VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

SCHOOL OF COMPUTER SCIENCE AND TECHNOLOGY

STUDENT MANAGEMENT SYSTEM

Course: Web Application Developemnet

Major: Information Technology

Group members:

La Văn Phú – ITITIU21282

Trần Khánh Tài - ITITIU21300

Nguyễn Di Niên - ITITIU21272

Academic Year: 2021–2025

Table Of Contents

1. Introduction

1.1. Background and Motivation

The rapid advancement of information technology has led to significant changes in the way educational institutions manage academic and administrative activities. Traditional student management methods, which often rely on manual record-keeping or isolated software systems, are inefficient and prone to errors. As the number of students, teachers, and courses increases, managing academic data such as enrollment, grades, and attendance becomes more complex. Therefore, there is a strong demand for an integrated web-based system that can centralize data management, improve operational efficiency, and support multiple user roles within an educational environment.

1.2. Problem Statement

Many educational institutions still face challenges in managing student information effectively. Common problems include duplicated data, lack of real-time updates, limited access control, and difficulty in tracking academic performance. Additionally, systems without proper role-based authorization may expose sensitive data to unauthorized users. These limitations highlight the need for a secure, scalable, and well-structured student management system that clearly separates responsibilities among administrators, teachers, and students.

1.3. Problem Statement

The main objective of this project is to design and develop a web-based Student Management System that supports efficient academic management through a centralized platform. The system aims to provide role-based access control for Admin, Teacher, and Student users, ensure secure authentication and authorization, and facilitate key functions such as user management, course management, student enrollment, and grade management. By applying modern web technologies and architectural patterns, the system seeks to enhance maintainability, usability, and data integrity.

1.4. Objectives of the System

The main objective of this project is to design and develop a web-based Student Management System that supports efficient academic management through a centralized platform. The system aims to provide role-based access control for Admin, Teacher, and Student users, ensure secure authentication and authorization, and facilitate key functions such as user management, course management, student enrollment, and grade management. By applying modern web technologies and architectural patterns, the system seeks to enhance maintainability, usability, and data integrity.

1.5. Scope of the Project

This project focuses on developing the core functionalities of a Student Management System for educational institutions. The system includes user authentication, role-based access control, management of students, teachers, courses, enrollments, and grades. The application is implemented as a web-based system using the Spring Boot framework and a relational database for data persistence. Advanced features such as data analytics, artificial intelligence, and mobile application support are beyond the scope of this project but may be considered for future development.

1.6. Structure of the Report

This report is organized into several sections to present the system in a structured manner. The Introduction provides an overview of the project background and objectives. The Theoretical Background section discusses the technologies and architectural concepts used in the system. The System Analysis section presents the analysis of system requirements, user roles, and workflows. The System Design section describes the system architecture and database design. The Implementation section explains how the system is developed and deployed. Finally, the Results and Evaluation section assesses the system performance, followed by the Conclusion and Future Work section, which summarizes the project and proposes potential enhancements.

2. Theoretical Background

2.1. Overview of Web-Based Information Systems

A web-based information system is an application that operates on a client–server architecture, where users interact with the system through a web browser while data processing and storage are handled on a centralized server. Such systems enable real-time data access, centralized management, and scalability. In the context of education, web-based information systems are widely used to manage students, teachers, courses, and academic records efficiently while ensuring data consistency and accessibility across different user roles.

2.2. Spring Boot Framework

Spring Boot is a Java-based framework designed to simplify the development of enterprise-level web applications. It provides built-in configurations, embedded servers, and dependency management, allowing developers to focus on business logic rather than infrastructure setup. Spring Boot supports rapid application development and integrates seamlessly with other Spring modules such as Spring MVC, Spring Data JPA, and Spring Security. These features make Spring Boot suitable for developing scalable, maintainable, and secure student management systems.

2.3. MVC Architectural Pattern

The Model–View–Controller (MVC) pattern is a software architectural pattern that separates an application into three main components: Model, View, and Controller. The Model represents the application data and business rules, the View is responsible for presenting data to users, and the Controller handles user requests and coordinates interactions between the Model and the View. By applying the MVC pattern, the system achieves better separation of concerns, improved code readability, and easier maintenance.

2.4. Layered Architecture in Enterprise Applications

In addition to the MVC pattern, modern Spring Boot applications commonly adopt a layered architecture. This architecture separates the system into distinct layers, including the Controller layer, Service layer, Repository layer, and Entity layer. Each layer has a specific responsibility, such as handling HTTP requests, implementing business logic, accessing the database, or mapping database tables. This separation enhances modularity, testability, and scalability, which are essential for enterprise-level systems.

2.5. Role-Based Access Control (RBAC)

Role-Based Access Control (RBAC) is a security mechanism that restricts system access based on user roles. Instead of assigning permissions to individual users, permissions are associated with roles such as Admin, Teacher, and Student. In a student management system, RBAC ensures that administrators can manage users and system data, teachers can manage classes and grades, and students can only view their personal academic information. RBAC improves system security, reduces unauthorized access, and simplifies permission management.

2.6. Database Management Systems

A relational database management system (RDBMS) stores data in structured tables with defined relationships between them. RDBMSs such as MySQL or PostgreSQL support data integrity, consistency, and efficient querying through structured query language (SQL). In this project, a relational database is used to store information related to users, students, teachers, courses, enrollments, and grades. The use of relational databases ensures reliable data storage and supports complex relationships within the student management system.

3. System Analysis

3.1. Stakeholders and User Roles

The Student Management System involves three primary stakeholders: Admin, Teacher, and Student. Each stakeholder represents a distinct user role with specific responsibilities and access permissions.

The Admin is responsible for managing the overall system, including user accounts, roles, and system-wide configurations. Teachers represent academic staff who manage teaching activities such as classes, grading, and attendance. Students are the end users who interact with the system to view academic information and manage their learning-related activities.

Clear identification of stakeholders and user roles ensures proper access control and supports secure and efficient system operation.

3.2. Functional Requirements

Functional requirements define what the system is expected to do.

The system must allow Admin users to create, update, and manage accounts for Teachers and Students. Admins must also be able to assign roles and permissions and publish system announcements.

Teachers must be able to view assigned classes and student lists, input and update grades, manage quizzes, and record student attendance.

Students must be able to access a personal dashboard, view schedules and notifications, check grades and attendance status, and view tuition-related information with basic payment simulation functionality.

3.3. Non-Functional Requirements

The system must ensure security through authentication and role-based authorization. It must protect sensitive academic and personal data from unauthorized access. The system should provide acceptable performance and responsiveness when handling concurrent users. In addition, the system must be maintainable, scalable, and reliable to support future enhancements and long-term use.

3.4. Main System Entities

The Student Management System includes core entities that support administrative, teaching, and learning activities. The User entity stores authentication data, while the Role entity defines access permissions for Admin, Teacher, and Student.

Academic structures are represented by Course and Classroom. Student enrollment in courses is managed through the Enrollment entity. Teachers manage academic performance using the Grade, Quiz, Question, and QuizResult entities. Student participation is recorded through the Attendance entity, with attendance states defined by AttendanceStatus.

System communication is handled by the Notification entity. Tuition-related information is managed through the Transaction entity. Secure authentication sessions are supported by the RefreshToken entity.

These entities form the foundation of the system's data model and support both teaching and learning processes.

3.5. Use Case Analysis

Use case analysis describes how each user role interacts with the system to achieve specific goals. Admin users interact with the system to manage user accounts, assign roles, and publish system announcements. Teacher users interact with the system to manage assigned classes, input and update grades, create quizzes, and record student attendance. Student users interact with the system to view personal information, academic results, attendance records, and tuition-related information.

