PROJECT REPORT

On

Student Management System

Submitted in fulfilment of the requirement for the Course
Web-Development of

COMPUTER SCIENCE AND ENGINEERING
B.E. Batch 2023

APC-7923

Submitted From:	**Submitted By:**
Under the Guidance of	**Megha - 2310990736**
Ms. Anvi	**Mehak Garg-2310990737**
	Nandini Jhunjhunwal - 2310990747
	Nomita Patwal- 2310990755

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHITKARA UNIVERSITY

Abstract

The Student Management System (SMS) is a Java-based microservices project designed to efficiently manage student records and course enrollments in a modular, scalable, and fault-tolerant architecture. The system is built using Spring Boot and demonstrates the integration of microservices, RESTful APIs, Spring Cloud components, and database operations.

The project is divided into two core services:

- **Student Service** → manages student information.
- **Enrollment Service** → manages course enrollments, verifying student details via inter-service communication.

A **Spring Cloud Gateway** serves as the unified entry point, while **Spring Cloud OpenFeign** enables service-to-service calls. To ensure fault tolerance,**Circuit Breaker** is implemented, providing fallback responses when dependent services are unavailable.The backend uses **Spring JDBC (JdbcTemplate)** for database operations, with each microservice managing its own relational table (students and enrollments). The frontend is a **Java Console Application (CLI)** that interacts with the services through REST APIs exposed at the Gateway.This project not only showcases practical implementation of microservices in Java but also highlights important Spring concepts such as **Dependency Injection, IoC, Bean Scopes, AOP logging, and Actuator health checks**.

Keywords

- Student Management System, Microservices, Spring Cloud Gateway
- Spring Boot
- Microservices
- Spring Cloud Gateway
- OpenFeign
- Circuit Breaker
- REST API
- JDBC
- Dependency Injection
- AOP
- CLI Frontend
- Modular Architecture

Index

2. Introduction to the Project

In today's digital era, efficient and reliable management of student records and academic processes is a critical requirement for educational institutions. Traditional systems, whether manual or monolithic in nature, often fall short in providing the flexibility, scalability, and fault tolerance necessary to handle the growing volume of students and courses. To overcome these challenges, modern software architectures such as microservices have become the preferred approach for building distributed and resilient applications.

This project, the **Student Management System**, adopts a microservices-based architecture using **Spring Boot** and **Spring Cloud** to ensure modularity, scalability, and ease of maintenance. The system is composed of two core services: the **Student Service**, which manages student details, and the **Enrollment Service**, which handles course enrollments. These services communicate seamlessly through REST APIs, orchestrated via **Spring Cloud Gateway** for unified routing and API management. To enhance resilience, **Circuit Breaker** is integrated to safeguard against service failures, ensuring smooth user experience even under uncertain conditions.

The backend relies on **Spring JDBC with JdbcTemplate** for efficient database interactions, with each service maintaining its own schema for loose coupling. A lightweight **Java Console Application (CLI)** acts as the frontend, enabling users to interact with the system and perform operations such as student registration and enrollment management through the exposed REST APIs.

The primary objective of this project is to design and implement a **scalable, fault-tolerant, and modular Student Management System** that demonstrates core concepts of microservices, RESTful communication, database integration, and resilience in distributed systems. Beyond its academic utility, the project serves as a hands-on demonstration of modern backend development practices, making it a valuable learning resource for understanding enterprise-level application design.

2.1 Background

In earlier student management systems, academic records were often maintained manually or stored in monolithic applications that bundled all functionalities into a single unit. While these systems served their purpose initially, they quickly became difficult to manage as the number of students and courses grew. Making even small changes, such as adding a new feature or updating an existing function, required modifications across the entire system, which often introduced new bugs and reduced overall efficiency. Furthermore, scaling such systems to accommodate a larger user base was both resource-intensive and time-consuming, as the entire application had to be scaled rather than individual components.

