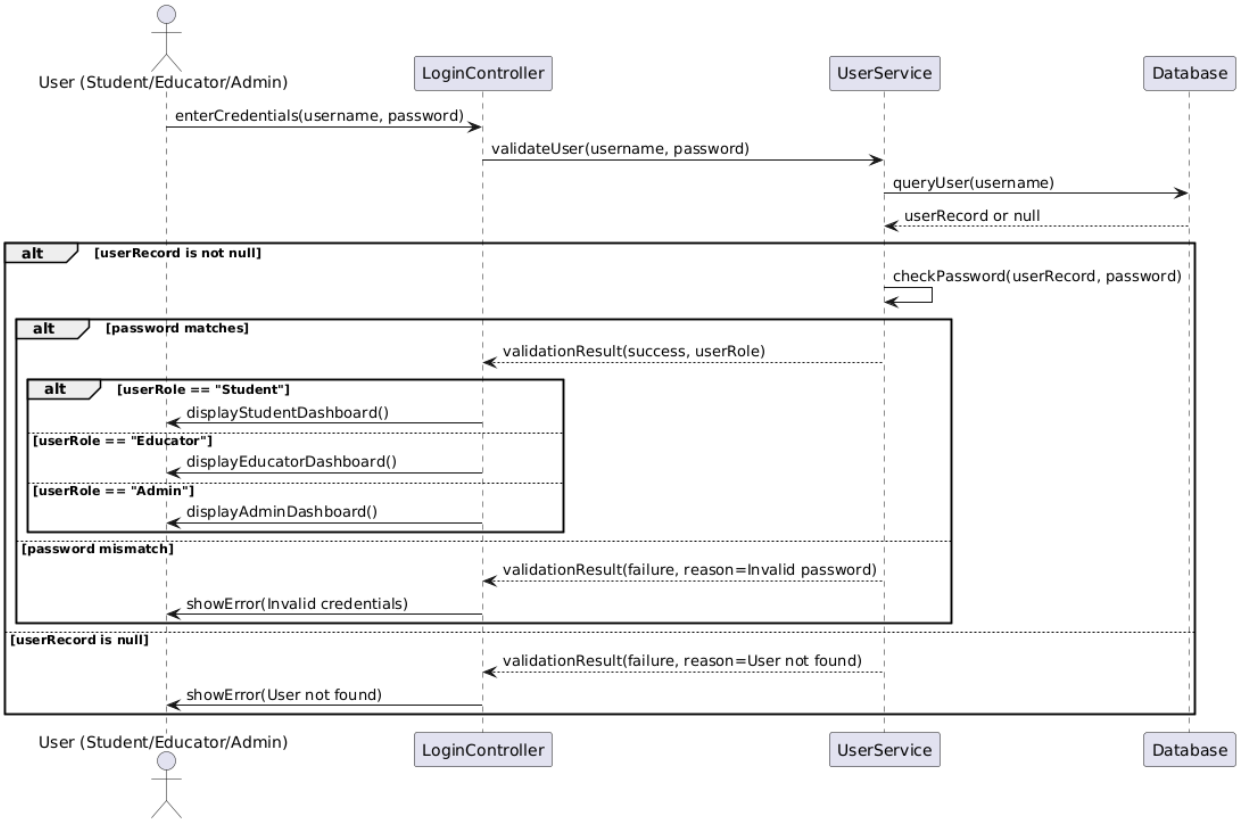
4. API Communication:
   api.js: Centralized API service with Axios for HTTP requests
   Handles request/response with proper error handling

5. Utility Functions:
   Data formatting and conversion utilities
   Chart configuration helpers
   Authentication helpers

