1. Executive Summary

1.1. Goal & Scope

Week 2 delivered a secure, role-based foundation for the Course Enrollment System, focused on user authentication, course creation, student enrollment, and basic admin oversight, using only core academic entities.

Key Achievements:

- **Security:** Full Role-Based Access Control (RBAC) with method-level security and stateless session management.
- **Scheduling:** Robust classroom scheduling with double-booking prevention and capacity validation.
- **Enrollment Logic:** Automated waitlist handling; when a student drops a course, the next student on the waitlist is automatically promoted.
- **Attendance:** Lecturers can record attendance with strict data integrity (one record per student/session).
- **Infrastructure:** Complete database schema versioning via Flyway (Migrations V1–V4).

1.2. Technologies

- **Backend:** Spring Boot, Spring Security (Stateless/JWT), Spring Data JPA
- **Frontend:** Thymeleaf, Tailwind CSS, JavaScript
- **Database:** MySQL with Flyway Migration
- **Tools:** Maven, Git, Postman

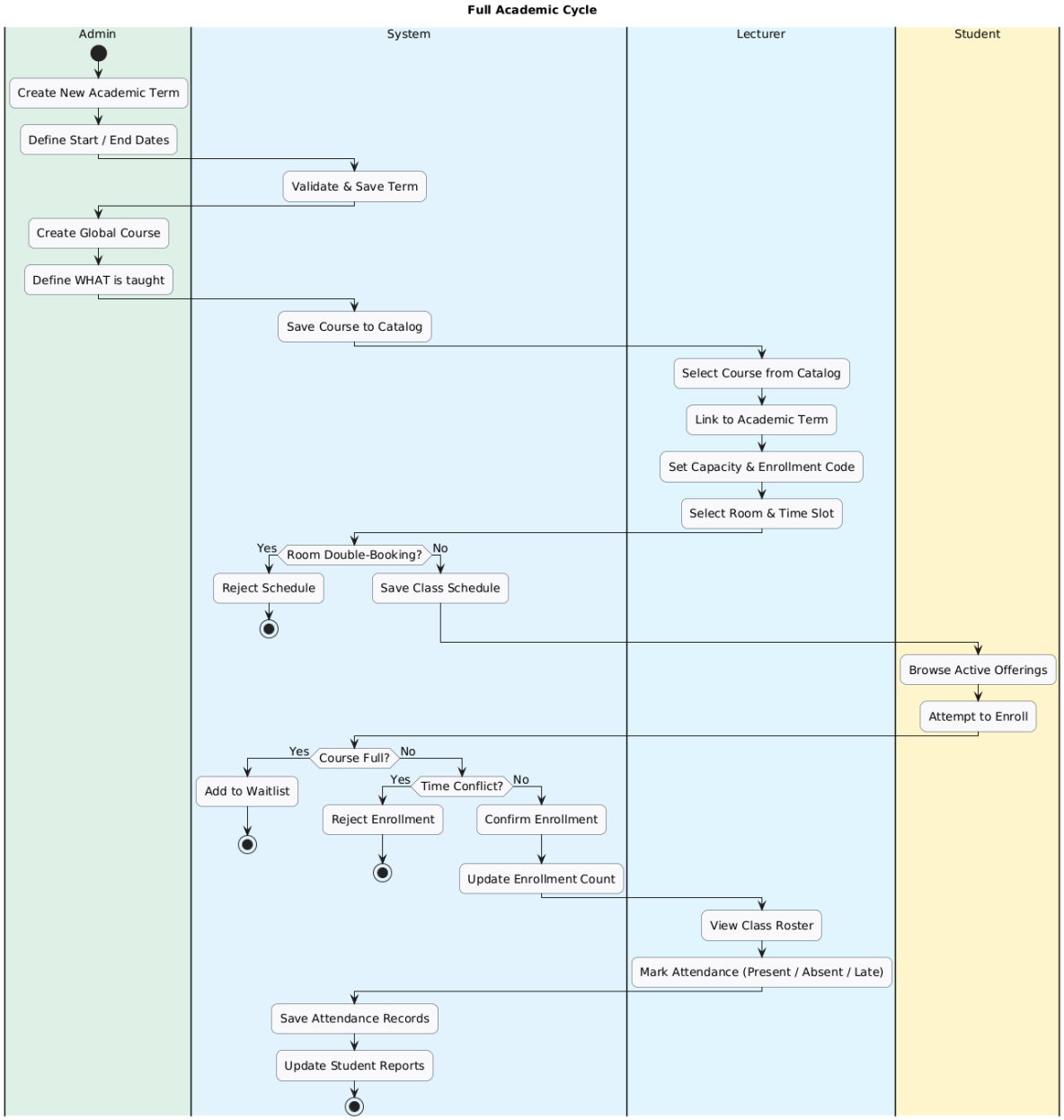
1.3. Team Roles and Contributions

Role	Team Member
Security & Authentication Lead	David
Student Features	Rasy
Lecturer Features	Hour
Admin Features	Seyha
Database & Infrastructure	Rith

2. System Overview

The **Course Enrollment & Classroom Scheduling System** is a web-based academic management platform designed to streamline course registration, scheduling, and role-based interactions among **Students**, **Lecturers**, and **Administrators**. Built using a classic **Spring**

Boot layered architecture, the system ensures separation of concerns, testability, and maintainability.



2.1. Core Modules Delivered

- **Administrative Core:** Centralized management of Users, Academic Terms, Rooms, and the Global Course Catalog (ensuring a standardized curriculum).
- **Course Offerings:** Lecturers manage class instances (Offerings) by selecting existing courses from the catalog and assigning them to specific terms, rather than creating courses from scratch.
- **Scheduling Engine:** Validation logic enforces conflict checks to prevent room double-booking and ensures room capacity accommodates the class size.

- **Enrollment Engine:** Automates capacity management by triggering Waitlist placement for full courses and executing auto-promotion when seats become available.
- **Attendance System:** Tracks student presence (Present, Absent, Late) per class session with access controls restricted to the assigned lecturer.

2.2. Key Characteristics