3.6. Use Case Diagram

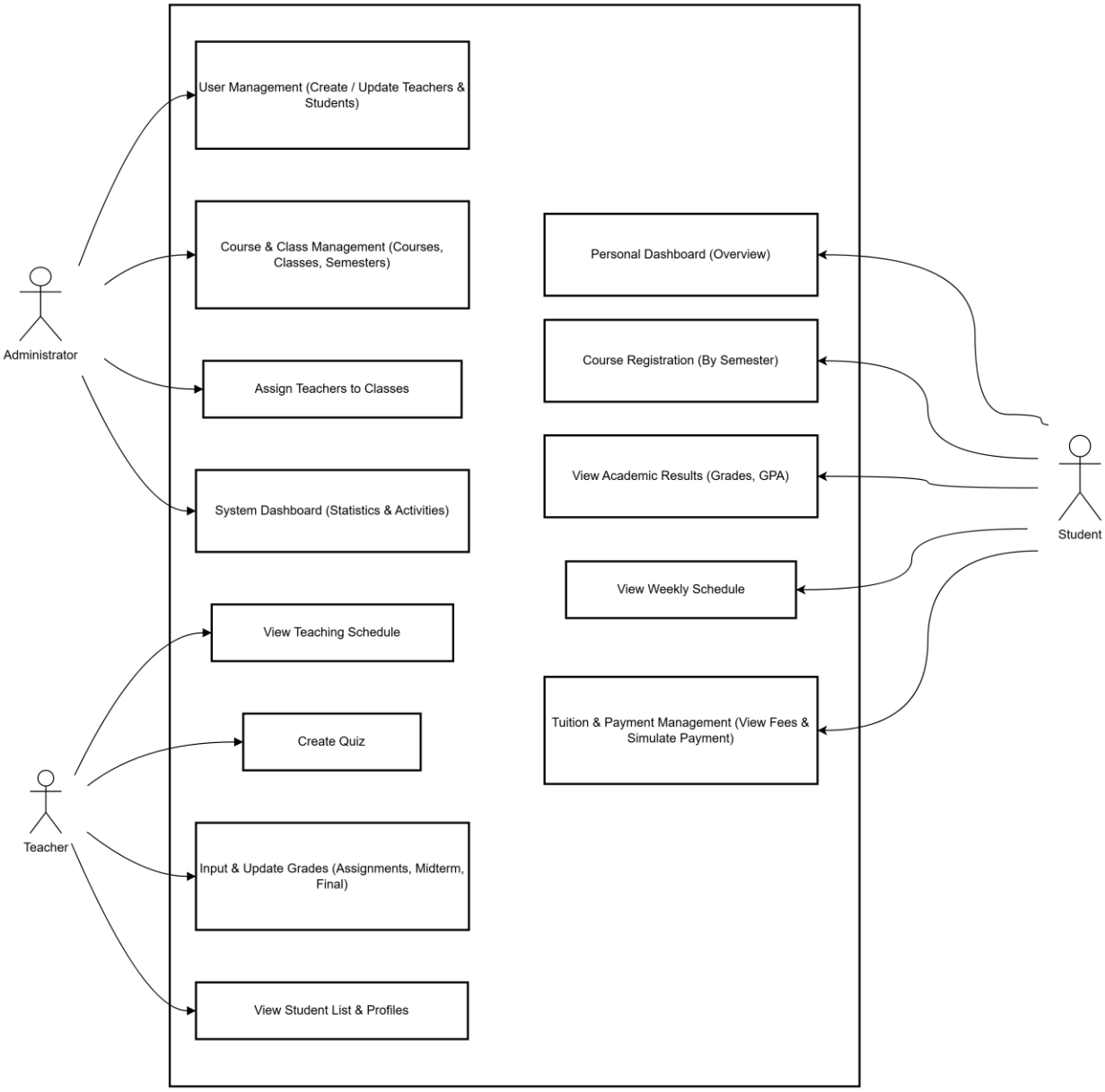


Figure 1. Use case diagram

3.7. Data Flow Analysis

This section details the movement of information through the system, describing how inputs are processed, validated, and stored across the core modules: Authentication, Attendance, Quiz Assessment, and Grading.

3.7.1. Authentication and Authorization Flow

The security module ensures that only authorized users access specific resources.

1. **Input:** The user enters their username and password on the login page.
2. **Processing:**
    a. The request is intercepted by **Spring Security**.
    b. The `CustomUserDetailsService` queries the `users` table in the database to retrieve the account details.
    c. The system compares the input password with the stored hashed password using the **BCrypt algorithm**.
3. **Output:**
    a. If valid, an HTTP Session is created, and the user is redirected to their specific Dashboard based on their `Role (TEACHER or STUDENT).`
    b. If invalid, an error message is returned to the login view.

3.7.2. Attendance Management Flow (Smart Merge Strategy)

To prevent data redundancy and ensure historical accuracy, the attendance flow utilizes a data merging technique.

1. **Data Retrieval (Parallel Fetching):**
    a. When a teacher selects a class and a date, the client-side sends two simultaneous requests: one to fetch the **Student List** (from `Enrollments`) and another to fetch **Historical Attendance** (from `Attendance` table) for that specific date.
    b. The frontend merges these two datasets. If a student already has an attendance record, their status is pre-filled and locked (disabled) to prevent accidental overwrites.
2. **Submission:**
    a. The teacher selects the status (`PRESENT, ABSENT, LATE`) for the remaining students and submits the form.
3. **Storage:**
    a. The backend accepts the list of `AttendanceDTO` objects.
    b. The system iterates through the list and saves new records to the `attendances` table linked to the specific `Classroom` and `Student`.

3.7.3. Online Quiz and Auto-Grading Flow

This module handles the lifecycle of an online exam, prioritizing data integrity and security.

1. **Quiz Retrieval (Security Layer):**
    a. The student requests to take a specific Quiz.

b. The server retrieves the Quiz and its Questions from the database.
c. **Crucial Step:** Before sending the data to the client (browser), the system filters out the `correctAnswer` field from the Data Transfer Object (DTO). This prevents students from inspecting the source code to find answers.

2. **Submission and Grading:**
   a. The student submits a list of selected options.
   b. The `StudentQuizService` retrieves the *actual* correct answers from the database.
   c. The system compares the submitted answers against the database records.
   d. The score is calculated immediately (e.g., *Correct Answers / Total Questions * 100*).
3. **Result Storage:**
   a. The final score and timestamp are saved into the `quiz_results` table.
   b. The calculated score is returned to the student immediately for real-time feedback.

3.7.4. Gradebook Management Flow

This flow manages the calculation and storage of academic performance.

1. **Input:** The teacher inputs raw scores for specific criteria: *In-class participation*, *Mid-term exams*, and *Final exams*.
2. **Processing (Auto-Calculation):**
   a. The system receives the `GradeUpdateDTO`.
   b. The backend automatically calculates the **Total Grade** based on the pre-defined weighted formula (e.g., 30% In-class + 30% Mid-term + 40% Final).
   c. `Total = (In-class * 0.3) + (Mid-term * 0.3) + (Final * 0.4)`
3. **Output:**
   a. The updated grades are saved to the `enrollments` table.
   c. The new Total Grade becomes visible on the Student's dashboard immediately.

