DESIGN DOCUMENT for JDBC School Management Project

Version 1.0

Prepared by : 1. Rutul Patel (IMT2022021)
2. Varnit Mittal (IMT2022025)
3. Aditya Priyadarshi (IMT2022075)
4. Hemang Seth (IMT2022098)

Submitted to : Sujit Kumar Chakrabarti
Lecturer

November 14, 2024

Contents

1 Introduction

1.1 Overview

The JDBC project employs a three-tier architecture with presentation, business logic, and data access layers, enabling modularity and clear separation of concerns. Data Access Objects (DAOs) interact with the database using prepared statements to prevent SQL injection, while service classes manage business logic for user, course, and library modules. The project uses design patterns like DAO and Singleton for efficiency and includes a menu-driven command-line interface to streamline user interactions.

1.2 Objectives

The main objectives include −

- Enable **Student Enrollment and Management** to allow administrators to add, update, or remove students, ensuring that student records are current and accessible.
- Provide **Course Assignment and Management** capabilities for administrators to manage course offerings, assign students and teachers to courses, and maintain organized course records.
- Facilitate **Teacher Assignment and Salary Management** to add or update teacher details, assign teaching roles, and manage salary records, supporting human resource management.
- Implement **Grades and CGPA Calculation** to record student performance, aggregate scores, and ensure transparent grading and evaluation processes.
- Manage **Library Resources and Course Materials** to link books to specific courses and maintain an updated list of available materials for students and teachers.
- Develop **Reports and Analytics** to generate insights on student performance, teacher assignments, and academic trends for data-driven decision-making.
- Ensure **User Access and Security Controls** through role-based access, securing sensitive information and enforcing compliance.

2 High-Level Design

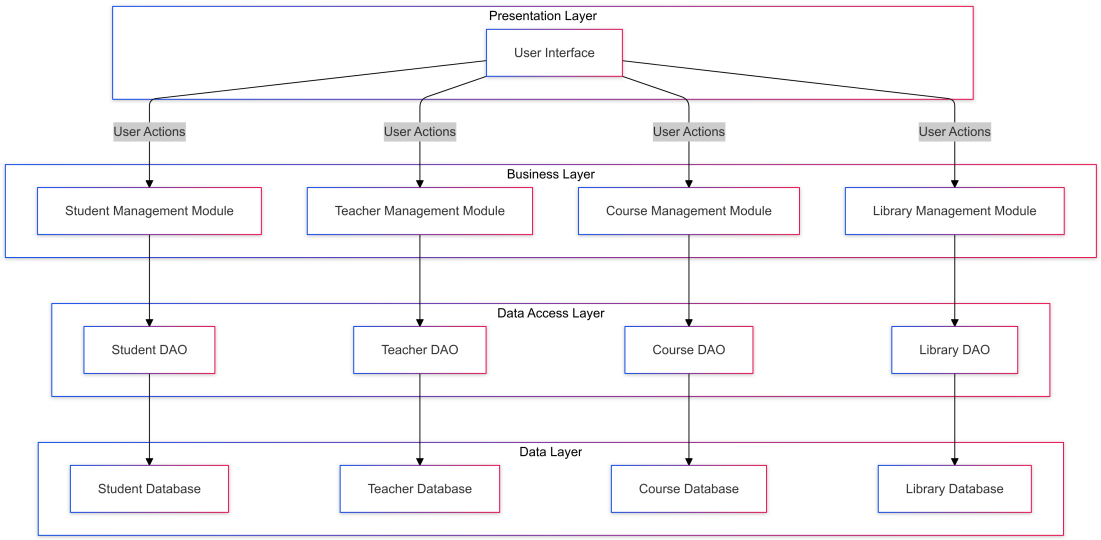

Figure 1: Architecture Diagram

2.1 Presentation Layer (Frontend)

The presentation layer in this JDBC project provides a command-line interface (CLI) that enables users to interact with the system through a simple, menu-driven format. It captures user inputs, validates them, and displays outputs, ensuring smooth navigation across various functions like managing students, teachers, courses, and library data. This layer is designed to be intuitive, minimizing user errors and directing commands to the business logic layer efficiently.

2.2 Business Logic Layer (Backend)

The business layer in this JDBC project handles core application logic, processing user requests from the presentation layer. It coordinates data operations for students, teachers, courses, and library resources by calling specific services for each module. This layer ensures that business rules are applied consistently, maintaining data integrity before interactions reach the data access layer.