- **User Roles**:
  - **Student**: Browse courses, enroll, view schedule
  - **Lecturer**: Create courses, assign schedules, record attendance
  - **Admin**: Manage academic terms, rooms, and system metrics
- **Core week 1 & 2 Scope**: Authentication, course creation, enrollment, and role-based dashboards
- **Data Integrity**: Enforced via MySQL constraints and Flyway-managed schema
- **Security**: Role-based access at URL, method, and UI levels
- **UI**: Responsive, server-rendered with Thymeleaf + Tailwind CSS

2.3. High-Level Architecture

The system follows a clean 5-layer architecture, from user interface down to persistent storage:

- **UI Layer**: Renders responsive HTML pages using Thymeleaf templates; handles user input and displays feedback.
- **Controller Layer**: Receives HTTP requests, validates input, and delegates to services; returns view names or redirects.
- **Service Layer**: Contains core business logic (enrollment rules, validation, waitlist promotion); transactional boundary.
- **Repository Layer**: Manages data access via JPA; abstracts SQL interactions.
- **Database Layer**: Persistent storage using MySQL with enforced constraints (unique keys, foreign keys, checks).

3. Requirements Specification

3.1. Functional Requirements

- **User Login:** Students, Lecturers, and Admins log in securely and are redirected to their specific role-based dashboard.
- **Create Course:** Lecturers can create course offerings with validation for course codes (like GIC25DB), capacity, and term assignment.
- **Class Scheduling:** Lecturers can schedule classes (Day/Time/Room). The system prevents double-booking rooms.
- **Enrollment & Drops:** Students can enroll in available courses with enrolment code. They can also drop a course, which triggers a status update.

- **Automated Waitlist:** If a course is full, students are added to a waitlist. Crucially, if an enrolled student drops, the system automatically promotes the next waitlisted student.
- **Attendance Tracking:** Lecturers can record student attendance (Present, Absent, Late) for specific class sessions.
- **Admin Management:** Admins manage infrastructure (Rooms, Academic Terms) and view system-wide metrics.

3.2. Non-Functional Requirements

- **Secure**: Users only see features for their role.
- **Reliable**: Data rules (like unique course codes and capacity limits) are enforced.
- **Maintainable**: Database changes use Flyway; code is reviewed before merging.
- **User-Friendly**: Clear messages (like "Added to waitlist") appear after actions.