3.8. Data Flow Diagram (DFD)

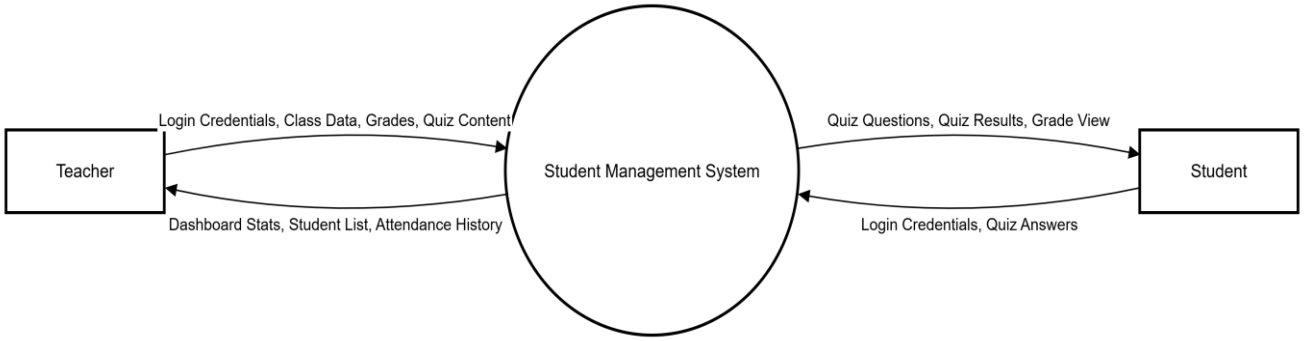


Figure 2. Data Flow Diagram Level 0

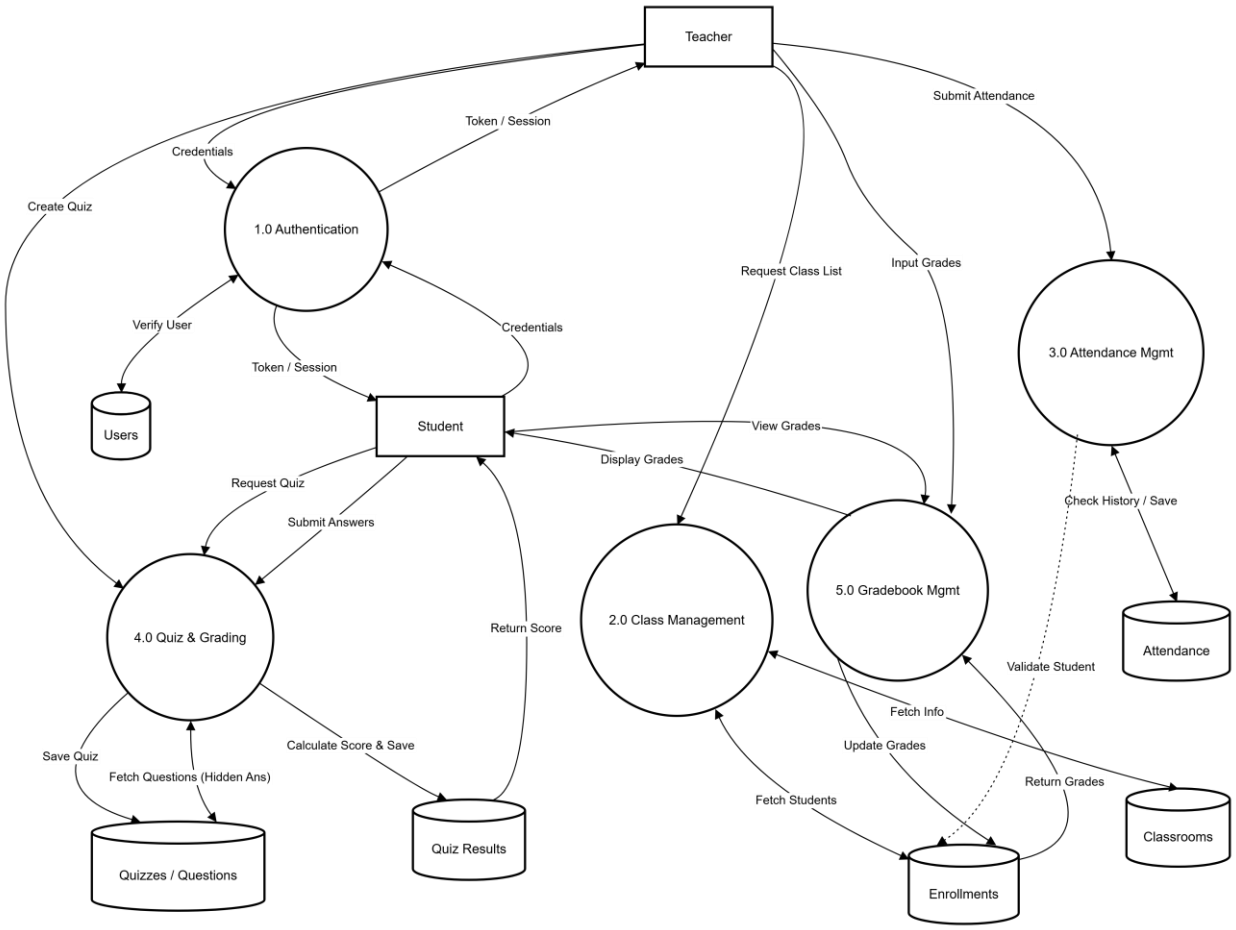


Figure 3. Data Flow Diagram Level 1 (Detailed System Decomposition)

3.9. System Workflow Analysis

This section describes the operational logic and sequential activities performed by the system actors (Teacher and Student) to achieve specific functional goals. The workflows are categorized by the core modules of the application.

3.9.1. Access Control and Role-Based Routing

The workflow begins with system entry, where the application determines the user's privileges.

1. Initiation: The user accesses the web application and is presented with the Login Interface.
2. Credential Submission: The user submits their `username` and `password`.
3. Verification: The system validates the credentials against the database using Spring Security.
4. Role Evaluation:
    a. If the user has the `ROLE_TEACHER` authority, the system redirects them to the /teacher/dashboard.
    b. If the user has the `ROLE_STUDENT` authority, the system redirects them to the /student/dashboard.
5. Session Management: An authenticated session is established, persisting until the user triggers the "Logout" function or the session expires.

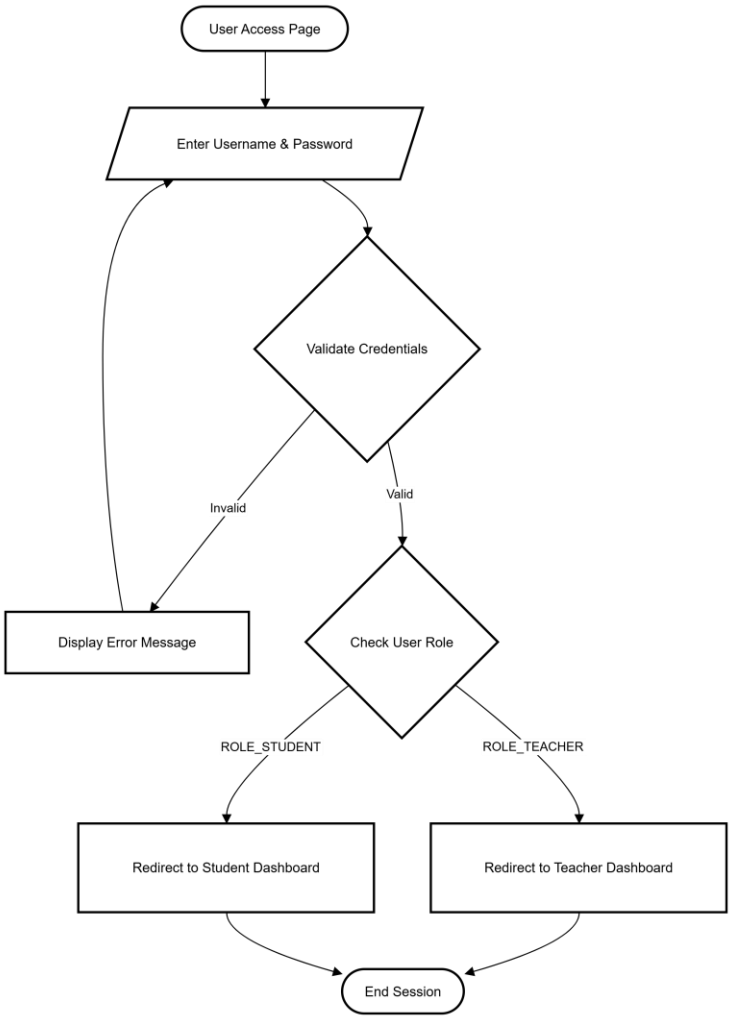


Figure 4: User Login & Role-Based Routing Activity Diagram

3.9.2. Attendance Recording Workflow

This workflow ensures accurate tracking of student presence while preserving historical data integrity.

1. Class Selection: The Teacher navigates to the "My Classes" section and selects a specific classroom.
2. History Check: Upon loading the Attendance Interface, the system performs a parallel check:
    a. It retrieves the full list of enrolled students.
    b. It queries for any existing attendance records for the current date.

3. Interface Rendering: The system renders the student list. If a student has already been marked (e.g., from a previous session on the same day), their status defaults to the existing value and the input field is disabled (locked).
4. Data Entry: The Teacher assigns a status (`Present`, `Absent`, or `Late`) to the remaining unlocked students.
5. Persistence: The Teacher submits the form. The system iterates through the valid entries and updates the `Attendance` table in the database.

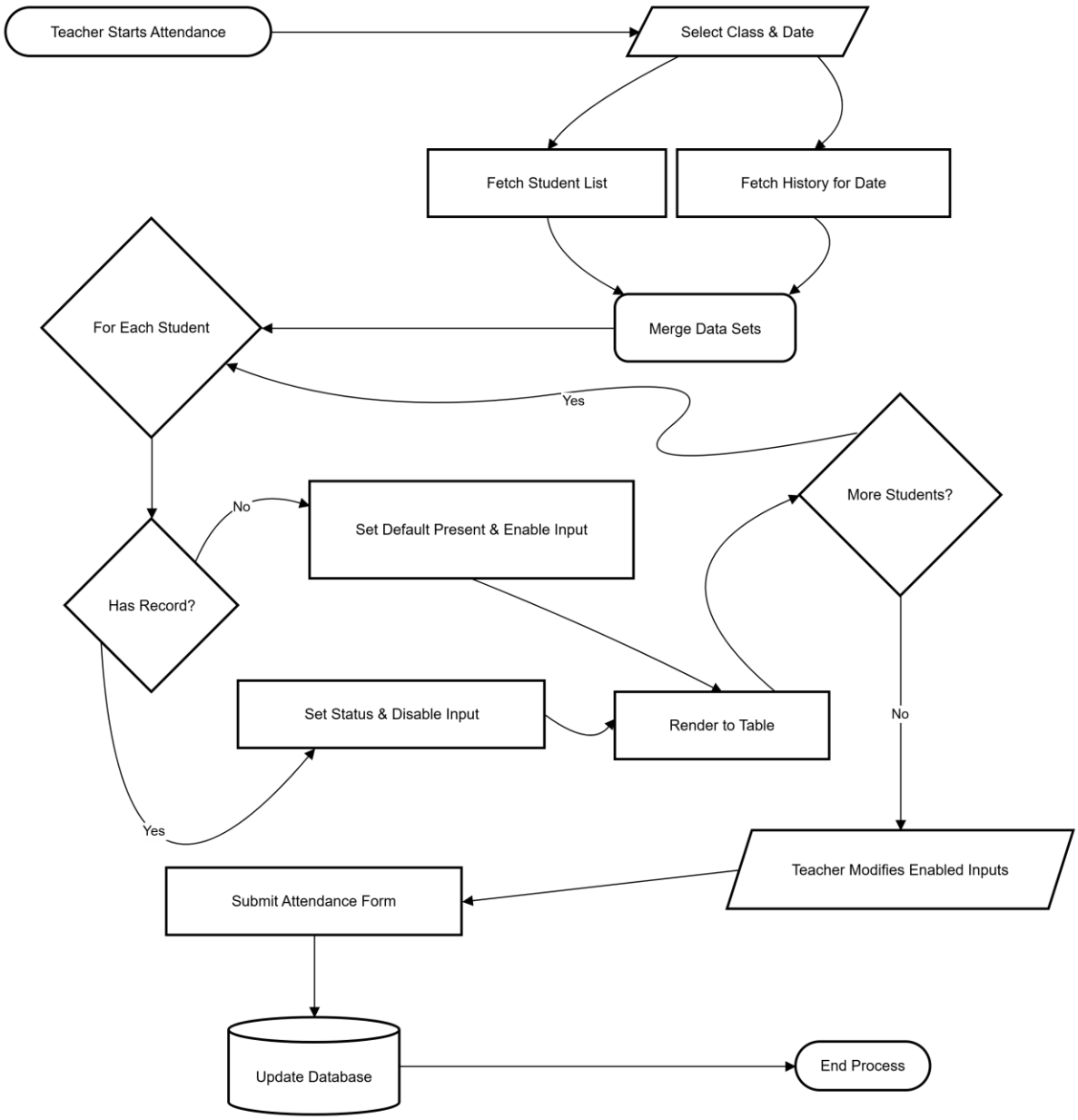


Figure 5: Attendance Tracking with History Merge Logic

3.9.3. Quiz Lifecycle and Auto-Grading Workflow

The examination workflow is divided into two phases: Creation (Teacher) and Execution (Student).