With the rise of modern software development practices, microservices have emerged as a robust alternative to traditional monolithic architectures. By dividing functionality into smaller, independent services, each component of the system becomes more manageable, flexible, and reusable. These services run independently, communicate through lightweight REST APIs, and can be deployed or scaled without affecting the rest of the system. In the context of a Student Management System, this approach allows for seamless management of core academic operations such as student registration, course enrollment, and record maintenance, while ensuring higher scalability, maintainability, and fault tolerance compared to traditional systems.

2.2 Problem Statement

Traditional student management applications face several limitations that hinder their effectiveness in modern academic and professional environments. The lack of modularity in monolithic systems makes them difficult to maintain, as all functionalities are tightly bound together. Scalability is another pressing issue in monolithic architectures. As the number of students, courses, and administrative tasks grows, the system struggles to handle increased load efficiently.

Fault tolerance is also a significant concern. In a monolithic system, a failure in one part of the application can potentially bring down the entire system, leading to data unavailability and interruptions in service. This lack of resilience directly impacts the user experience and reduces trust in the system. Additionally, poor integration mechanisms in traditional systems make it challenging to achieve smooth communication between different modules, often resulting in delays, errors, or data inconsistencies.

To address these challenges, the Student Management System adopts a microservices-based architecture powered by Spring Boot. The system separates functionalities into two core services: the Student Service and the Enrollment Service. These services interact with each other through **Spring Cloud Gateway** and **OpenFeign**, ensuring smooth communication while maintaining modularity. To enhance reliability,**Circuit Breaker** has been integrated to provide fallback mechanisms during service failures. Database operations are handled using **Spring JDBC**, with each microservice managing its own schema to ensure loose coupling and independence. Finally, a simple yet effective **CLI frontend** allows users to interact with the system via REST APIs, providing an efficient, resilient, and scalable solution compared to conventional student management applications.

3. Software and Hardware Requirement Specification

To successfully develop and deploy the **Student Management System,** it is essential to define both the software and hardware requirements. This section outlines the necessary components for building, testing, and running the application efficiently.

3.1 Software Requirements

The application is developed using modern frameworks and tools that ensure scalability, maintainability, and reliability:

- **Programming Language**: *Java 17+* – the core language for developing backend microservices.
- **Framework**: *Spring Boot 3.x* with *Spring Cloud* – used to implement microservices, API routing, inter-service communication, and fault tolerance.
- **APIs**: *REST (via Spring Web)* – ensures smooth communication between services and the CLI frontend.
- **Database**: *H2 Database* (for demo) or *MySQL* – to store student and enrollment details securely.
- **Development Tools**: *Maven* for build automation and dependency management, *Postman* for API testing, and *IntelliJ IDEA/Eclipse* as the primary development environment.
- **Libraries**: *Spring Cloud OpenFeign* for inter-service communication, and *Spring Boot Actuator* for monitoring and health checks.

3.2 Hardware Requirements

The following hardware configuration is recommended for smooth execution of the project:

- **Processor**: Intel i3 or higher
- **RAM**: 4 GB minimum (8 GB recommended)
- **Storage**: At least 2 GB of free space
- **Operating System**: Windows, Linux, or MacOS

4. Database Analyzing, Design, and Implementation

Database Design

The database is the backbone of the Student Management System, ensuring efficient management of student records and course enrollments. The system is designed using a **relational database** (H2 for demo or MySQL for deployment), chosen for its reliability, structured schema support, and ability to maintain relationships between entities. The database schema ensures that essential entities such as **Students** and **Enrollments** are properly structured and interlinked, providing seamless operations for student registration and course enrollment.

Database Tables

The application uses two primary relational tables:

1. Students Table

Stores information about registered students who can be enrolled in courses.

Schema Example:

- **student_id**: Unique identifier for each student (Primary Key).
- **name**: Full name of the student.
- **email**: Unique email address of the student.

2. Enrollments Table

Maintains records of student course enrollments. Each enrollment entry links a student to a course.

Schema Example:

- **enrollment_id**: Unique identifier for each enrollment (Primary Key).
- **enrollment_course**: Foreign Key referencing the Students table.
- **course_credits**: Unique identifier of the enrolled course.

Database Relationships

- **One-to-Many Relationship**: A single student can have multiple enrollments.
- **Foreign Key Constraints**: Ensure data integrity between the Students and Enrollments tables.