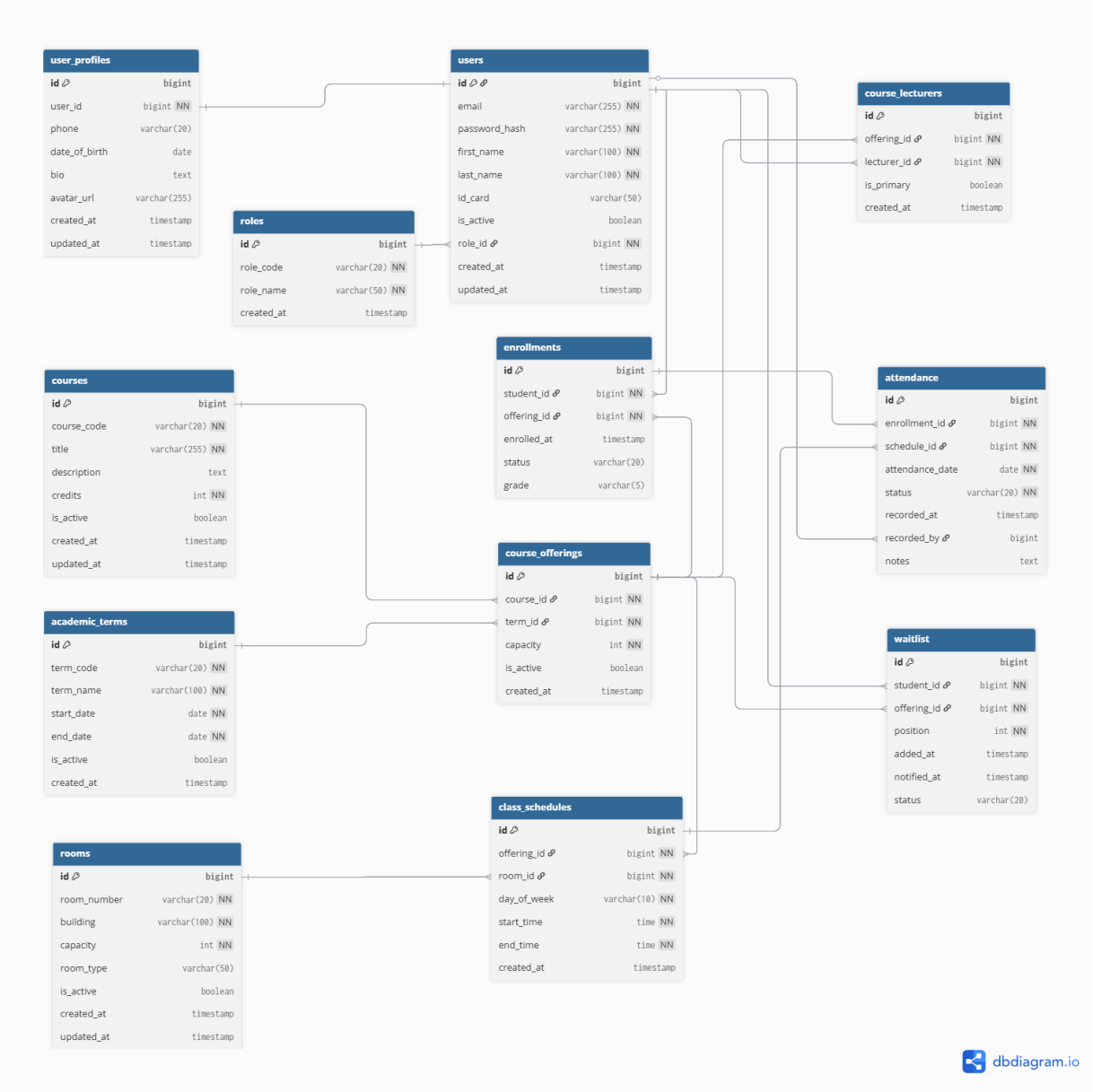
4. Database Design

4.1. Entity-Relationship Diagram (ERD)

The system uses a normalized **12-table relational schema** designed to support flexible course offerings, multi-lecturer assignments, waitlists, scheduling, and attendance. The core structure separates reusable definitions (like courses) from term-specific instances (course_offerings) and physical resources (rooms).