Phase A: Quiz Creation (Teacher)

1. The Teacher initiates the "Create Quiz" process.
2. The Teacher defines the Quiz metadata (Title, Description) and links it to a specific Class.
3. The Teacher inputs questions and defines four options (A, B, C, D) per question, marking one as the `correctAnswer`.
4. The system saves the Quiz and Questions to the database.

Phase B: Quiz Execution and Grading (Student)

1. **Retrieval:** The Student selects a quiz from the "My Quizzes" list.
2. **Sanitization:** The system retrieves the questions but removes the `correctAnswer` attribute from the data payload before rendering the interface to the Student.
3. **Submission:** The Student selects answers and submits the exam.
4. **Auto-Grading:**
    a. The system retrieves the hidden correct answers from the database.
    b. It compares the Student's submission against the key.
    c. It calculates the score percentage.
5. **Feedback:** The system stores the result in the `QuizResult` table and immediately displays the score to the Student via an alert modal.

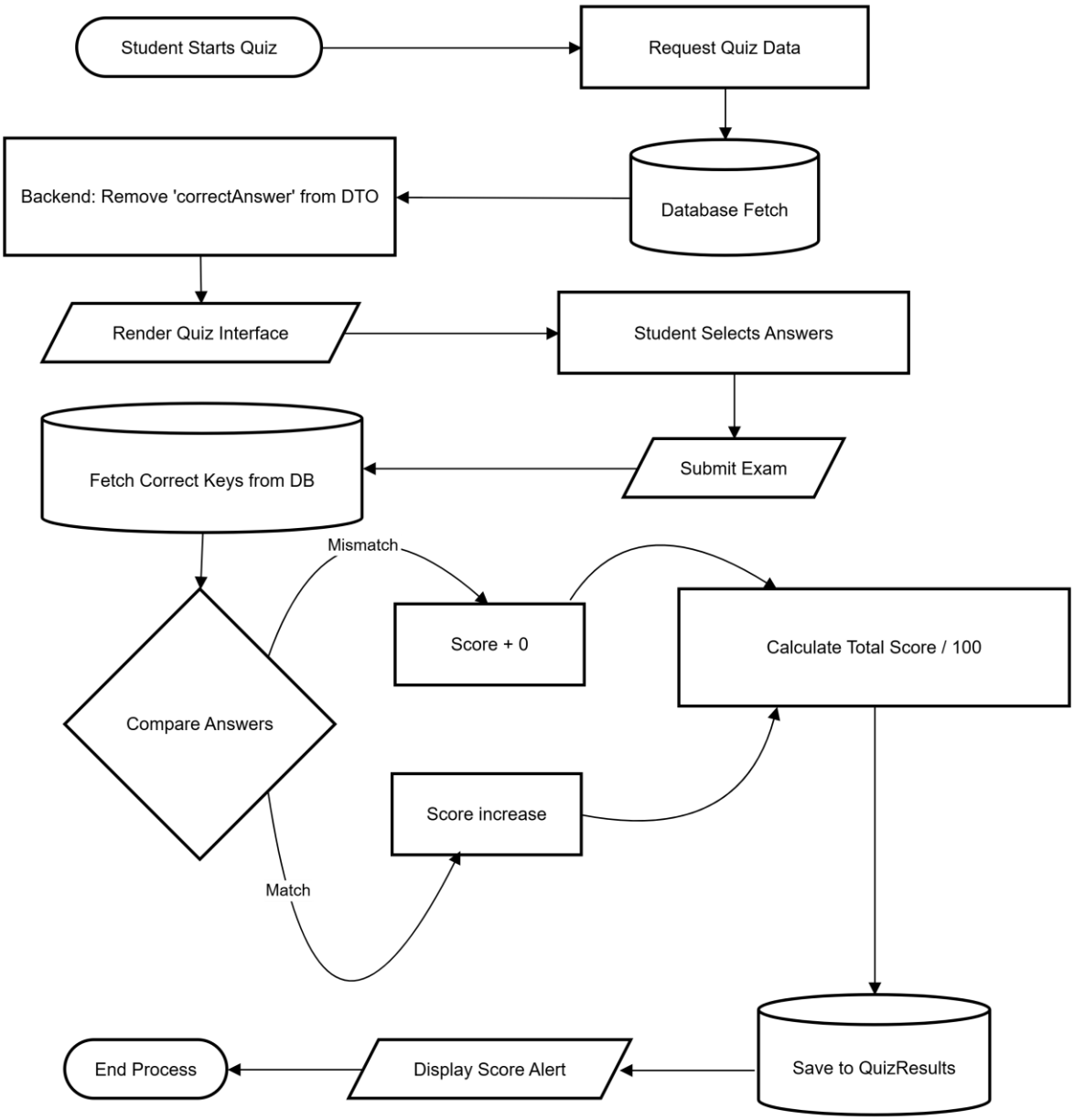


Figure 6: Quiz Submission & Auto-Grading Logic

3.6.4. Academic Grading Workflow

This workflow manages the calculation of the final course grade based on weighted components.

1. **Retrieval:** The Teacher accesses the "Grades" section for a specific class. The system loads the current gradebook.
2. **Input:** The Teacher enters or updates scores for three specific components:

a. *In-class Participation*
b. *Mid-term Exam*
c. *Final Exam*

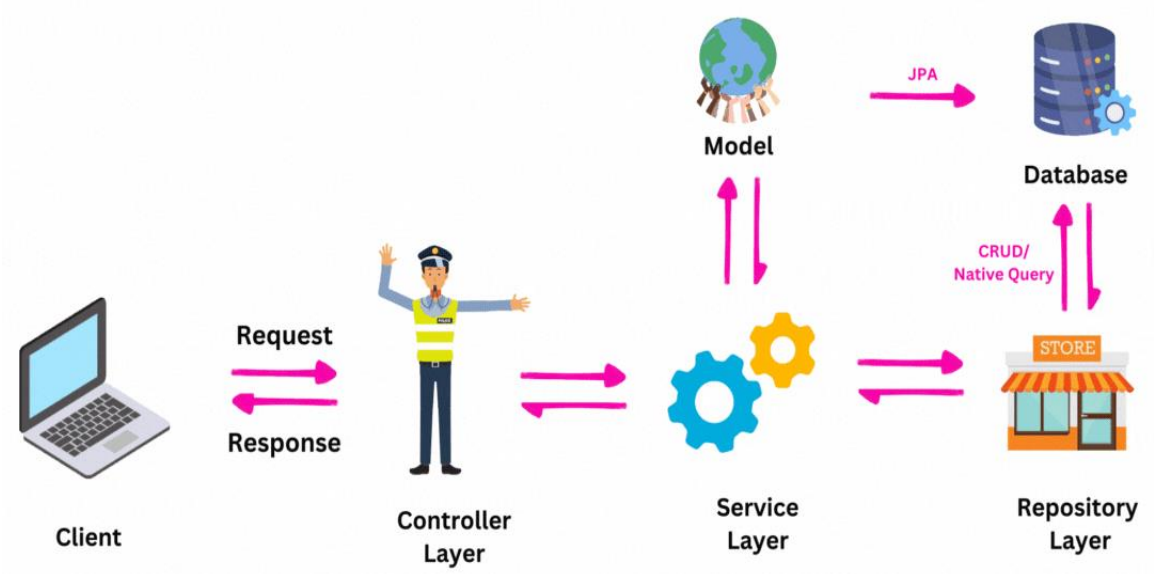
3. Calculation: Upon submission, the system applies the weighted formula:

$$Total = (InClass * 0.3) + (MidTerm * 0.3) + (Final * 0.4)$$

4. **Update:** The calculated `Total Grade` is saved to the `Enrollment` entity.
5. **Visibility:** The updated total is immediately reflected on the Student's dashboard under "Course Progress."

4. System Design

4.1. Overall System Architecture



The Student Management System is designed using a layered architecture combined with the MVC pattern. The system follows a client–server model, where users (Admin, Teacher, and Student) access the application through a web browser, and requests are processed by a Spring Boot backend.
The backend is structured into Controller, Service, and Repository layers. The Controller layer handles HTTP requests and routes them to the appropriate services. The Service layer contains the core business logic, while the Repository layer manages data access using Spring Data JPA. Data is stored in a relational database and mapped through entity classes.

A security layer based on Spring Security and JWT is integrated across all layers to handle authentication and role-based authorization. This architecture ensures clear separation of concerns, maintainability, and scalability of the system.

4.2. Controller Layer Design

The Controller layer is responsible for handling HTTP requests from users and managing request routing within the system. It acts as the entry point between the client interface and the backend application.

Controllers are organized according to user roles and system functionalities, including AdminController, TeacherController, StudentController, and AuthController. Each controller receives user requests, performs basic input validation, and delegates processing tasks to the corresponding Service layer.