2.3 Data Access Layer (Database)

- The data access layer in this JDBC project uses Data Access Object (DAO) classes to manage CRUD operations directly with the database, ensuring organized and reusable data handling.
- It employs prepared statements to securely execute SQL queries, preventing SQL injection and improving database interaction efficiency.
- This layer is abstracted from the business logic, making it adaptable to future changes in database structures or connection details without impacting other layers.

2.4 Module Breakdown

The JDBC project's architecture is divided into well-defined modules, each corresponding to distinct functionalities: User Management, Course Management, and Library Management. This modular breakdown is critical for maintainability, enabling targeted development, debugging, and enhancement without affecting unrelated parts of the system.

- **User Management Module:** This module is responsible for handling all operations related to student and teacher data, including registration, authentication, profile updates, and role management. The module is implemented with a Service Class (UserService) for core logic and a DAO Class (UserDAO) for data persistence. Through these classes, the module enables secure password handling (e.g., using hashing algorithms), input validation, and efficient data retrieval, making it extensible to support advanced functionalities like user role control.

- **Course Management Module:** The Course Management module provides functionality to manage course-related data, such as course creation, enrollment, and scheduling. It integrates seamlessly with both the User and Library Management modules, ensuring that course enrollment considers current student and teacher data. The CourseService class encapsulates business logic, enforcing course prerequisites or constraints, while the CourseDAO manages database transactions. This separation allows complex operations—such as updating course rosters or class schedules—to be streamlined and secure.

- **Library Management Module:** The Library Management module tracks resources (e.g., books and multimedia) and their availability. It enables inventory management, lending, and return transactions. Through its LibraryService and LibraryDAO classes, this module enforces lending rules, such as resource availability and user eligibility, ensuring data integrity. The module's ability to link with User Management (to restrict access to specific roles) and Course Management (to link resources with courses) exemplifies the system's cohesive structure.