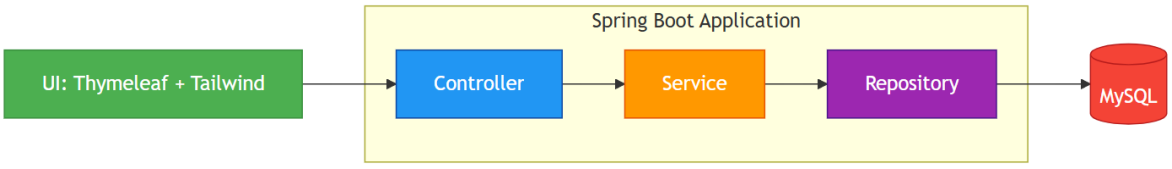
Key Constraints Added:

- uk_schedule: Prevents room conflicts (Room + Day + Time).
- uk_attendance: Ensures one attendance record per student per session.



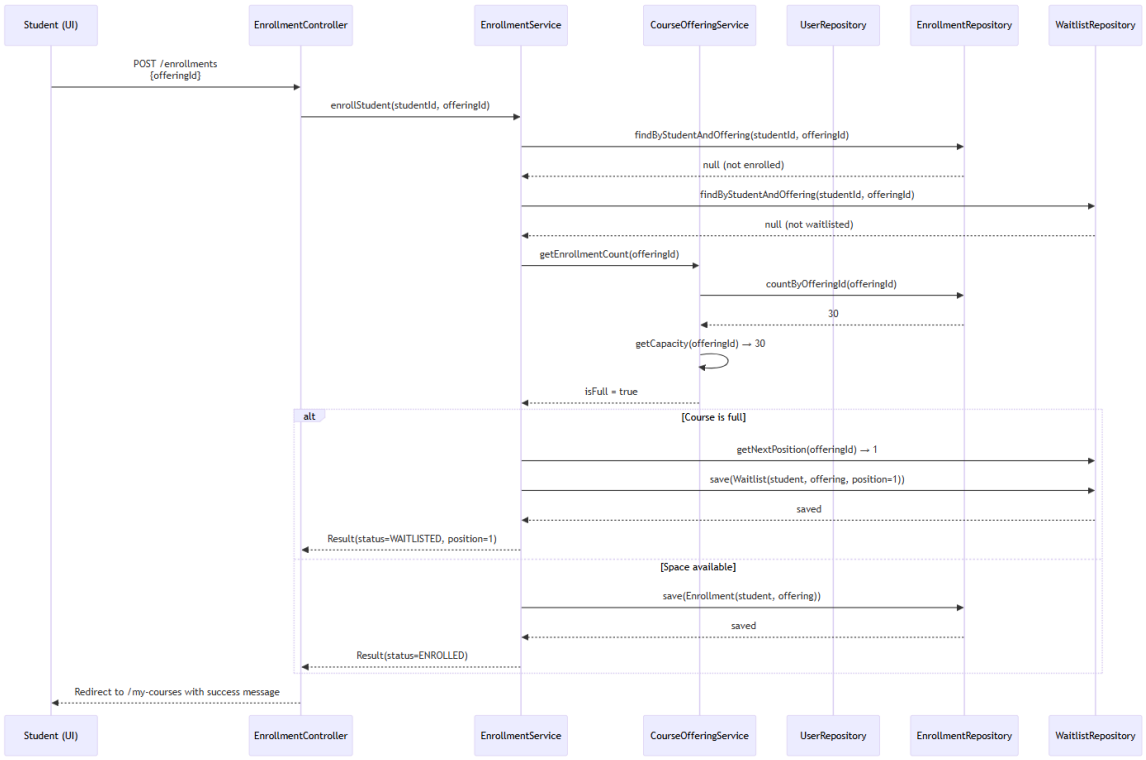
5. System Architecture & Design

5.1. Component Diagram



5.2. Sequence Diagram: Student Enrollment Flow

This diagram captures the **core Sprint 1 workflow**: a student enrolling in a course, including **capacity check** and **automatic waitlist fallback**.