The Controller layer does not contain business logic. Its primary role is to coordinate the flow of data between the client and the Service layer, and to return appropriate responses or views based on the user's role and request**.**

4.3. Service Layer Design

The Service layer contains the core business logic of the Student Management System. It is responsible for processing system rules, validating data, and coordinating operations between the Controller and Repository layers.

Services handle key functions such as user management, course enrollment, grade management, quiz processing, and attendance tracking. This layer ensures that all business constraints are enforced before data is stored or retrieved from the database.

By isolating business logic within the Service layer, the system achieves better modularity, easier maintenance, and improved scalability.

4.4. Repository Layer Design

The Repository layer is responsible for direct interaction with the database. It handles data persistence and retrieval using Spring Data JPA, ensuring efficient and consistent access to stored information.

Each repository corresponds to a specific entity in the system, such as UserRepository, CourseRepository, EnrollmentRepository, GradeRepository, QuizRepository, AttendanceRepository, and TransactionRepository. These repositories provide standard CRUD operations as well as custom query methods when needed.

By isolating data access logic in the Repository layer, the system maintains a clear separation of concerns, improves code readability, and simplifies database management.

4.5. Security Design

The system applies a security mechanism based on Spring Security combined with JWT (JSON Web Token) authentication to protect system resources and enforce role-based access control.

User authentication is handled using a custom user details implementation (CustomUserDetails), which loads user information and roles from the database. When a user attempts to access protected resources, the system validates the JWT through a dedicated authentication filter (JwtAuthenticationFilter). Token generation and validation are managed by the JwtTokenProvider.

Unauthorized access attempts are handled by a custom authentication entry point (JwtAuthenticationEntryPoint), ensuring proper error handling and secure responses. Security rules and access policies are configured centrally within the SecurityConfig class.

This security design ensures that only authenticated users with appropriate roles can access system functionalities, thereby maintaining data confidentiality and system integrity.

4.6. Database Design

The system utilizes **MySQL**, a relational database management system, to ensure data integrity and persistence. The database schema is designed according to the Third Normal Form (3NF) to minimize redundancy.

4.6.1. Entity-Relationship Diagram (ERD)

The following diagram illustrates the logical structure of the database, highlighting the entities, their attributes, and the relationships between them.

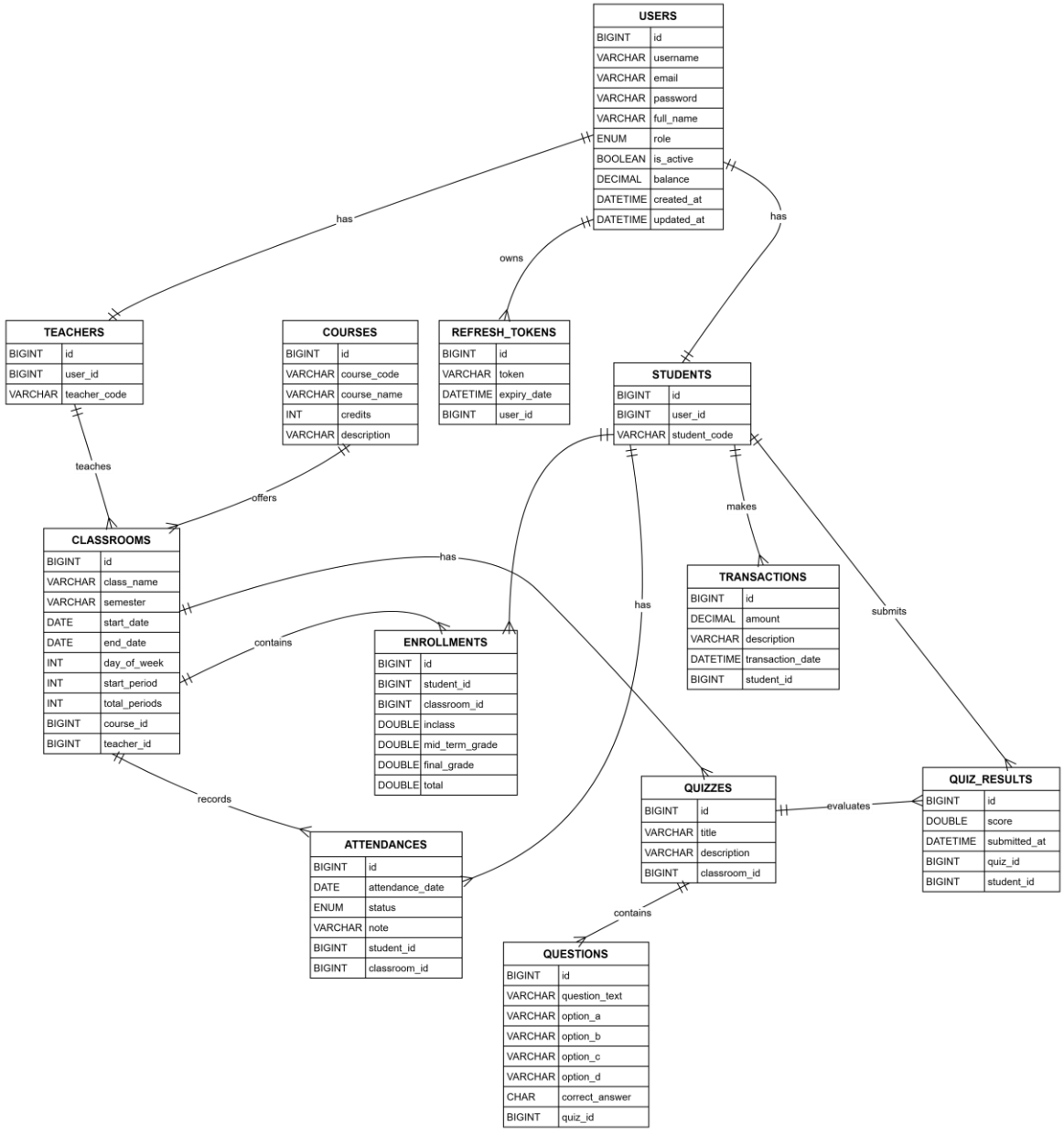
4.6.2. Data Dictionary

The database consists of the following primary tables:

1. **`users`**: Stores authentication credentials and personal information for all system actors.
    a. **Primary Key:** `id`
    b. **Attributes:** `username`, `password` (hashed), `full_name`, `email`.
    c. **Relationships:** One-to-Many with `classrooms` (as Teacher), One-to-Many with `enrollments` (as Student).
2. **`roles`**: Defines the access authorities within the system (e.g., `ROLE_TEACHER`, `ROLE_STUDENT`).
    a. **Relationship:** Many-to-Many with `users` (managed via a join table `user_roles`).
3. **`classrooms`**: Represents the academic classes created by teachers.
    a. **Primary Key:** `class_id`
    b. **Foreign Key:** `teacher_id` (References `users`).
    c. **Attributes:** `class_name`, `course_name`, `room_number`, `semester`.
4. **`enrollments`**: A junction table resolving the Many-to-Many relationship between Students and Classrooms. It also serves as the Gradebook.
    a. **Primary Key:** `id`
    b. **Foreign Keys:** `student_id`, `class_id`.

    c. **Attributes:** `in_class_score`, `mid_term_grade`, `final_grade`, `total_grade`.
5. **`quizzes`**: Stores assessment metadata created by teachers.
    a. **Primary Key:** `id`
    b. **Foreign Key:** `class_id` (The class this quiz belongs to).
    c. **Attributes:** `title`, `description`.
6. **`questions`**: Stores the content of the quizzes.
    a. **Primary Key:** `id`
    b. **Foreign Key:** `quiz_id`.
    c. **Attributes:** `question_text`, `option_a`, `option_b`, `option_c`, `option_d`, `correct_answer` (Encrypted/Hidden in DTO).
7. **`quiz_results`**: Stores the history of students' attempts and scores.
    a. **Foreign Keys:** `student_id`, `quiz_id`.
    b. **Attributes:** `score`, `submitted_at`.
8. **`attendances`**: Records the daily attendance status of students.
    a. **Foreign Keys:** `student_id`, `class_id`.
    b. **Attributes:** `date`, `status` (PRESENT, ABSENT, LATE), `note`.

4.7. Entity Relationship Diagram (ERD)



4.8. API Design

The Student Management System (SMS) is built upon a RESTful Architecture. The backend exposes a set of HTTP endpoints that the frontend client (Thymeleaf/JavaScript) consumes to perform CRUD operations. All API responses are formatted in JSON (JavaScript Object Notation), and access is secured via JWT (JSON Web Tokens).

Below is the comprehensive list of API endpoints categorized by functional modules.

4.8.1. Authentication Module

This module handles user identity verification and token generation. These endpoints are public (permit-all) to allow initial access.

HTTP Method	Endpoint URL	Description	Request Body
POST	/api/auth/login	Authenticates user credentials and returns a JWT Access Token.	`LoginRequestDTO` (username, password)

4.8.2. Teacher Core Module

These endpoints are secured and require the authority `ROLE_TEACHER`. They manage classroom data, student attendance, and academic grading.

HTTP Method	Endpoint URL	Description	Functionality
GET	/api/teacher/dashboard	Retrieves aggregate statistics (Total Classes, Students, Quizzes).	Dashboard Visualization
GET	/api/teacher/classes	Fetches the list of classes assigned to the logged-in teacher.	Class Management
GET	/api/teacher/attendance/history	Checks if attendance has already been taken for a specific Class and Date.	Attendance Validation
POST	/api/teacher/attendance	Saves or updates attendance records for students.	Attendance Tracking
POST	/api/grades/update	Updates academic scores (In-class, Midterm, Final) for a list of students.	Gradebook Management

4.8.3. Quiz & Assessment Management (Teacher)

Allows teachers to create and modify assessments.