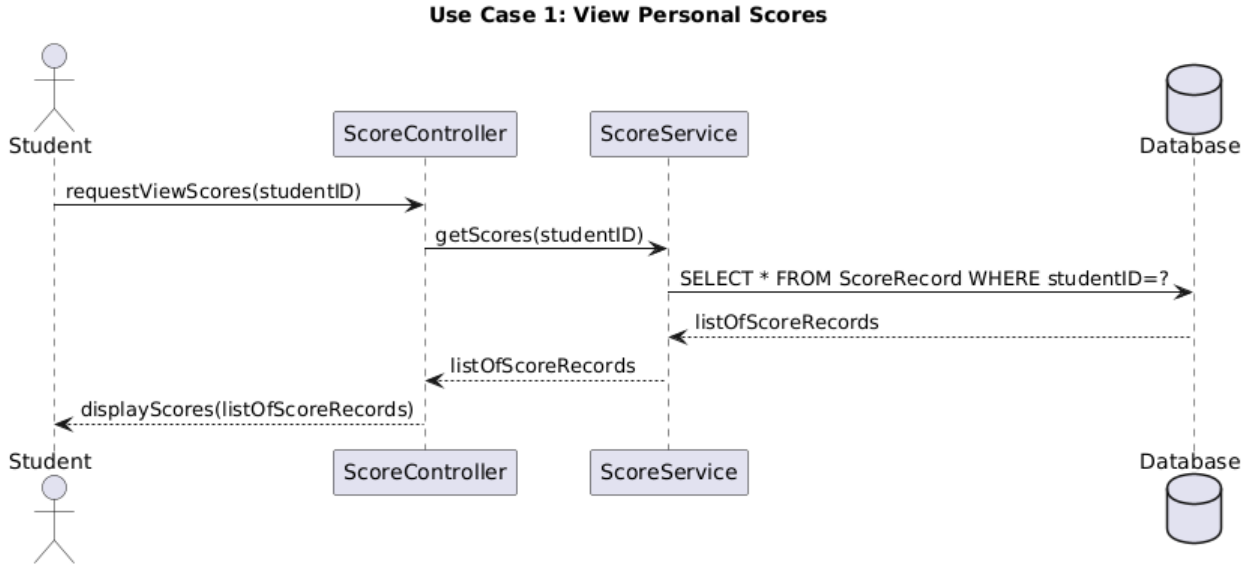
4.2 USE CASE REALIZATION

This section details how each use case is realized within the system architecture using sequence diagrams.

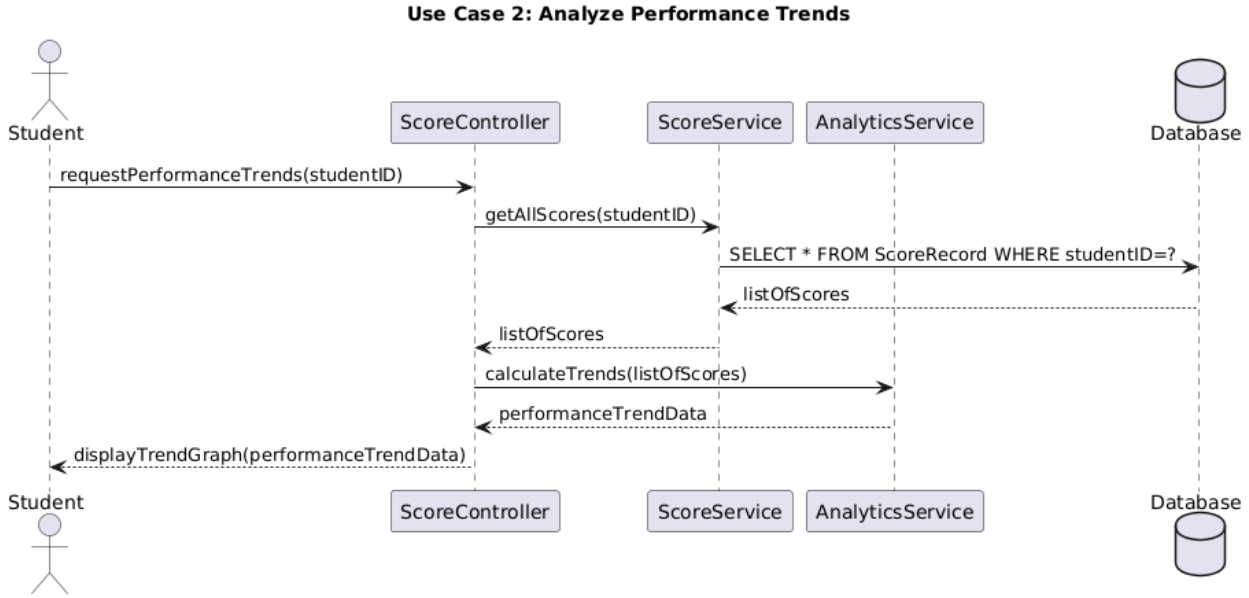
0. Login and authentication:



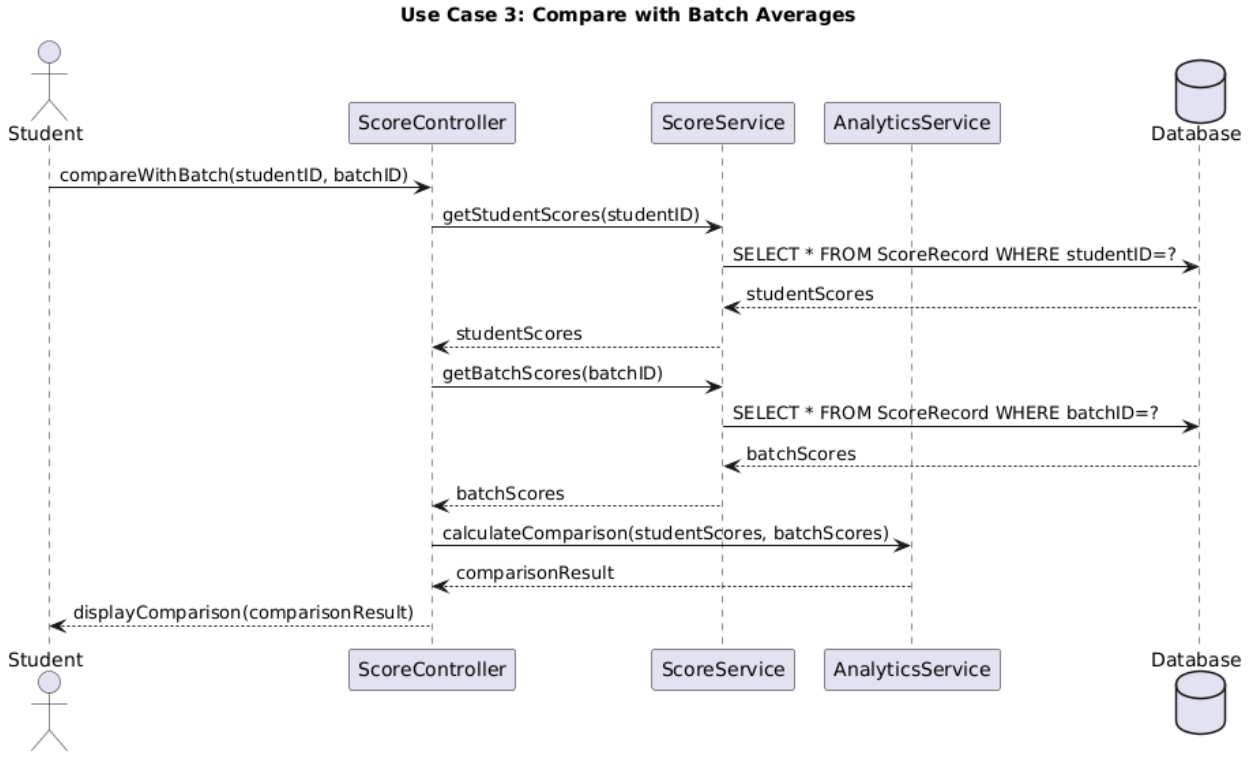
1. View Personal Scores:



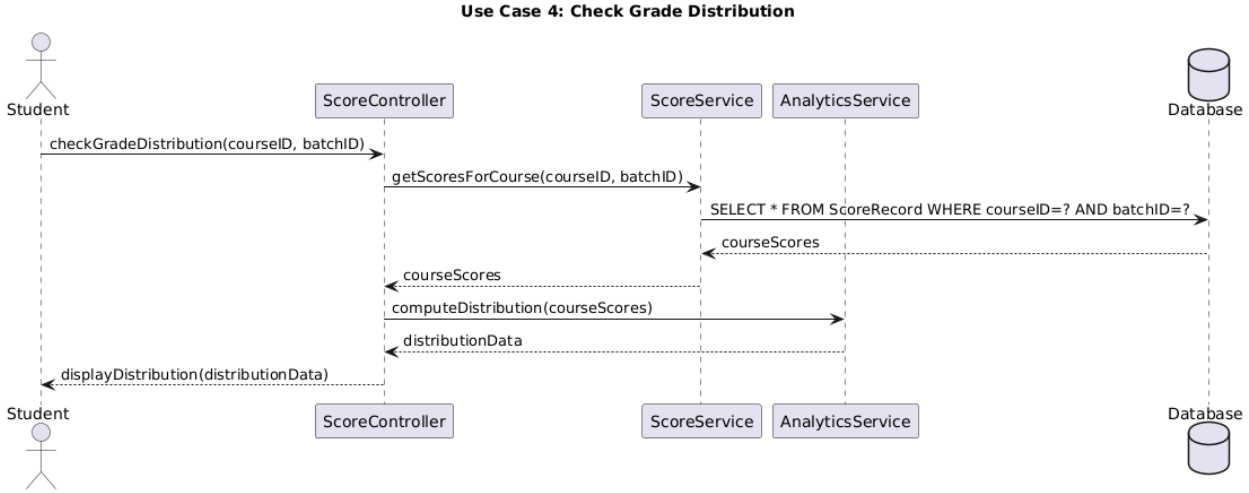
2. Analyze Performance Trends:



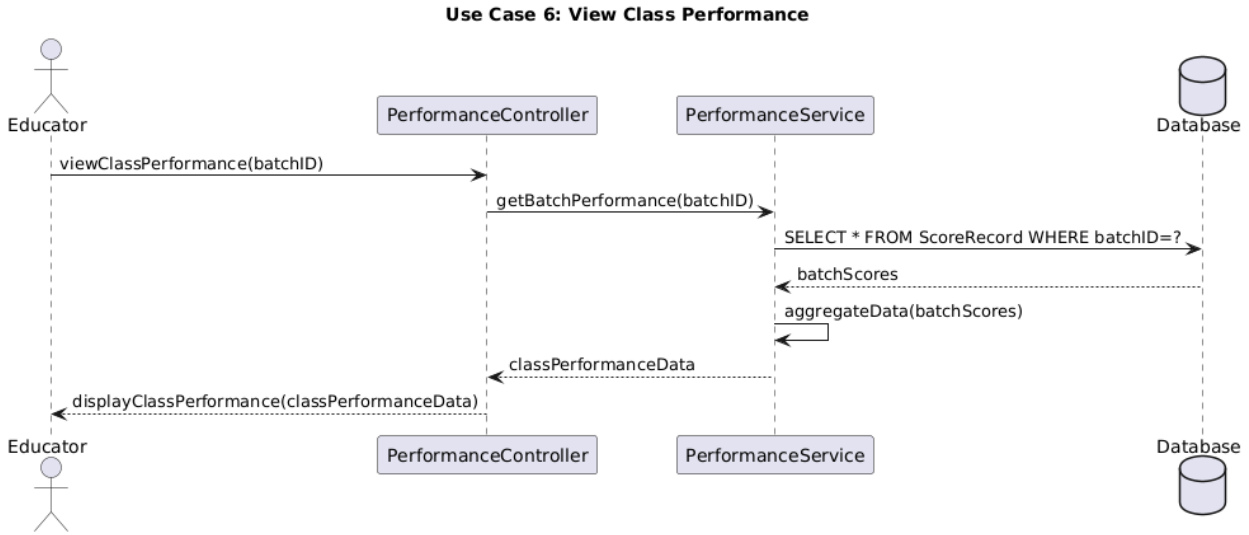
3. Compare with Batch Averages:



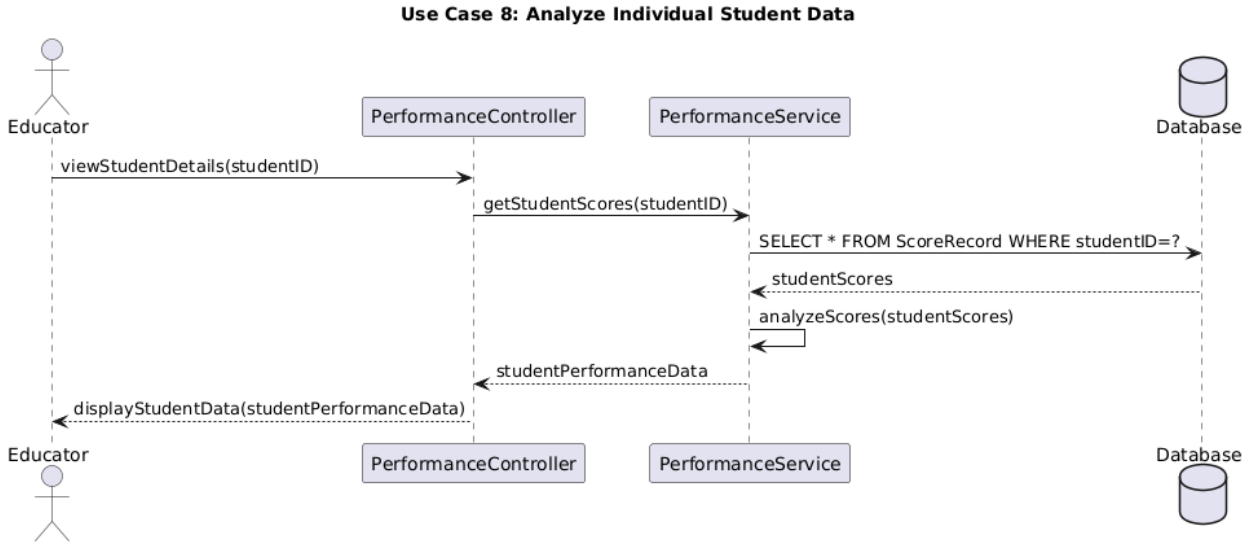
4. Check Grade Distribution:



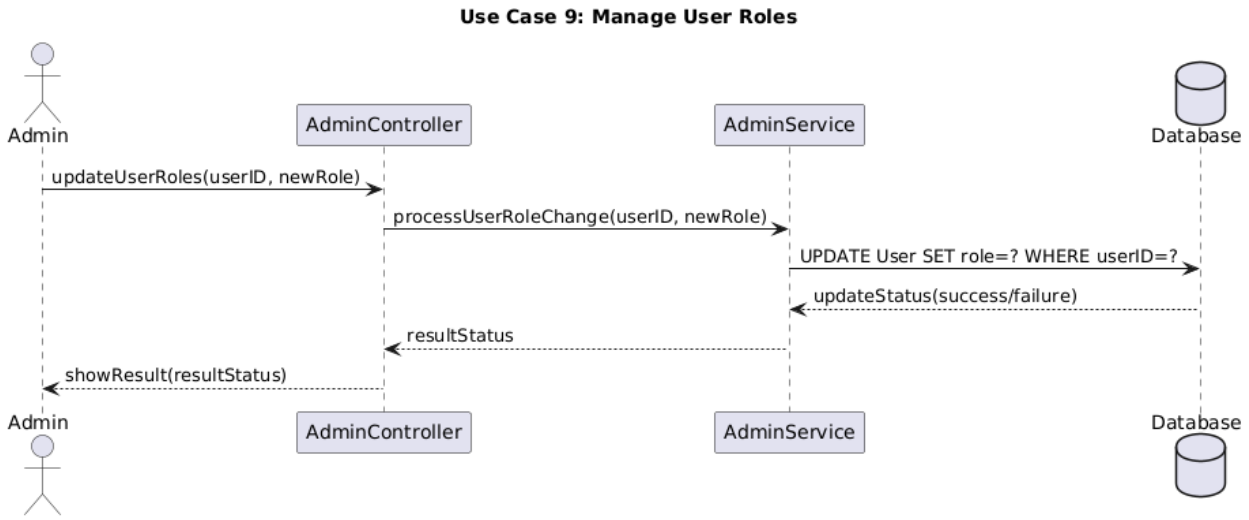
5. View Class Performance (Educator):