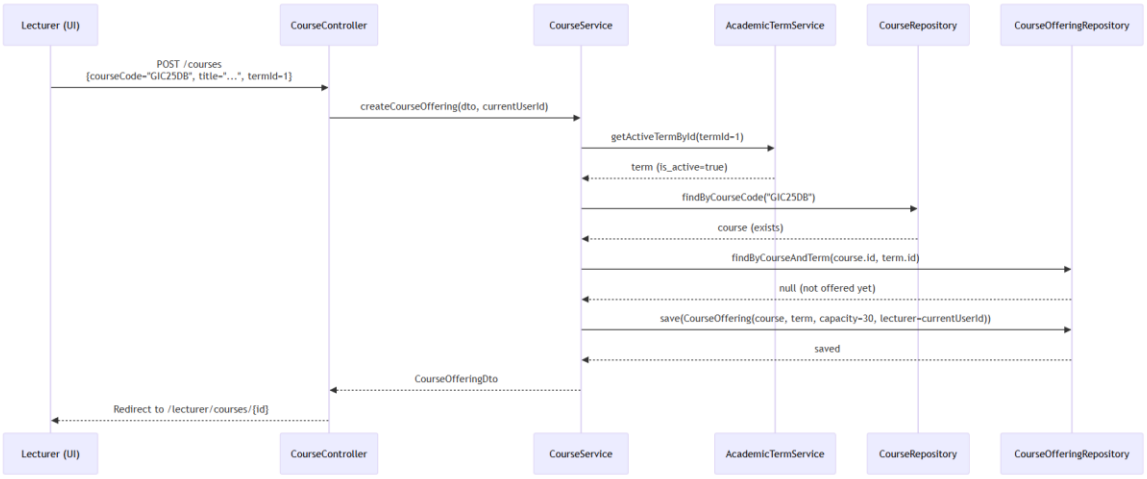


Rules Enforced:

- No duplicate enrollment/waitlist
- Capacity check → auto waitlist
- Atomic save operation

5.3. Sequence Diagram: Lecturer Course Creation

This diagram shows how a lecturer creates a **course offering** in an active academic term, with **auto-assignment of lecturer** and **code validation**.



6. Security Implementation

6.1. Authentication Flow

- **User Login:**
  - Users authenticate via email/password at /api/auth/login.
  - Credentials are verified against the users table.
- **Token Generation:**
  - On success, the server generates a **JWT (JSON Web Token)** containing the user's email and role_id.
  - This token is returned to the client and stored (like LocalStorage/Cookie).
- **Stateless Request Processing:**
  - Every subsequent HTTP request includes the JWT in the Authorization header.
  - The JwtFilter intercepts requests, validates the token, and sets the Spring Security context (Stateless Session).

6.2. Data & Password Security

Aspect	Implementation
Password Storage	BCrypt hashing (BCryptPasswordEncoder) with strength 10.
Sensitive Data	**Stateless** (No server-side session storage), reducing server memory load.
CSRF Protection	URL-based (.requestMatchers()) and Method-based (@PreAuthorize) security.
SQL Injection	Prevented via Spring Data JPA repositories and parameterized queries.

7. UI/UX Design

The user interface is built with **server-rendered Thymeleaf templates** and styled using **Tailwind CSS (via CDN)**. The design follows a **component-based approach**, promoting reuse and consistency.

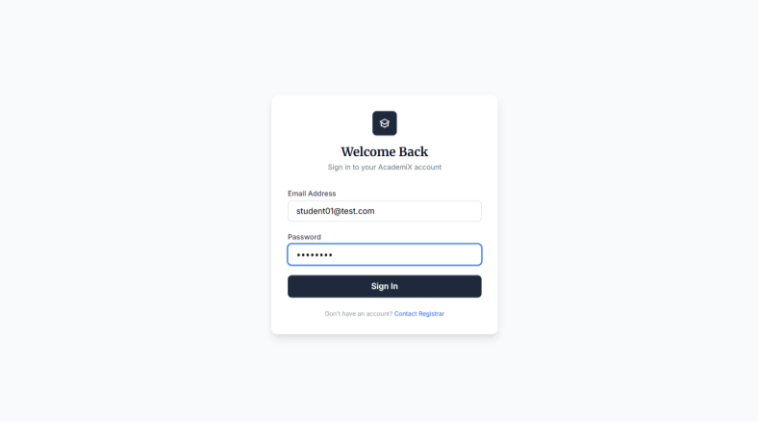
7.1. Design Principles

- **Role-Centric Views:** Security tags (`sec:authorize`) hide elements dynamically. A student never sees "Take Attendance" buttons.
- **Visual Scheduling:** Time blocks are rendered in a grid format rather than simple lists.
- **Feedback Loops:** Toast notifications appear for "Waitlist Joined" or "Attendance Saved".