HTTP Method	Endpoint URL	Description	Payload
GET	/api/teacher/quizzes	Lists all quizzes created by the teacher.	N/A
POST	/api/teacher/quizzes	Creates a new quiz with associated questions and correct answers.	QuizRequestDTO
GET	/api/teacher/quizzes/{id}	Retrieves full details of a specific quiz (including correct answers) for editing	N/A
PUT	`/api/teacher/quizzes/{id}`	Updates the title, description, and questions of an existing quiz.	QuizRequestDTO

4.8.4. Student Module

These endpoints are secured and require the authority `ROLE_STUDENT`. They allow students to access their data and perform examinations.

HTTP Method	Endpoint URL	Description	Security Note
GET	/api/student/dashboard	Retrieves personal academic statistics (Enrolled courses, Average score).	Context-aware
GET	/api/student/quizzes	Lists available quizzes based on the student's enrolled classes.	Filtered by Enrollment

GET	/api/student/quizzes/{id}/take	Fetches quiz questions for the exam interface.	Hides `correctAnswer` field
POST	/api/student/quizzes/{id}/take	Submits student answers, triggers auto-grading, and saves the result.	Auto-calculates Score

4.8.5. HTTP Status Codes

The API utilizes standard HTTP status codes to indicate the success or failure of a request:

- 200 OK: The request was successful, and the response body contains the requested data.
- 401 Unauthorized: The request lacks a valid JWT token.
- 403 Forbidden: The user is authenticated but does not have the required Role (e.g., a Student trying to access Teacher APIs).
- 404 Not Found: The requested resource (e.g., Quiz ID, Class ID) does not exist in the database.
- 500 Internal Server Error: An unexpected server-side error occurred (e.g., Database connection failure).

5. Implementation

5.1. Development Environment

The system is developed using Java and the Spring Boot framework. Development and testing are performed using a modern integrated development environment (IDE). A relational database management system is used for data storage. Maven is utilized for dependency management and project build configuration.

5.2. Project Folder Structure

```
src/main/java/com/example/studentmanagementsystem
│
├── config          # Application-wide configurations (Security, Beans)
├── controller      # API Controllers (Handling HTTP requests)
├── dto             # Data Transfer Objects (Request/Response models)
├── entity          # JPA Entities (Database table mappings)
├── exception       # Global exception handling
├── repository      # Data Access Layer (Spring Data JPA interfaces)
├── security        # Security configurations (JWT Filter, Provider, etc.)
├── service         # Business logic layer
│   └── impl        # Service implementations
└── utils           # Utility classes (e.g., DataSeeder)
```

```
src/main/resources
├── static          # Static assets (CSS, JS, Images)
├── templates       # Thymeleaf views (HTML files)
│   ├── admin       # Admin-specific pages
│   ├── teacher     # Teacher-specific pages
│   ├── student     # Student-specific pages
│   └── fragments   # Reusable UI components (Header, Footer, etc.)
├── application.properties # Main application configuration
├── messages.properties    # I18n message bundles (Default/English)
└── messages_vi.properties # I18n message bundles (Vietnamese)
```

5.3. Backend Implementation

Backend Overview

The backend is implemented using Spring Boot, follows a layered architecture, and communicates with the frontend through the Controller layer.

Backend Layers

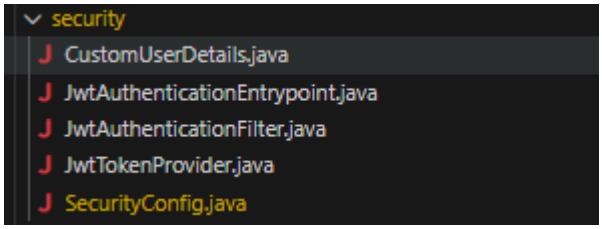
- Controller: Routes requests by role and delegates processing to services
- Service: Handles business logic, validates data, and applies system rules

- Repository: Manages database access and performs CRUD operations

Implemented Backend Functionalities

- Authentication and Authorization
- User and Role Management
- Course and Enrollment Management
- Grade and Attendance Management
- Quiz Processing
- Payment Simulation

5.4. Authentication and Authorization Implementation



Authentication

- User authentication is implemented using JWT (JSON Web Token).
- Users authenticate by submitting login credentials.
- Upon successful authentication, the backend generates a JWT and returns it to the client.
- The JWT is attached to subsequent requests for identity verification.

Authorization

- The system applies role-based access control.
- User roles include Admin, Teacher, and Student.
- Each role is restricted to specific backend endpoints and operations.
- Unauthorized access attempts are blocked at the security filter level.

Security Processing Flow

- Incoming requests pass through a JWT authentication filter.
- The filter validates the token and extracts user identity and role.
- Access permissions are checked before request processing continues.

Security Configuration

- Security rules and access policies are defined centrally.
- Public endpoints are separated from protected endpoints.
- Unauthorized requests return appropriate error responses.

5.5. REST API Implementation

API Namespace + Role-based Design

```
@RestController
@RequestMapping("/api/admin")
@PreAuthorize("hasRole('ADMIN')")
public class AdminController { }
```

- Groups all administrative APIs under the /api/admin path
- Applies role-based authorization at the controller level
- Restricts access to users with the ADMIN role only
- Clearly separates Admin APIs from Teacher and Student APIs

RESTful CRUD API – User Management

```
@PostMapping("/users/create-account")
public ResponseEntity<UserResponseDTO> createUser(
        @RequestBody CreateUserRequestDTO request) {
    return new ResponseEntity<>(userService.createUser(request), HttpStatus.CREATED);
}
```

- Uses HTTP POST for resource creation
- Accepts input data via request DTO
- Returns response DTO instead of entity
- Delegates business logic to the Service layer

```
@DeleteMapping("/users/{id}")
public ResponseEntity<String> deleteUser(@PathVariable Long id) {
    userService.deleteUser(id);
    return ResponseEntity.ok("User deleted successfully");
}
```

- Uses HTTP DELETE to remove a resource
- Identifies the target user by path variable
- Follows standard RESTful conventions

Dashboard Statistics API (Aggregated JSON Response)

```java
@GetMapping("/stats")
public ResponseEntity<Map<String, Long>> getDashboardStats() {
    Map<String, Long> stats = new HashMap<>();
    stats.put("students", userRepository.countByRole(Role.STUDENT));
    stats.put("teachers", userRepository.countByRole(Role.TEACHER));
    stats.put("courses", courseRepository.count());
    stats.put("classes", classroomRepository.count());
    return ResponseEntity.ok(stats);
}
```

- Aggregates data from multiple repositories
- Returns statistical data in JSON format
- Designed to support dashboard and chart visualization
- Avoids returning raw entity data

Context-Aware API – Student Dashboard

```java
@GetMapping("/dashboard")
public ResponseEntity<StudentDashboardDTO> getDashboard(Authentication authentication) {
    CustomUserDetails userDetails =
        (CustomUserDetails) authentication.getPrincipal();
    return ResponseEntity.ok(
        userService.getStudentDashboardInfo(userDetails.getId()));
}
```

- Extracts user identity from the security context
- Returns data based on the authenticated user
- Ensures students can access only their own information
- Enforces data isolation at the backend level

Role Enforcement at Controller Level (Teacher)

```java
java

@RestController
@RequestMapping("/api/teacher")
@PreAuthorize("hasRole('TEACHER')")
public class TeacherController { }
```

- Applies role-based authorization to all teacher APIs
- Restricts access to users with the TEACHER role
- Reduces repetitive security annotations on individual endpoints

Academic Processing API – Grade and Attendance

```java
@PostMapping("/grades")
public ResponseEntity<Grade> saveGrade(@RequestBody Grade grade) {
    return ResponseEntity.ok(gradeService.saveOrUpdate(grade));
}
```

- Allows teachers to create or update student grades
- Delegates validation and business rules to the Service layer
- Keeps Controller logic minimal and focused

```java
@PostMapping("/attendance")
public ResponseEntity<?> markAttendance(
        @RequestBody AttendanceRequestDTO request) {
    return ResponseEntity.ok(attendanceService.markAttendance(request));
}
```

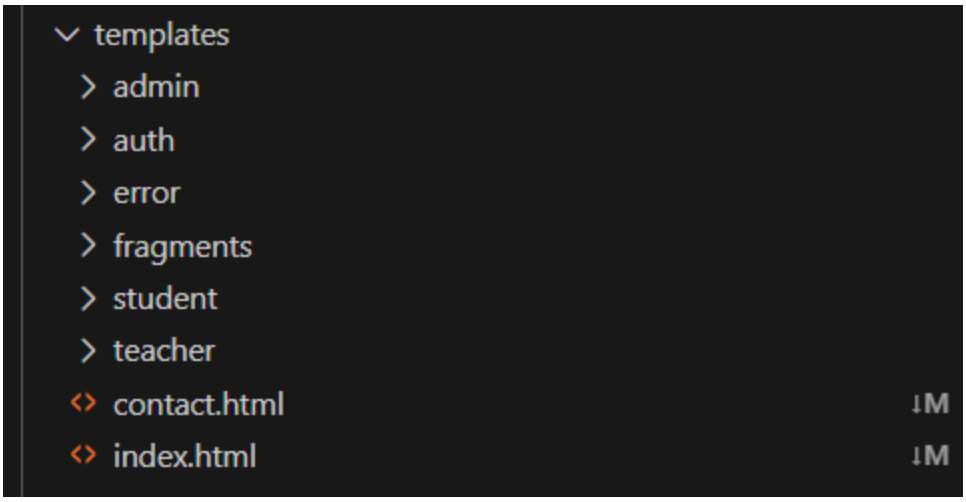
- Uses a DTO to handle structured attendance input
- Prevents direct exposure of entity models
- Ensures consistent and validated request data