3 Low-Level Design (LLD)

3.1 Class Diagram Overview

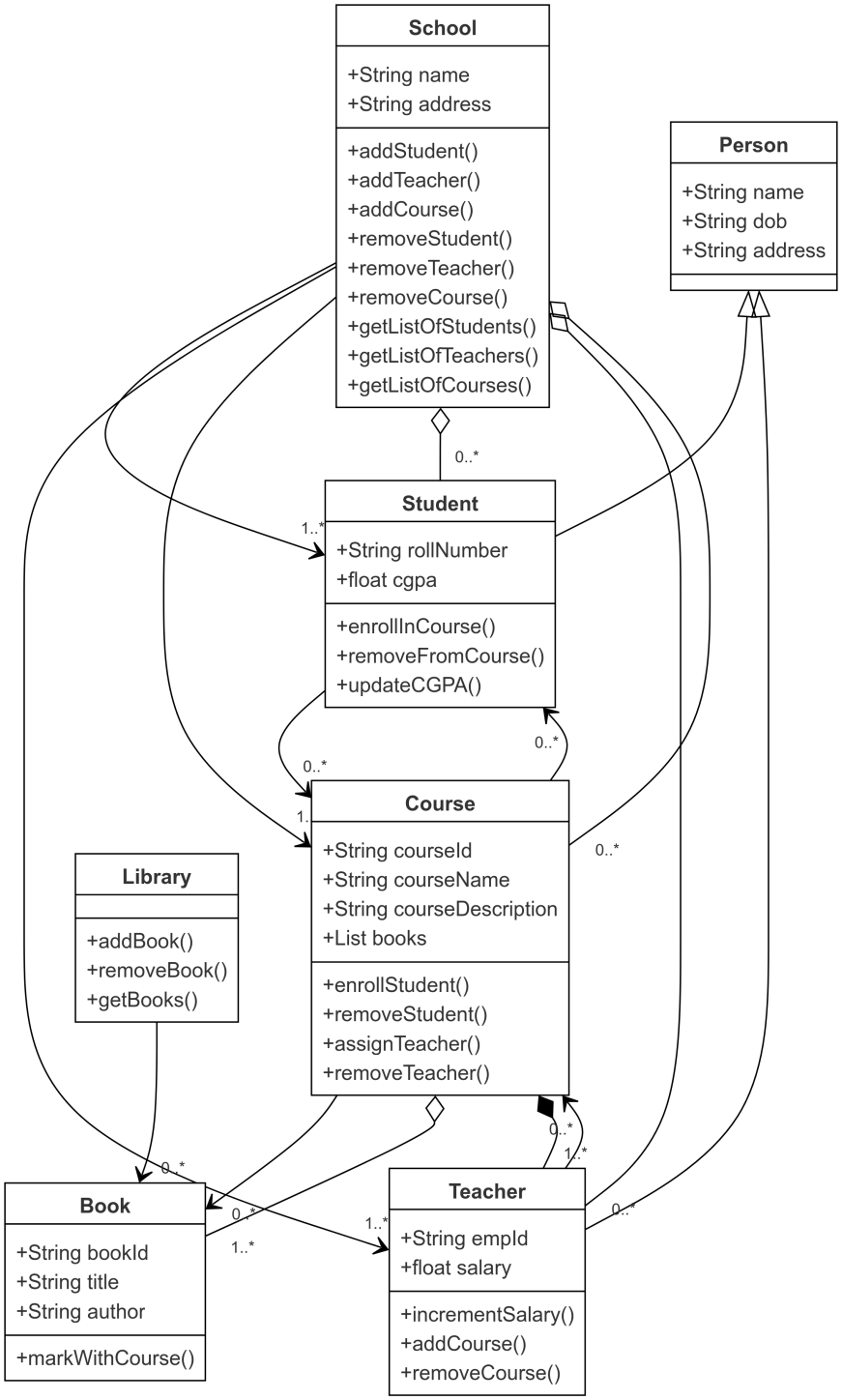

Figure 2: Class Diagram of School Management System

3.2 Class Descriptions

1. **School**
   - **Attributes:** `name`, `address`.
   - **Methods:** `addStudent()`, `addTeacher()`, `addCourse()`, `removeStudent()`, `removeTeacher()`, `removeCourse()`, `getListOfStudents()`, `getListOfTeachers()`, `getListOfCourses()`.
   - **Explanation:** Aggregates students, teachers, and courses.
2. **Person**
   - **Attributes:** `name`, `dob`, `address`.
   - **Explanation:** Base class for `Student` and `Teacher`.
3. **Student**
   - **Attributes:** `rollNumber`, `cgpa`.
   - **Methods:** `enrollInCourse()`, `removeFromCourse()`, `updateCGPA()`.
   - **Explanation:** Inherits from `Person`, represents a student.
4. **Teacher**
   - **Attributes:** `empId`, `salary`.
   - **Methods:** `incrementSalary()`, `addCourse()`, `removeCourse()`.
   - **Explanation:** Inherits from `Person`, represents a teacher.
5. **Course**
   - **Attributes:** `courseId`, `courseName`, `courseDescription`, `books`.
   - **Methods:** `enrollStudent()`, `removeStudent()`, `assignTeacher()`, `removeTeacher()`.
   - **Explanation:** Represents a course and its related entities.
6. **Book**
   - **Attributes:** `bookId`, `title`, `author`.
   - **Methods:** `markWithCourse()`.
   - **Explanation:** Represents a book in the library.
7. **Library**
   - **Attributes:** `books`.
   - **Methods:** `addBook()`, `removeBook()`, `getBooks()`.
   - **Explanation:** Represents the library's collection of books.

4 Key Design Patterns Used

In the JDBC project, several critical design patterns contribute to its structured, efficient, and maintainable architecture. These patterns—the Data Access Object (DAO) pattern, Singleton pattern, and Factory pattern—are strategically implemented to separate concerns, streamline database interactions, and ensure optimal resource management.

- **Data Access Object (DAO) Pattern:** The DAO pattern is central to this project, abstracting and encapsulating database access. Each entity, like User, Course, and LibraryResource, has its own DAO class (e.g., UserDAO, CourseDAO) for managing data operations. This pattern enforces a clear separation between business and database logic, allowing service classes to interact with entities without concern for database specifics. The DAO pattern also simplifies maintenance, as schema changes affect only the DAO classes, keeping other layers unaffected.

- **Singleton Pattern:** The Singleton pattern is used for the ConnectionPool class to ensure only one instance manages database connections throughout the application. This approach optimizes resources by reusing connections, reducing the overhead of repeatedly opening and closing them. It's especially effective in applications with frequent database access, providing centralized, scalable connection management that easily supports expansion.
- **Factory Pattern:** The Factory pattern instantiates specific DAO classes based on entity type, adding flexibility and reducing dependencies in service layers. Using a DAOFactory, the project dynamically provides DAOs (e.g., UserDAO, CourseDAO) as needed, decoupling service classes from DAO implementations. This setup makes the system adaptable, as only factory configurations need updating if database changes occur.