7.2. Key Screens (week 1-2)

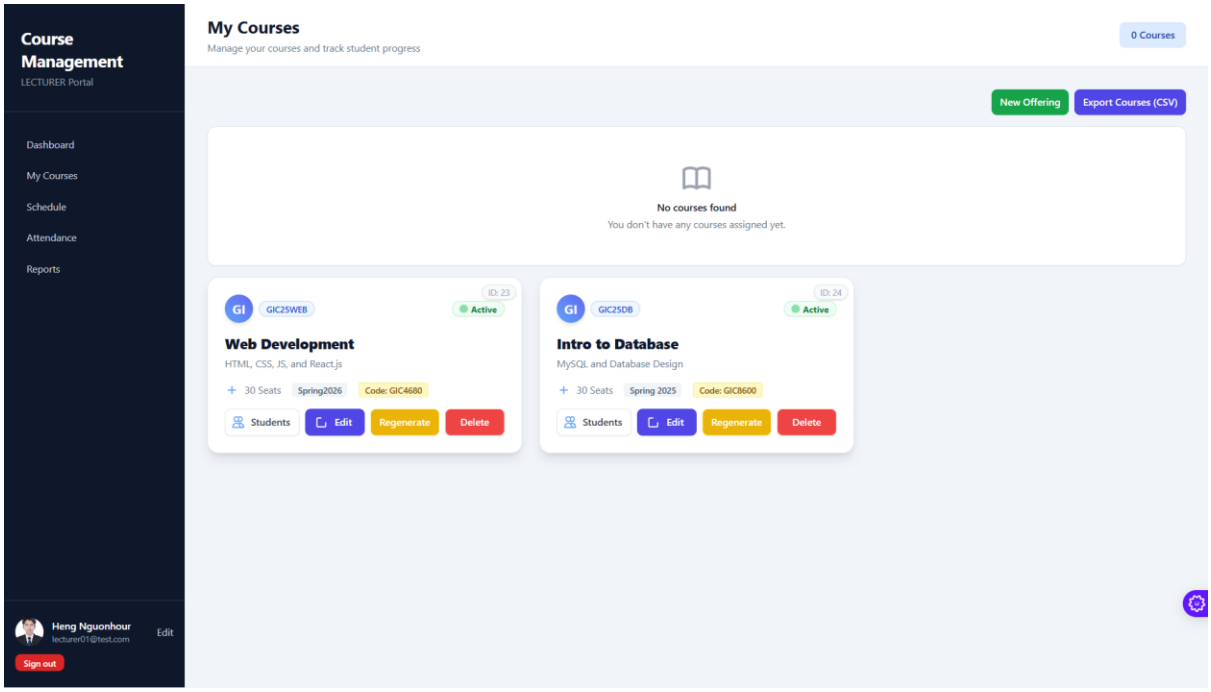
🎓 Login Page

- Clean form with email/password fields
- Error display for invalid credentials
- University-branded header