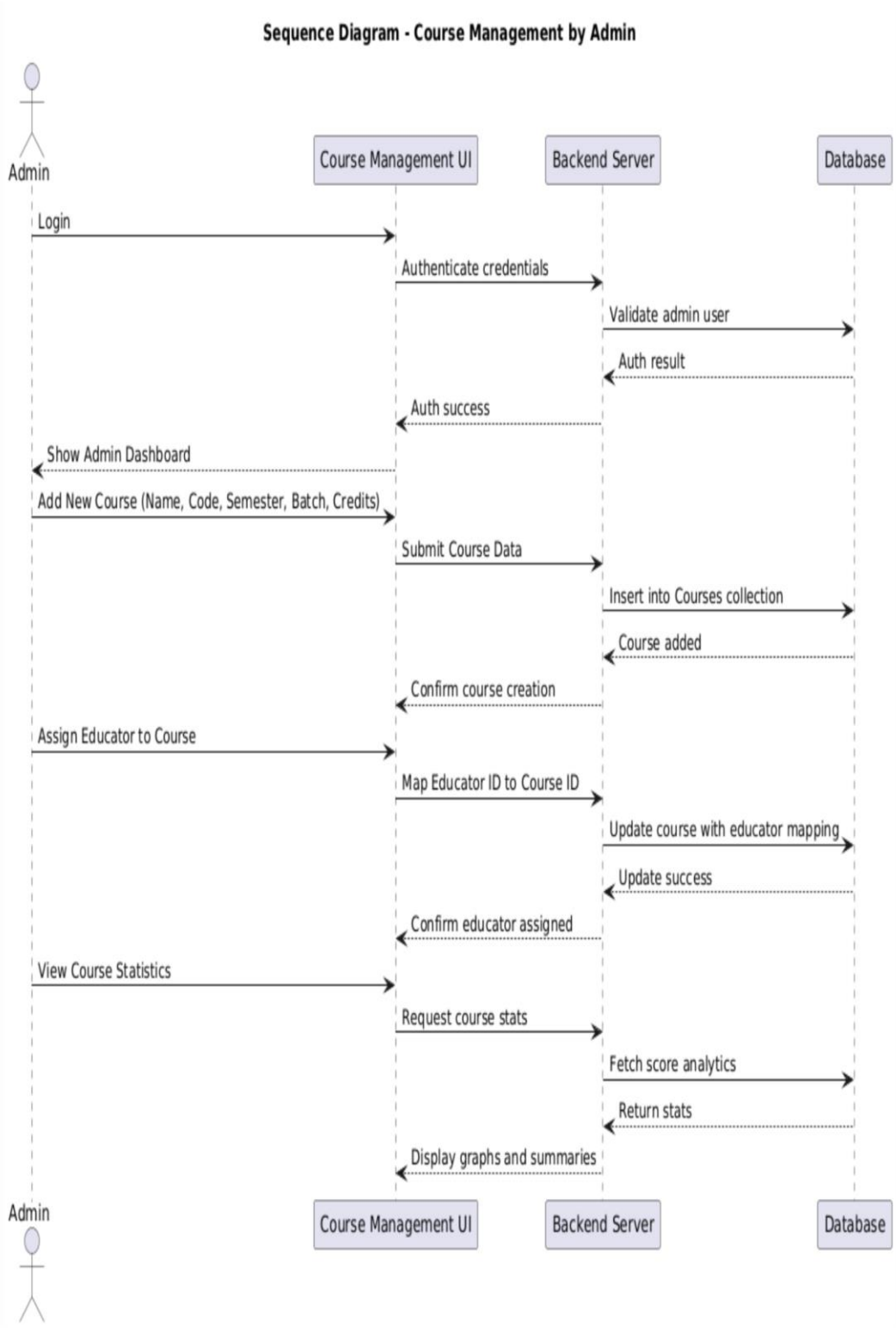
6. Analyze Individual Student Data (Educator):



