A PROJECT REPORT

Online Student Management System

Submitted by

Arla Srishanth(23BCS10257)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

Chandigarh University

June, 2024

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

BONAFIDE CERTIFICATE

Certified that this project report "**Online Student System**" is the bonafide work of "**Arla Srishanth" AND "Sujal Rathore"** who carried out the project work under my/our supervision.

SIGNATURE	**SIGNATURE**
BATCH HEAD	**SUPERVISOR**
Pf. Sandeep Singh Kang	**Er.Pravindra Kumar Gole**

Table of Contents

1. Introduction

1.1 Introduction to Project

In today's academic institutions, managing student data efficiently is crucial for both administrative staff and faculty members. The **Online Student Management System** is a web-based application designed to automate and streamline the management of student-related information, including admissions, courses, and payments.

Built using **Spring Framework** and **Hibernate ORM**, this project demonstrates key enterprise Java concepts such as **Dependency Injection**, **Transaction Management**, and **Object Relational Mapping (ORM)**. The system supports **Create, Read, Update, Delete (CRUD)** operations on core entities—students, courses, and payments—providing an intuitive interface for seamless data management.

The application follows a **layered architecture**, ensuring clean separation of concerns among DAO, Service, and Controller layers. The use of **Hibernate ORM** ensures efficient database interaction, while **Spring Framework** manages dependencies and transactions declaratively.

This system helps institutions maintain organized and easily retrievable student data, automate payments and refunds, and ensure data consistency and reliability.

1.2 Problem Identification

Traditional methods of managing student records involve manual paperwork or isolated digital files, leading to errors, redundancy, and inefficiency. Lack of centralized data management results in inconsistent information and time-consuming administrative work.

The problem identified is the **need for a digital solution** that enables:

- Centralized data management for students, courses, and payments.
- Automated and secure handling of transactions.
- Real-time access and modification of academic records.

1.3 Objectives

- Develop a robust and scalable **Student Management System** using Spring and Hibernate.
- Implement **CRUD functionalities** for student and course data management.
- Ensure **transaction safety** and data consistency in payments and refunds.
- Integrate **Spring Dependency Injection** for modular architecture.
- Provide clear database mappings through **Hibernate ORM annotations**.

2. Literature Review / Background Study

2.1 Existing Solutions

Existing student management systems often depend on heavy ERP software or web portals requiring extensive configuration. These are either too expensive for small institutions or lack customization.

Most traditional systems lack real-time database synchronization, flexible architecture, or reliable transaction rollback features—problems effectively solved using **Spring + Hibernate** integration.

2.2 Problem Definition

Developing a cost-effective, easily maintainable, and modular system that allows:

- Efficient management of student and course records.
- Atomic transaction handling during payments.
- Secure data consistency through ORM mapping.

2.3 Goals and Objectives

- Implement **Spring-Hibernate integration** for backend operations.
- Utilize **dependency injection** to reduce coupling between modules.
- Maintain **data integrity** using transactional annotations.
- Offer scalability, maintainability, and clean architecture.

3. System Analysis

3.1 Requirement Analysis

Functional Requirements

✓ **Student Management:** Add, edit, view, and delete student details.

✓ **Course Management:** Create and manage course records with fees and duration.

✓ **Payment Management:** Handle payments and refunds securely.

✓ **Reporting Module:** Generate student and course-based summaries.

✓ **Search and Filter:** Retrieve data efficiently through query operations.

Non-Functional Requirements

✓ **Performance:** Fast response times for CRUD operations.

✓ **Security:** Controlled access and transactional rollback for failed payments.

✓ **Reliability:** Ensures atomicity, consistency, isolation, and durability (ACID).

✓ **Scalability:** Designed to handle large datasets efficiently.

✓ **Maintainability:** Clear layered structure for easy debugging and extension.

3.2 Feasibility Study

Technical Feasibility

- Built with **Spring Framework**, **Hibernate ORM**, and **MySQL**.
- Follows an open-source stack ensuring platform independence and flexibility.
- Integrates **HikariCP** for high-performance connection pooling.

Economic Feasibility

- Uses free and open-source technologies.
- Requires minimal deployment cost for educational institutions.