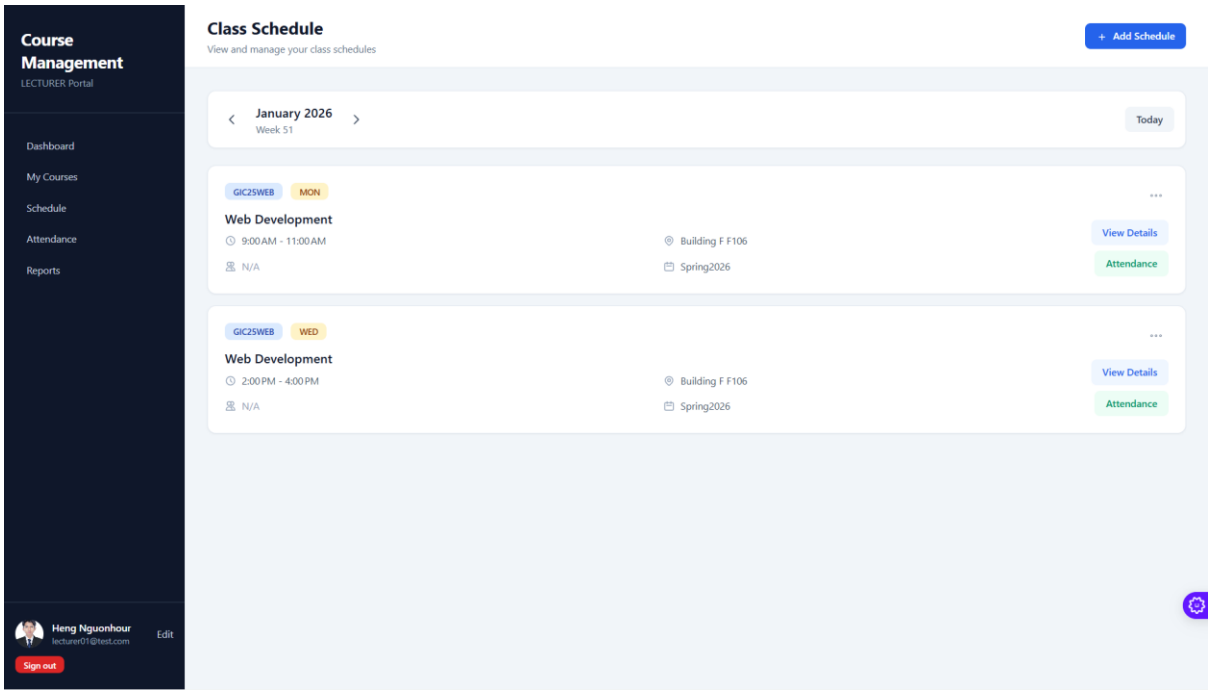


👩‍🏫 Lecturer Dashboard

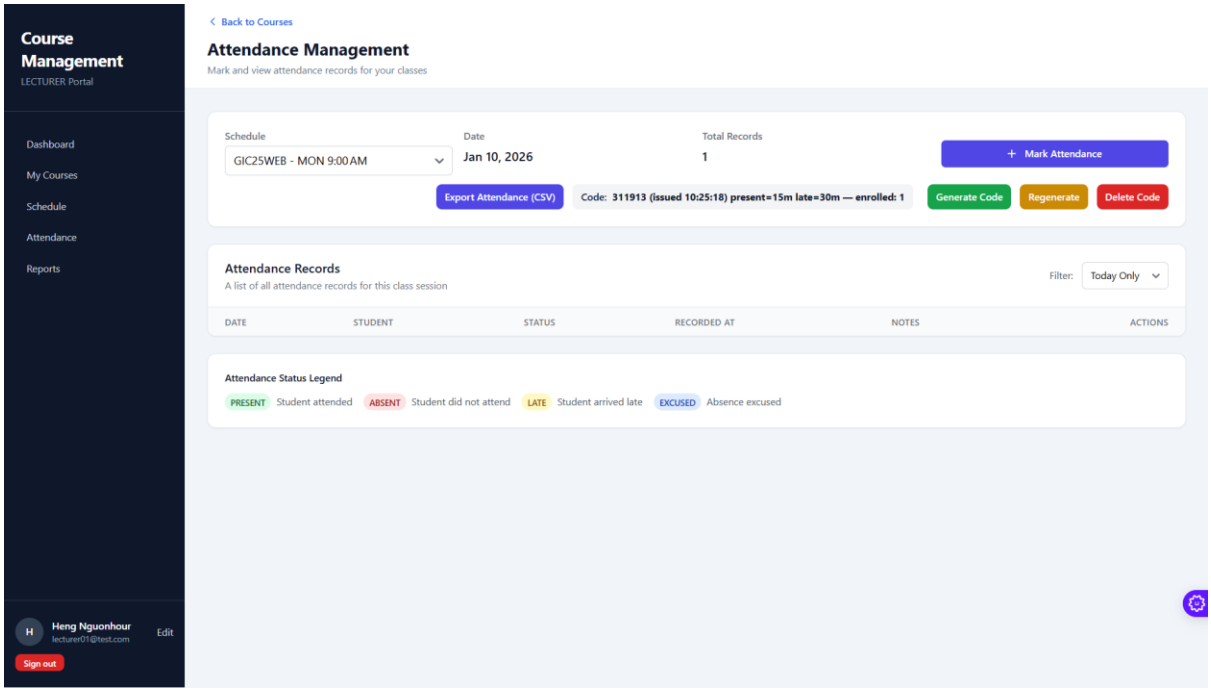
- **My Courses:** List of active courses.