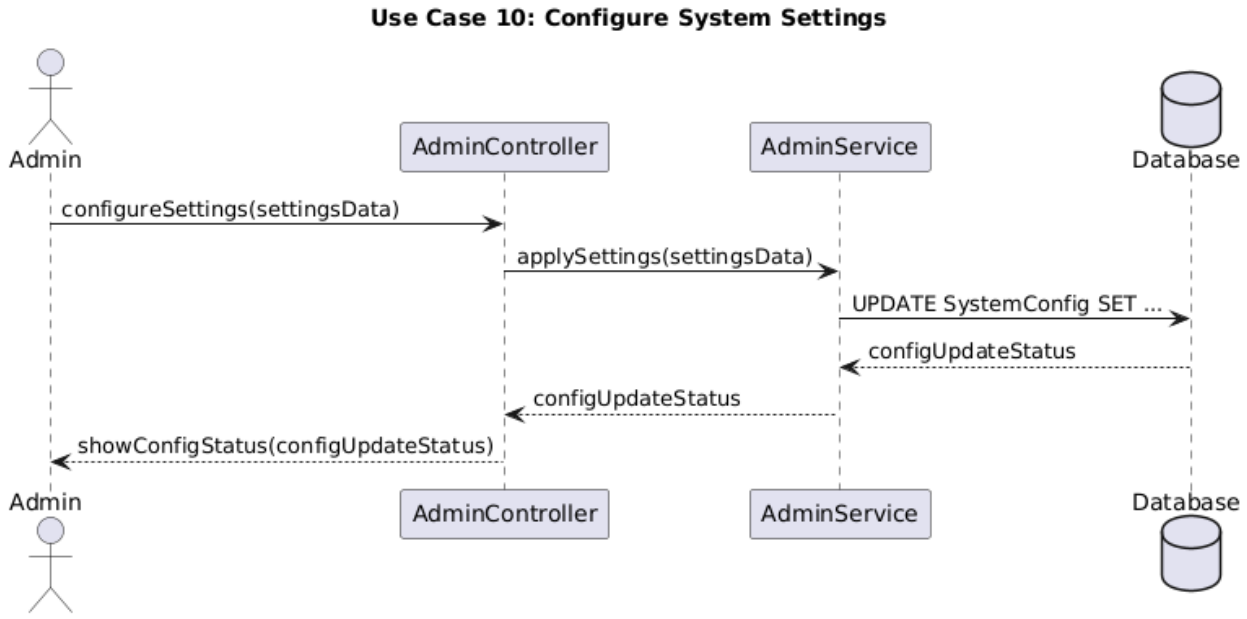
7. Manage User Roles (Admin):



8. Course management(Admin):


Sequence Diagram - Course Management by Admin
Admin
Course Management UI
Backend Server
Database
Login
Authenticate credentials
Validate admin user
Auth result
Auth success
Show Admin Dashboard
Add New Course (Name, Code, Semester, Batch, Credits)
Submit Course Data
Insert into Courses collection
Course added
Confirm course creation
Assign Educator to Course
Map Educator ID to Course ID
Update course with educator mapping
Update success
Confirm educator assigned
View Course Statistics
Request course stats
Fetch score analytics
Return stats
Display graphs and summaries
Admin
Course Management UI
Backend Server
Database

9. Configure System Settings (Admin):



5 DATA VIEW

5.1 DOMAIN MODEL

The domain model represents the key entities in the Student Score Visualization Tool and their relationships.

Key Entities:

User: Base entity for all system users

Student: Contains student-specific information

Educator: Contains educator-specific information

Admin: Contains administrator privileges

ScoreRecord: Individual score entries

Course: Course information

SystemConfig: System configuration parameters

Relationships:

Student has many ScoreRecords

ScoreRecord belongs to one Course

Admin creates and manages Courses

Course is assigned to one Educator

Educator teaches multiple Courses

Admin manages Users and SystemConfig

Course has many ScoreRecords

5.2 DATA MODEL (PERSISTENT DATA VIEW)

The persistent data model is implemented using MongoDB (NoSQL) with the following schema structure:

User Schema

Stores user authentication and profile information with role-based fields.

```
const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true,
    unique: true
  },
  password: {
    type: String,
    required: true
  },
  role: {
    type: String,
```

```
  enum: ['student', 'educator', 'admin'],
  required: true
},
// Student-specific fields
rollNo: {
  type: String,
  sparse: true,
  unique: true
},
batch: {
  type: String
},
graduationYear: {
  type: Number
},
department: {
  type: String
},
// Educator-specific fields
designation: {
  type: String
},
```

```
  // Common fields
  createdAt: {
    type: Date,
    default: Date.now
  },
  updatedAt: {
    type: Date,
    default: Date.now
  }
});
```

Course Schema

Stores course information.

```
const courseSchema = new mongoose.Schema({
  code: {
    type: String,
    required: true,
    unique: true
  },
  name: {
    type: String,
    required: true
  },
```

```
  credits: {
    type: Number,
    required: true
  },
  semester: {
    type: String,
    required: true
  },
  branch: {
    type: String,
    required: true
  },
  createdAt: {
    type: Date,
    default: Date.now
  },
  updatedAt: {
    type: Date,
    default: Date.now
  }
});
```