- **Service-wise Database Separation**: Each microservice (Student Service and Enrollment Service) manages its own database schema, ensuring loose coupling and independence.

Database Implementation

- **Technology Used**:
  The system uses **Spring JDBC with JdbcTemplate** for interacting with relational databases. This ensures efficient CRUD operations with minimal configuration.

- **Data Security**:
  Validation is applied at both service and database layers to prevent invalid entries. Sensitive operations such as student registration and enrollment updates are validated using service-level checks.

- **Optimization**:
  Indexing is applied to frequently queried fields such as `student_id` to speed up searches and enrollment lookups. Queries are optimized to reduce redundancy and ensure smooth retrieval of student and enrollment details.

5. Key Features and CLI

The **Student Management System (SMS)** is designed to provide a seamless, user-friendly, and efficient platform for managing student records, enrollments, and academic operations. The following are the major features of the system, along with their implementation and representation in the CLI/frontend:

1. User Authentication

- **Functionality:**
  Users, including administrators, teachers, and students, can securely register and log in using unique credentials. Authentication ensures that sensitive information such as grades, enrollment records, and personal details remain protected.
- **Implementation:**
  The backend (Spring Boot) manages authentication and authorization. Passwords are securely hashed, and sessions are maintained to restrict unauthorized access.
- **CLI Interface:**
  Login and signup prompts for entering credentials. Error messages are displayed for invalid inputs. Successful login grants access to relevant functionalities based on user roles.

2. Student Management

- **Functionality:**
  Administrators can add, update, view, and delete student records. Users can list all students, and details include name, ID, course enrollments.
- **Implementation:**
  The Student Service exposes REST endpoints to manage student data. JDBC (JdbcTemplate) handles database operations for CRUD actions.
- **CLI Interface:**
  Menu options allow the admin to perform operations like adding a new student, listing all students, updating details, or removing a student. Inputs are collected via the console, and confirmations are displayed after each operation.

3. Course Enrollment Management

- **Functionality:**
  Students can be enrolled in courses, view their enrollments, or unenroll if needed. Administrators can list all enrollments for specific courses.
- **Implementation:**
  The Enrollment Service communicates with the Student Service via OpenFeign to verify student existence before enrollment. Circuit Breaker ensures fallback responses if the Student Service is unavailable. JDBC handles database operations for enrollment records.
- **CLI Interface:**
  Options for enrolling a student, viewing enrolled courses, and unenrolling from courses. Feedback is provided after each action, ensuring clarity.

4. Role-Based Access Control

- **Functionality:**
  Different roles have specific access:
  - **Admin:** Full access to manage students and enrollments.
  - **Teacher:** Can view enrolled students and course details.
  - **Student:** Can view personal details and enrolled courses.

  Implementation:
  Spring Security or custom role checks enforce access control at the API layer.
  CLI Interface:
  Menu options dynamically adjust according to user role, hiding unauthorized operations.

5. Reporting and Listing Features

- **Functionality:**
  Users can view lists of students, enrollments, and courses for easy tracking.
- **Implementation:**
  Backend APIs fetch required data and return structured responses. JDBC queries ensure efficient data retrieval.
- **CLI Interface:**
  Data is displayed in a tabular format in the console, showing key information such as student ID, name, enrolled courses, and enrollment status.

User Interface Overview (CLI Frontend):

- **Main Menu:** Presents all available options, including adding/viewing students, enrolling in courses, and exiting the system.
- **Input Prompts:** Collect necessary details such as student name, course ID, or enrollment ID.
- **Confirmation Messages:** Provide feedback after successful operations like "Student added successfully" or "Enrollment completed."
- **Error Handling:** Displays informative messages for invalid inputs, missing data, or failed operations.

6. Limitations

While the **Student Management System (SMS)** provides an effective and user-friendly platform for managing student records and course enrollments, there are certain limitations that may affect its current functionality or overall user experience. Addressing these in future iterations will enhance scalability, usability, and system robustness.

1. Limited Analytics and Reporting
Description: At present, the system offers only basic reporting features, such as student lists and enrollment summaries.
Impact: Administrators lack advanced insights into student performance trends, attendance patterns, or course effectiveness, limiting data-driven decision-making.