5 Use Case Diagrams

5.1 Use Case Diagram

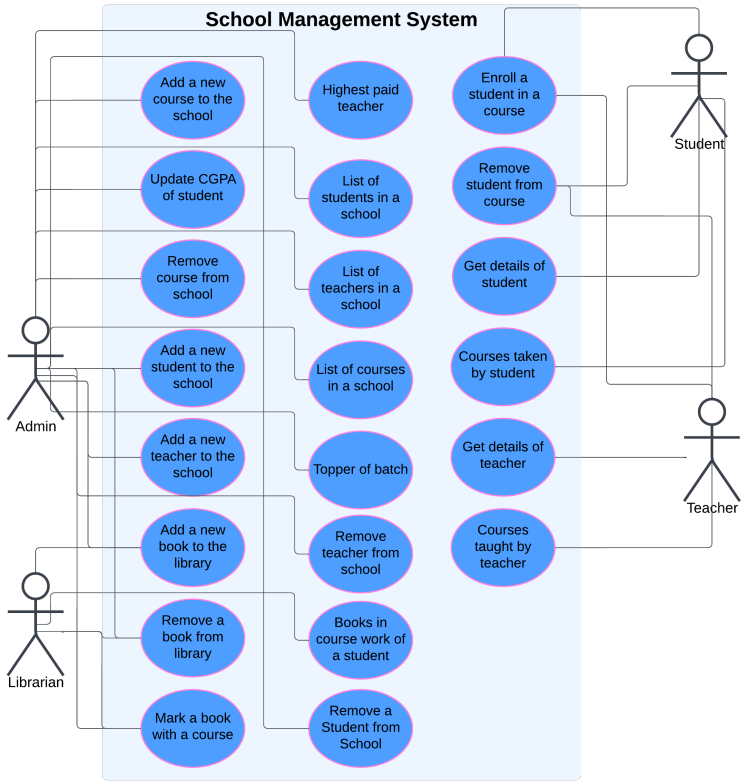

Figure 3: Use Case Diagram

5.2 Use Case Description

The **Use Case Diagram** shows the key interactions in the School Management System, highlighting the roles and functionalities for student, teacher, course, and library management.

5.2.1 Actors Description

The system has four main actors:

- **Admin:** Manages students, teachers, courses, and critical data (e.g., CGPA, salaries).
- **Student:** Enrolls in courses, checks progress, and views personal details.
- **Teacher:** Manages course enrollments and reviews course details.
- **Librarian:** Adds/removes books, links them to courses, and tracks their usage.

5.2.2 Core Use Cases

The main functionalities are: be

- **Student Management:** Admin manages students; students enroll/withdraw from courses.
- **Teacher Management:** Admin manages teachers; teachers manage course enrollments.
- **Course Management:** Admin manages courses and links books; students and teachers access course details.
- **Library Management:** Admin and librarian manage the catalog and link books to courses.