Operational Feasibility

- User-friendly console interface for operation.
- Simple configuration via Java-based Spring setup.
- Minimal training required for operation and maintenance.

3.3 System Specification

Frontend Technologies:

- Java Console Application / Optional Web UI (Spring MVC).

Backend Technologies:

- **Spring Framework** for dependency injection and transaction management.

- **Hibernate ORM** for relational mapping and CRUD operations.
- **MySQL** for relational data storage.

Other Tools:

- **Maven** for build automation.
- **SLF4J + Logback** for logging.

4. System Design

4.1 Design Flow

1. User runs the application.
2. Chooses operation (Student/Course/Payment Management).
3. Inputs or updates relevant data.
4. Hibernate persists data to MySQL database.
5. Spring manages transactions and rolls back on exceptions.

4.2 User Interface Design

The UI (console-based or optional Spring MVC frontend) includes:

- Menu-driven navigation.
- Clear prompts for CRUD operations.
- Simple text-based reports for readability.

4.3 Database Design

Tables:

- **students**
    - student_id (PK)
    - name
    - email
    - course_id (FK)
    - balance
- **courses**
    - course_id (PK)
    - course_name
    - duration
    - fee
- **payments**
    - payment_id (PK)
    - student_id (FK)
    - amount
    - payment_type
    - payment_date
    - description

4.4 System Architecture

- **Layer 1: DAO Layer** – Handles direct database communication via Hibernate.
- **Layer 2: Service Layer** – Contains business logic and transaction management.
- **Layer 3: Configuration Layer** – Sets up Spring beans and dependencies.

5. Implementation

5.1 Technology Stack

- **Backend:** Spring Framework, Hibernate ORM
- **Database:** MySQL
- **Build Tool:** Maven
- **Logging:** SLF4J + Logback
- **Language:** Java

5.2 Service Layer Implementation

Implements business logic for adding and managing students, courses, and payments. Uses @Transactional for atomic operations ensuring rollback on failure.

5.3 DAO Layer Implementation

Implements CRUD using Hibernate APIs such as:

```
session.save(student);
session.get(Student.class, id);
session.update(student);
session.delete(student);
```

5.4 Integration

Spring handles object injection while Hibernate ensures persistence. Both integrate through **SessionFactory** and **HibernateTransactionManager** configured in AppConfig.java.

6. Testing and Validation

6.1 Testing Strategies

- **Unit Testing:** DAO and Service layer functions tested individually.
- **Integration Testing:** Spring-Hibernate configuration validated with real data.
- **User Acceptance Testing:** Checked for smooth execution and data consistency.

6.2 Test Cases and Results

Test Case	Description	Result
Add Student	Insert student record	Passed
Update Course	Modify fee/duration	Passed
Payment Transaction	Verify rollback on failure	Passed
Delete Record	Ensure referential integrity	Passed
Query Student Report	Retrieve by course	Passed

6.3 Validation

All modules validated through test scenarios ensuring correctness, speed, and consistency.

---

7. Results and Discussion

The Student Management System performed efficiently across all tested modules.

- CRUD operations executed without error.
- Transactions rolled back correctly on exceptions.
- Data persisted and retrieved accurately from MySQL.
- Application demonstrated modular, reusable, and scalable architecture.

---

8. Conclusion and Future Work

8.1 Conclusion

The **Student Management System** successfully integrates **Spring Framework** and **Hibernate ORM** to deliver a modular, secure, and efficient backend solution for student data management. It achieves all objectives including CRUD operations, transaction handling, and seamless data mapping.

8.2 Future Scope

- Add **RESTful API endpoints** for web integration.
- Build a **React or Angular frontend**.

- Implement **Spring Security** for role-based access.
- Add **report export options (PDF/Excel)**.
- Integrate **email notifications** and **logging dashboards**.

---

9. References

- Spring Framework Documentation
- Hibernate ORM Documentation
- MySQL Reference Manual
- Spring Transaction Management Guide

---

10. Appendix

- Sample schema.sql file.
- AppConfig.java configuration setup.
- Example StudentDAO implementation.
- Log output screenshots.