2. Scalability Constraints
Description: The current backend and database structure is optimized for moderate numbers of students and enrollments. Handling very large datasets or concurrent operations may lead to slower response times.
Impact: Institutions with a large student body may experience performance bottlenecks under heavy load.

3. No Dedicated Mobile Application
Description: The system is accessible via a desktop interface or web-based CLI frontend, but no mobile app is currently available.
Impact: Students and teachers cannot access the system conveniently on mobile devices, missing out on features like push notifications or real-time updates.

4. Limited Role-Based Features
Description: While basic roles like student, teacher, and admin are implemented, the system does not yet support granular role permissions or dynamic access control.
Impact: This may restrict flexibility in customizing workflows or granting specific privileges to different user types.

5. Offline Access Not Supported
Description: The system requires an active connection to the backend for all operations.
Impact: Users cannot perform critical tasks like updating student records or checking enrollments without network access, limiting accessibility in low-connectivity environments.

6. Lack of Integration with External Educational Tools
Description: The system currently operates as a standalone solution.
Impact: Features like automatic grade import from LMS, calendar sync, or integration with online learning platforms are not available, reducing interoperability.

7. Conclusion

The **Student Management System (SMS)** successfully addresses the challenges faced by educational institutions in managing student data, academic records, and administrative tasks by providing a centralized, secure, and user-friendly platform. Leveraging **Spring Boot** for the backend, **React/Next.js** for the frontend, and **MySQL** for the database, the system ensures scalability, maintainability, and efficient data handling.

Throughout the development process, the focus remained on **accuracy, accessibility, and automation** to reduce manual effort and improve productivity. Features such as **role-based authentication, attendance tracking, grade management, and intuitive dashboards** contribute to simplifying academic operations and enhancing collaboration between students, teachers, and administrators.

Despite existing limitations, such as the absence of mobile application support, limited analytics features, and basic reporting, the system lays a **strong foundation for future expansion**. The challenges faced during development—like ensuring secure data access, integrating multiple modules, and maintaining consistency across different operations—were successfully addressed through modern frameworks and best practices.

Overall, the Student Management System demonstrates the potential of technology in **transforming traditional education management into a smart, automated, and scalable digital solution**, paving the way for more advanced integrations in the future..

8. Future Scope

The Student Management System (SMS) has immense potential for future development to meet the evolving needs of educational institutions and learners. With advancements in technology and the growing emphasis on digital education, the following enhancements can be considered in future versions:

- **AI-Powered Student Analytics -** Implementing machine learning algorithms to analyze student performance, attendance, and engagement patterns will help identify at-risk students and provide personalized learning paths.
- **Integration with Learning Management Systems (LMS)-** Seamless integration with popular LMS platforms like Moodle, Google Classroom, or Canvas can streamline assignments, grades, and course content within a single platform.
- **Mobile Application Development-** Developing Android and iOS apps will allow students, teachers, and administrators to access SMS features on the go, improving accessibility and convenience.
- **Cloud Deployment & Scalability-** Hosting the system on cloud platforms (AWS, Azure, or GCP) will ensure high availability, scalability, and secure access for institutions of all sizes.
- **Biometric & RFID Attendance System-** Integrating biometric devices or RFID-based smart cards will automate attendance tracking, making it faster and more reliable.
- **Online Fee Payment & Financial Management-** Adding secure payment gateways (UPI, PayPal, credit cards, etc.) will allow parents and students to pay fees online while providing institutions with detailed financial reports.
- **Advanced Role-Based Dashboards-** Introducing custom dashboards for students, teachers, parents, and administrators will improve usability and provide relevant insights to each stakeholder.
- **Real-Time Notifications & Communication-** Enabling push notifications, SMS, and email alerts for attendance, exam schedules, grades, and announcements will strengthen communication between students, parents, and faculty.
- **Data Analytics & Reports for Admins-** Providing administrators with advanced reporting tools and analytics dashboards to track academic performance, resource utilization, and teacher effectiveness will help in informed decision-making.