6 Sequence Diagrams

6.1 Sequence Diagram

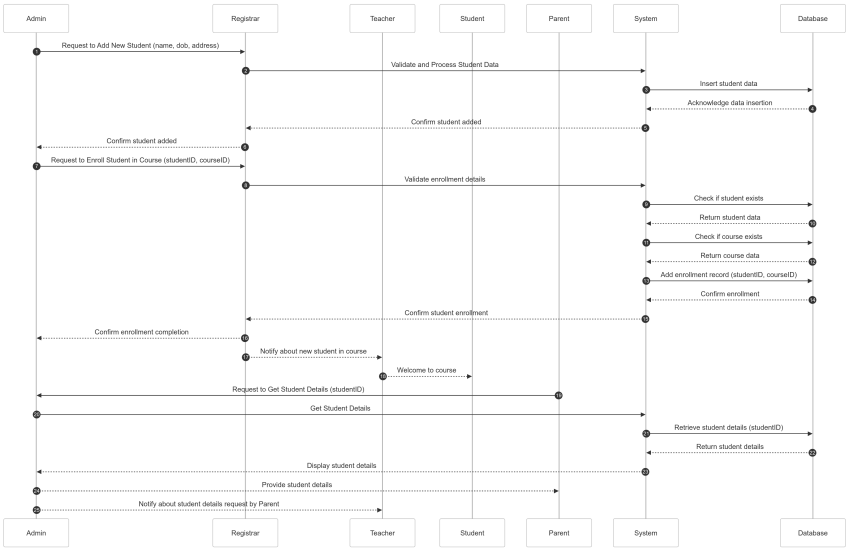

Figure 4: Sequence Diagram 1

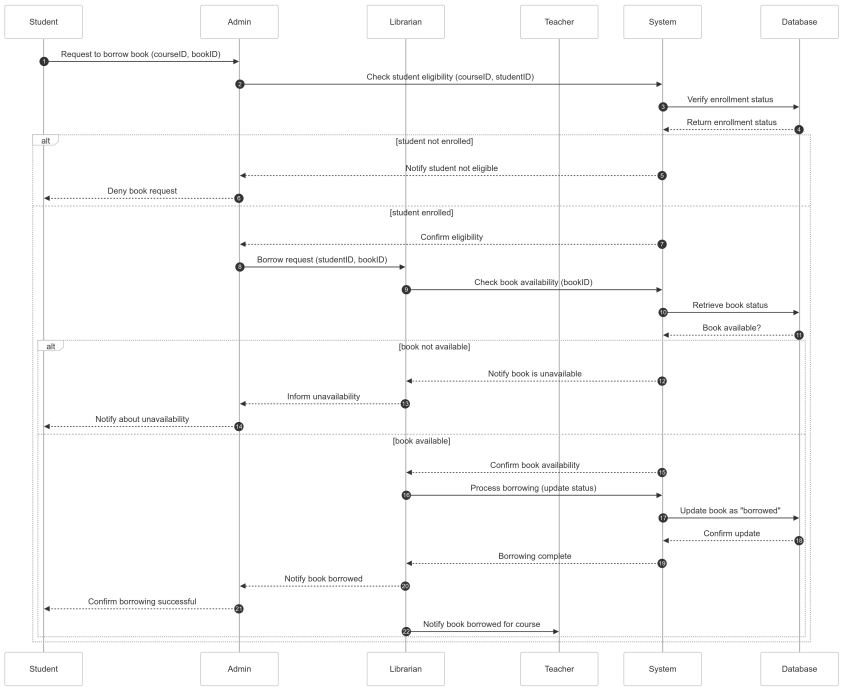

Figure 5: Sequence Diagram 2

6.2 Sequence Diagram Description

Add New Student:

1. The Admin requests the Registrar to add a student.
2. The Registrar validates the details and updates the System, which stores the data in the Database.
3. The System confirms the addition to the Registrar, who then notifies the Admin.

Enroll Student in Course:

1. The Admin instructs the Registrar to enroll a student in a course.
2. The Registrar verifies the request and forwards it to the System, which checks the Database for relevant records.
3. Upon successful enrollment, the System confirms the action to the Registrar, who then notifies the Admin and the Teacher.

Retrieve Student Details:

1. A Parent requests the Admin for details about a student.
2. The Admin queries the System, which retrieves the data from the Database.
3. The Admin provides the information to the Parent and optionally informs the Teacher.

Borrow Book for Course:

1. A Student requests a course-related book from the Admin, who forwards it to the System for eligibility verification.
2. The System checks the Database for the student's enrollment in the course.

3. If eligible, the Admin forwards the request to the Librarian, who queries the System for the book's availability.
4. If the book is available, the Librarian instructs the System to update the book's status to "borrowed" in the Database.
5. The System confirms the update, and the Librarian notifies the Admin and the Teacher.
6. The Admin confirms the borrowing action to the Student, and the Teacher is informed to maintain a record of course resources.

Key Points:

- *Clear Role Assignment:* Distinct roles for Admin, Registrar, Teacher, Librarian, and Parent ensure specific responsibilities and access control.
- *System-Database Integrity:* Each use case involves interactions between the System and the Database, ensuring data consistency and integrity.
- *Notification Flow:* Relevant parties are promptly notified of actions and state changes, emphasizing structured and efficient communication.

These sequence diagrams illustrate key processes in a structured educational system, supporting efficiency, data integrity, and role-based access.

7 Security Considerations

The School Management System implements several security measures to ensure data integrity and confidentiality:

- **User Authentication:** Users must authenticate with a valid username and password. Passwords are securely hashed using SHA-256 or bcrypt.
- **Role-Based Access Control (RBAC):** Different roles (Admin, Teacher, Student, Parent) have specific access levels to ensure users can only access relevant functionality.
- **Data Encryption:** Sensitive data, such as passwords and student information, is encrypted for safety and security.
- **Session Management:** Secure session IDs are used, with automatic session expiration to prevent hijacking.

8 Conclusion

The JDBC School Management Project provides a robust, modular system for managing student, teacher, course, and library data. By utilizing design patterns such as DAO, Singleton, and Factory, the project ensures maintainability, scalability, and efficient data handling, making it a reliable solution for academic institution management.