Summary for Section 5.5

- REST APIs are organized by user roles
- Standard HTTP methods are applied consistently
- Role-based access control is enforced at the backend
- DTOs are used for secure data exchange
- APIs support administrative, academic, and user-specific operations

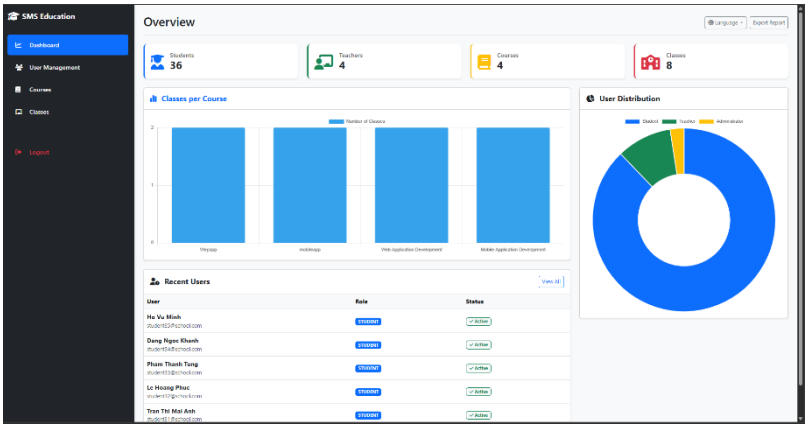
5.6. User Interface Implementation

The user interface of the system is implemented using Thymeleaf with a server-side rendering approach and is tightly integrated with Spring Boot. UI templates are organized within the templates directory based on user roles.



Separate interfaces are provided for Admin, Teacher, and Student, ensuring that each role can access only the features permitted by the system's authorization rules. This role-based structure improves clarity, security, and maintainability.

Common UI components such as headers, footers, and sidebars are stored in the fragments directory to promote reusability and consistent layout design. The system also includes authentication pages and error handling pages to support secure access and user experience.



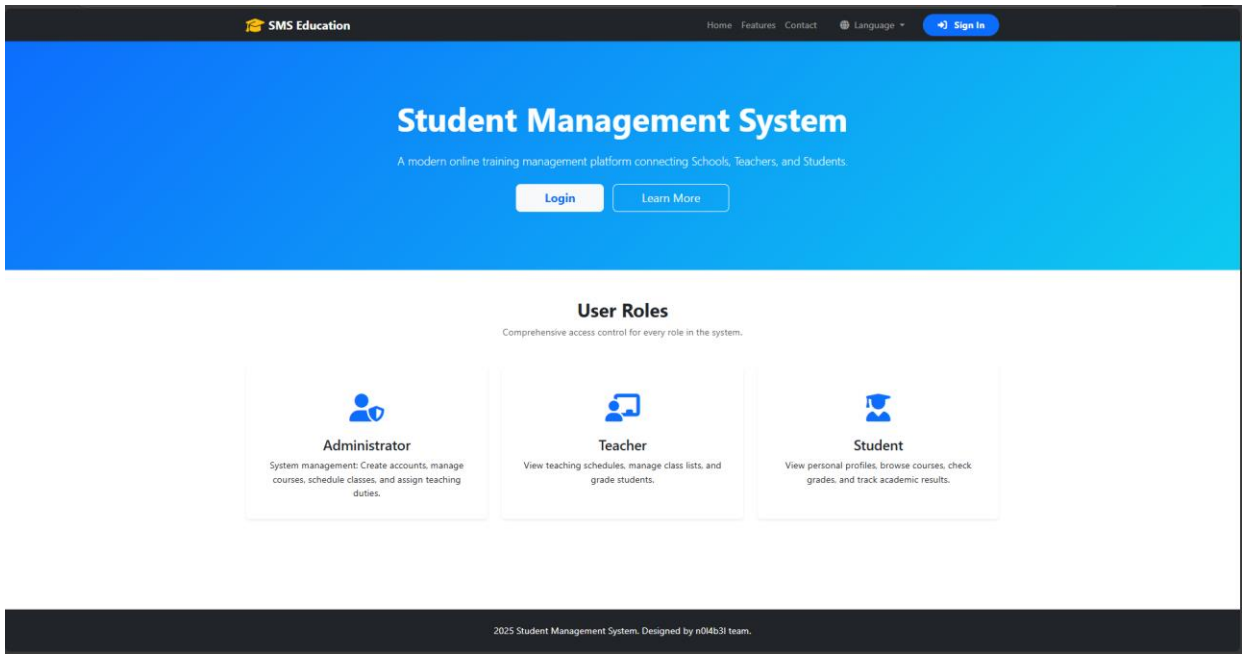
As shown in figure, the UI templates are clearly separated by user roles, including Admin, Teacher, and Student.

UI interactions are handled through Spring Boot controllers, which process user requests and render the appropriate HTML views with data supplied by the backend services.