Score Schema

Stores individual score entries.

```
const scoreSchema = new mongoose.Schema({
  studentId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  courseId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Course',
    required: true
  },
  semester: {
    type: String,
    required: true
  },
  academicYear: {
    type: String,
    required: true
  },
  score: {
    type: Number,
```

```
  required: true
},
maxScore: {
  type: Number,
  required: true,
  default: 100
},
gradePoint: {
  type: Number,
  required: true
},
grade: {
  type: String,
  required: true
},
createdAt: {
  type: Date,
  default: Date.now
},
updatedAt: {
  type: Date,
  default: Date.now
```

```
  }
});
// Compound index for unique constraints
scoreSchema.index(
  { studentId: 1, courseId: 1, semester: 1, academicYear: 1 },
  { unique: true }
);



educatorId: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'User',
  required: true
},
isActive: {
  type: Boolean,
  default: true
},
batchYear: {
  type: Number,
  required: true
}
```

System Configuration Schema

Stores system configuration parameters.

```
const configSchema = new mongoose.Schema({
  key: {
    type: String,
    required: true,
    unique: true
  },
  value: {
    type: mongoose.Schema.Types.Mixed,
    required: true
  },
  description: {
    type: String
  },
  isEditable: {
    type: Boolean,
    default: true
  },
  updatedBy: {
```

```
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  updatedAt: {
    type: Date,
    default: Date.now
  }
});
```

5.2.1 DATA DICTIONARY

Term	**Description**	**Type**	**Constraints**
userID	Unique identifier for each user	VARCHAR(50)	Primary key, non-null
name	Full name of the user	VARCHAR(100)	Non-null
email	User's email address	VARCHAR(100)	Unique, non-null
passwordHash	Hashed user password	VARCHAR(255)	Non-null

role	User's role in the system	ENUM	'student', 'educator', 'admin'
studentID	Unique identifier for each student	VARCHAR(50)	Primary key, non-null
rollNo	Student's roll number	VARCHAR(50)	Unique, non-null
batch	Student's batch identifier	VARCHAR(50)	Non-null
courseID	Unique identifier for each course	VARCHAR(50)	Primary key, non-null
courseCode	Course code	VARCHAR(50)	Unique, non-null
courseName	Full name of the course	VARCHAR(200)	Non-null
credits	Number of credits for the course	INT	Non-null
scoreID	Unique identifier for each score entry	INT	Auto-increment, primary key
score	Student's score in a subject/test	FLOAT	0 to maxScore

maxScore	Maximum possible score	FLOAT	> 0
gradePoint	Grade point equivalent of score	FLOAT	0.0 to 10.0
grade	Letter grade	VARCHAR(5)	'A', 'B', 'C', 'D', 'F'
semester	Semester identifier	VARCHAR(20)	Non-null
academicYear	Academic year	VARCHAR(20)	Non-null
educatorID	ID of educator assigned to course	OBJECTID	References User model
isActive	Course active status	BOOLEAN	true/false
courseStatus	Current status of the course	ENUM	active', 'inactive', 'upcoming
configKey	System configuration parameter key	VARCHAR(100)	Primary key, non-null
configValue	Configuration value	TEXT	Non-null

6 EXCEPTION HANDLING

The Student Score Visualization Tool implements a comprehensive exception handling strategy to ensure robustness and provide clear feedback to users. The following exceptions are defined and managed within the system:

1. AuthenticationException
   Thrown when login credentials are invalid or authentication fails
   Logged with user ID (if available) and timestamp
   Displayed to user as "Invalid credentials. Please try again."

2. AuthorizationException
   Thrown when a user attempts to access unauthorized resources
   Logged with user ID, requested resource, and timestamp
   Displayed to user as "You do not have permission to access this resource."

3. DataNotFoundException
   Thrown when requested data is not found in the database
   Logged with search parameters and timestamp
   Displayed to user as "The requested information could not be found."

4. ValidationException
   Thrown when input data fails validation
   Logged with invalid fields, submitted values, and timestamp
   Displayed to user with specific validation error messages

5. DatabaseException
   Thrown when database operations fail
   Logged with operation details, error message, and timestamp
   Generic error message displayed to users while administrators are notified

6. CourseManagementException
   Thrown when course creation, update, or educator assignment fails
   Logged with course details, educator ID, and error message
   Displayed to admin as 'Unable to manage course. Please check details and try again.'

7. ConfigurationException
   Thrown when system configuration issues occur
   Logged with detailed configuration information
   Administrators are notified immediately

8. NetworkException
   Thrown when communication with external services fails
   Logged with service details and error message
   Displayed to user as "Network error. Please check your connection."

All exceptions are centrally logged using the Logging Service, which captures:

- Exception type and message
- User ID (if authenticated)
- Request details
- Timestamp
- Stack trace (in development environments)

7 CONFIGURABLE PARAMETERS

The following table describes the configurable parameters in the Student Score Visualization Tool. These parameters can be adjusted to modify system behavior without requiring code changes. Parameters marked as "Dynamic" can be modified while the system is running, while those marked "No" require a system restart to take effect.