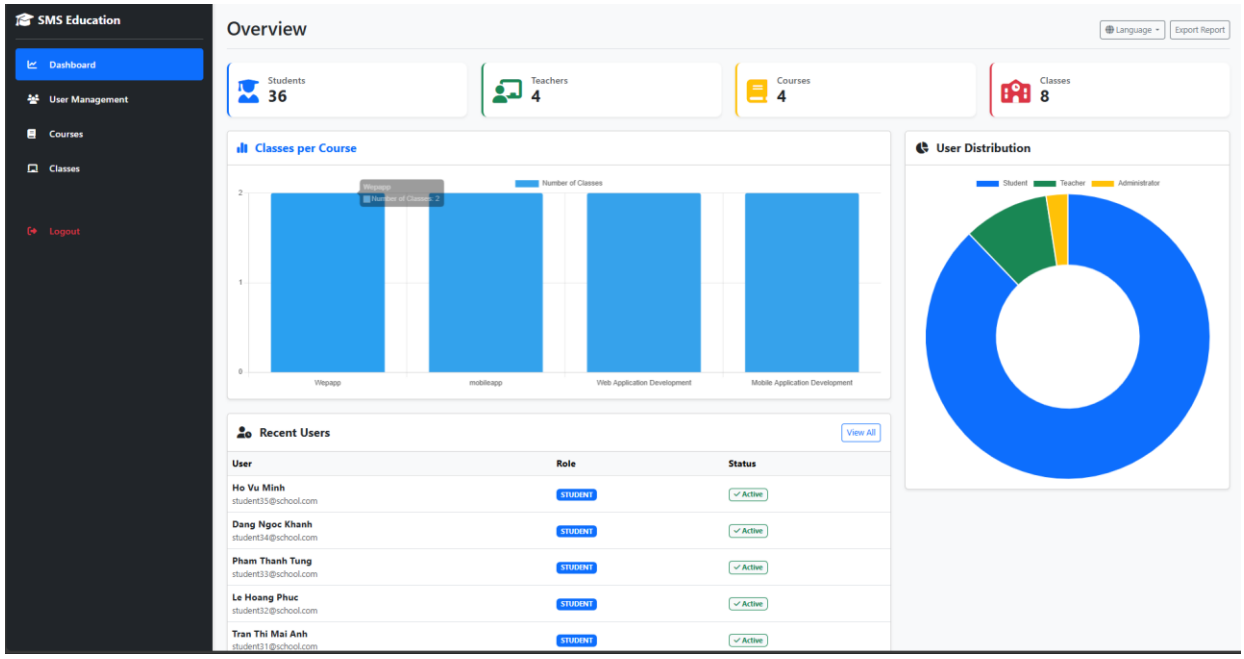
6. Results and Evaluation

6.1. System Demonstration

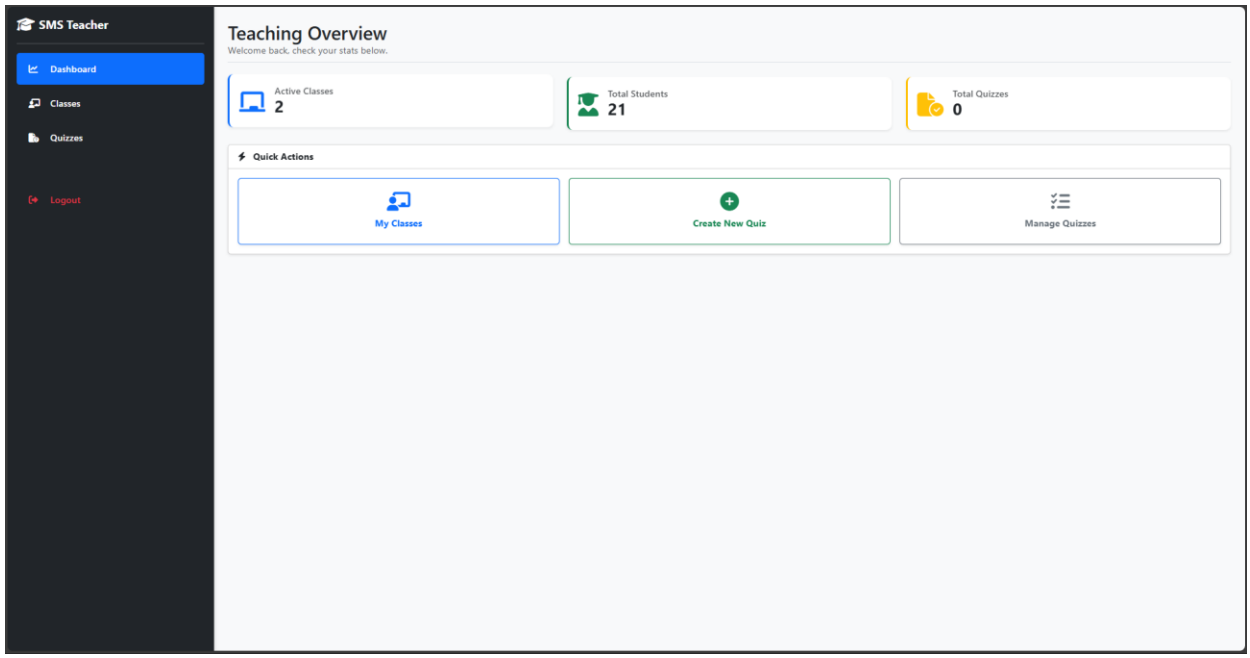
6.1.1. Login Page



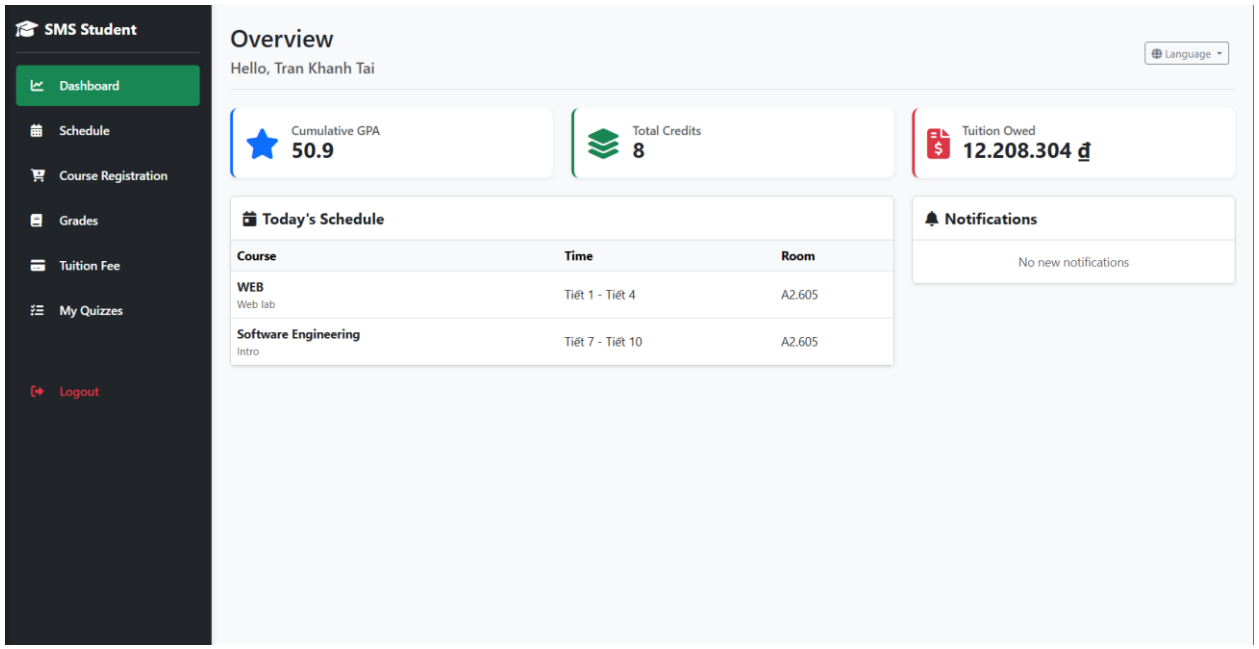
6.1.2. Admin Dashboard



6.1.3. Teacher Dashboard



6.1.4. Student dashboard



6.2. Performance Evaluation

Through testing, I found the system to be stable and well-suited to basic needs:

- Speed: Websites load quickly, and page transitions and data saving occur almost instantly thanks to the use of Spring Boot.

- Page Loading: The Bootstrap interface is lightweight and doesn't strain

the computer during access.

- Data: Information retrieval (such as viewing grades or schedules) is smooth, with no lag even with few users.

6.3. Security Evaluation

In terms of security, the system has ensured the most basic safety elements:

- Authentication: Login is required to use the system. Important links are blocked without login.
- Permissions: Three roles are clearly defined. Students cannot access the teacher's or administrator's management page.
- Passwords: User passwords are not stored as plain text but are encrypted (hash) in the database, providing better account security.

6.4. User Experience Evaluation

The system is designed for simplicity and ease of use for students:

- Interface: Clear layout, left-hand menu makes it easy for users to find the functions they need.
- Convenience: Includes an English-Vietnamese language switching feature for convenient use by various users.
- Visual: Important information such as grades and course status are highlighted in distinct colors for easy identification.

6.5. Limitations of the System

Due to time and knowledge limitations, the project still has some shortcomings:

- Payment: The tuition payment function is only a simulated interface and has not yet connected to banks for real-world payments.
- Real-time: The system lacks instant push notifications. Students still have to manually refresh the page (F5) to see updated grades or schedules.
- Forgot password: An automatic password recovery email function has not yet been integrated; currently, students need to rely on the administrator for support.

7. Conclusion and Future Work

7.1 Conclusion

The "Student Management System" (SMS) project has been successfully designed and developed, meeting the basic requirements for digital transformation in the higher education environment. The system has solved the problem of centralized management, effectively connecting three stakeholders: Administrators, Teachers, and Students.

Throughout the project, We successfully applied modern technologies such as Java Spring Boot for the backend, Thymeleaf & Bootstrap 5 for the frontend, and MySQL for the database.

7.2 Achievements of the Project

Functional Achievements:

- Student Module: Completed core features such as viewing timetables, looking up transcripts, online course registration, and tracking tuition fees.
- Faculty Module: Provides effective class management tools, allowing for automatic grade calculation (GPA/CPA) and the creation of multiple-choice question banks (Quiz).
- Security: Successfully built a secure login/logout mechanism, password encryption, and role-based access control.
- Multilingual (i18n): The system supports flexible switching between Vietnamese and English, enhancing the user experience.

Technical Achievements:

- We built a responsive interface that is compatible with both desktop devices.
- We use Spring Data JPA to optimize database operations.
- The project structure adheres to clean code standards, making it easy for team development.

7.3 Future Enhancements

Although the system is operating stably, there are still some limitations that need improvement to become a complete commercial product. Further development directions include:

- Online payment integration: Connecting with payment gateways (VNPay, Momo, PayPal) so students can pay tuition fees directly through the system.
- Mobile application development: Building an app that synchronizes data with

the website, allowing students to receive push notifications about class schedules and grades instantly.

- Real-time functionality: Integrating WebSockets to create an online chat feature between instructors and students, or providing immediate notifications when there are changes to the class schedule.

- Advanced Online Exam Features: Adding proctoring, randomizing questions per student, and detailed analytics for quiz results

8 REFERENCES AND GITHUB LINK

8.1 GitHub Link:

https://github.com/KhanhTaiTran/Student-Management-System.git

8.2 References:

Books:

1. *C. Walls, Spring in Action, 6th ed. Shelter Island, NY: Manning Publications, 2022.*
2. *L. Spilca, Spring Security in Action. Shelter Island, NY: Manning Publications, 2020.*
3. *R. C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design. Upper Saddle River, NJ: Prentice Hall, 2017.*
4. *G. King, MySQL 8 Administrator's Guide. New York, NY: McGraw-Hill Education, 2018.*

Official Documentation:

5. VMware, Inc., "Spring Boot Reference Documentation," Spring.io, 2024. [Online]. Available: https://docs.spring.io/spring-boot/docs/current/reference/html/. [Accessed: Dec. 21, 2024].

6. Thymeleaf, "Thymeleaf: Natural Templates for Java," Thymeleaf.org, 2023. [Online]. Available: https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html. [Accessed: Dec. 21, 2024].

7. Oracle, "Java Platform, Standard Edition Documentation," Oracle.com, 2023. [Online]. Available: https://docs.oracle.com/en/java/. [Accessed: Dec. 21, 2024].

8. The Bootstrap Team, "Bootstrap 5 Documentation," GetBootstrap.com, 2023. [Online]. Available: https://getbootstrap.com/docs/5.0/getting-started/introduction/..

Academic Papers & Standards:

9. R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000. (Foundational paper for REST API)

10. D. F. Ferraiolo and R. Kuhn, "Role-Based Access Controls," in Proc. 15th National Computer Security Conference, Baltimore, MD, 1992, pp. 554-563. (Foundational paper for RBAC Security)

11. M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, May 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7519.