Configuration Parameter Name	**Definition and Usage**	**Dynamic?**
session.timeout	Session timeout in minutes	Yes
password.minLength	Minimum password length	No

password.complexity	Password complexity requirements	No
course.allowMultipleEducators	Allow multiple educators per course	Yes
batch.anonymityThreshold	Minimum batch size for comparative analytics	Yes
chart.defaultColorScheme	Default color scheme for visualizations	Yes
email.notifications.enabled	Enable/disable email notifications	Yes
dashboard.refreshInterval	Data refresh interval in minutes	Yes
system.maintenanceMode	Enable/disable system maintenance mode	Yes
api.requestLimit	API rate limiting threshold	Yes
log.level	Logging detail level	Yes
security.failedLoginLimit	Maximum failed login attempts before lockout	Yes
cache.timeToLive	Cache expiration time in minutes	Yes

visualization.defaultType	Default visualization type for analytics	Yes
course.maxStudentsPerEducator	Maximum number of students per educator	Yes
course.requiredFields	Required fields for course creation	No
course.allowBatchReassignment	Allow reassigning courses to different batches	Yes

8 QUALITY OF SERVICE

8.1 AVAILABILITY

The Student Score Visualization Tool is designed to maintain high availability with the following characteristics:

1. Operational Hours: The system will be available 24/7 except during scheduled maintenance windows.

2. Scheduled Maintenance: Maintenance will be scheduled during low-usage periods (e.g., weekends or late nights) with advance notice to users.

3. Redundancy: The database will be configured with master-slave replication to prevent data loss in case of failure.

4. Monitoring: Continuous system monitoring will alert administrators of potential issues before they affect availability.

5. Recovery Plan: A disaster recovery plan is in place with regular database backups and restoration procedures.

6. Graceful Degradation: The system will maintain core functionality even if non-critical components fail.

8.2 SECURITY AND AUTHORIZATION

Security is a critical aspect of the Student Score Visualization Tool, particularly given the sensitive nature of academic data. The following security measures are implemented:

1. Authentication:
   Password hashing using bcrypt with appropriate salt rounds
   JWT (JSON Web Token) for session management
   Multi-factor authentication option for administrative accounts
   Automatic session timeout after configurable period of inactivity

2. Authorization:
   Role-Based Access Control (RBAC) with three primary roles: Student, Educator, and Admin
   Fine-grained permission system within each role
   Students can only access their own data
   Educators can access class-level data and individual student data for their classes
   Admins have full system access
   Course management permissions restricted to admin role only

3. Data Protection:
   Sensitive data encryption using AES-256
   HTTPS for all communications
   Data anonymization for batch comparisons
   Personally identifiable information (PII) protection
   Educators can only view courses assigned to them

4. Audit Logging:
   Complete audit trail of all data access and modifications
   Login/logout activity monitoring

Failed authentication attempts tracked for security analysis

8.3 LOAD AND PERFORMANCE IMPLICATIONS

The system is designed to handle the following load conditions:

1. User Load:
   Support for 1,000+ concurrent users
   Peak usage expected during exam result periods and start/end of semesters
   Load balancing implemented for user distribution

2. Performance Metrics:
   Dashboard load time: ≤ 3 seconds
   Chart rendering: ≤ 1 second
   Course management interface load time: ≤ 2 seconds
   Course creation and assignment: ≤ 1 second
   MongoDB query response: ≤ 500ms for 95% of queries

3. Data Volume:
   Initial data size: ~5GB
   Expected annual growth: ~2GB
   MongoDB collection indexing strategy for optimized queries

4. React.js Optimization:
   Component memoization using React.memo
   Virtual DOM optimizations
   Code splitting for improved load times
   Optimized re-rendering with proper state management

5. Caching Strategy:
   Client-side caching using browser localStorage for UI preferences
   React Context API for state management with minimal re-renders
   Material-UI component lazy loading

6. Optimization Techniques:
   MongoDB query optimization with proper indexing

Recharts configuration for optimized rendering
Data aggregation pipelines in MongoDB for analytics
Compression for data transfer using gzip/deflate
Minimized bundle size using webpack optimization

8.4 MONITORING AND CONTROL

The Student Score Visualization Tool incorporates comprehensive monitoring and control mechanisms:

1. System Monitoring:
   Real-time performance monitoring using Prometheus and Grafana
   Automated health checks for all services
   Database performance monitoring
   API call monitoring for usage patterns

2. Alerting System:
   Automated alerts for system anomalies
   Escalation procedures for critical issues
   Daily summary reports for administrators

3. User Activity Monitoring:
   Usage statistics by role and feature
   Peak usage time identification
   User behavior analytics for system improvement

4. Controllable Processes:
   Report generation queue management
   Data import/export scheduling
   Cache refresh mechanisms
   System maintenance mode activation

5. Measurable Values:
   System uptime percentage

Average response time
Database connection pool utilization
Memory usage
Error rates
User satisfaction metrics