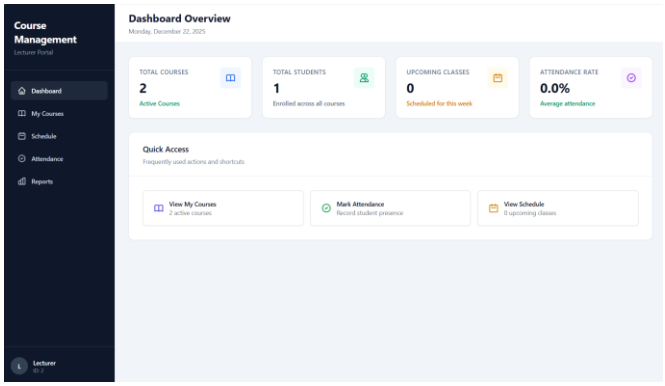


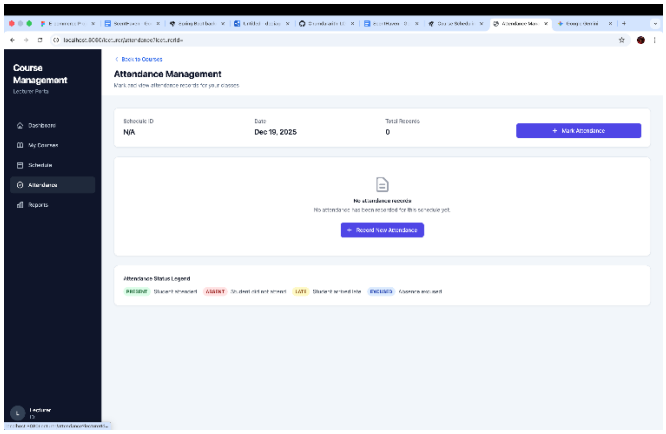
- **Schedule Manager:** Form to assign rooms/times with conflict error handling.

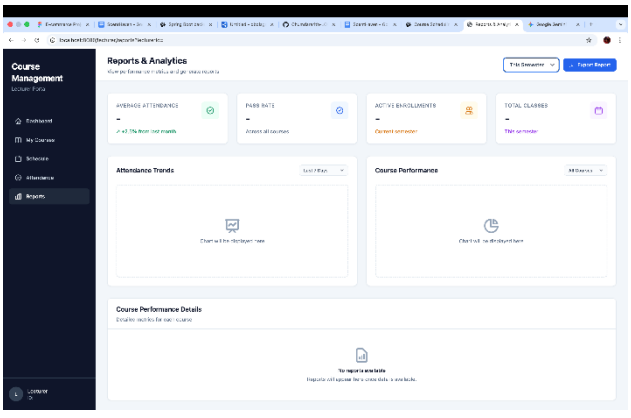


- **Attendance Sheet:** Table view of students with radio buttons for status (Present/Absent).


Course Management
LECTURER Portal
Dashboard
My Courses
Schedule
Attendance
Reports
Heng Nguonhour
lecturer01@test.com
Edit
Sign out
Back to Courses
Attendance Management
Mark and view attendance records for your classes
Schedule
GIC25WEB - MON 9:00 AM
Date
Jan 10, 2026
Total Records
1
Mark Attendance
Export Attendance (CSV)
Code: 311913 (issued 10:25:18) present=15m late=30m — enrolled: 1
Generate Code
Regenerate
Delete Code
Attendance Records
A list of all attendance records for this class session
Filter:
Today Only
DATE
STUDENT
STATUS
RECORDED AT
NOTES
ACTIONS
Attendance Status Legend
PRESENT Student attended
ABSENT Student did not attend
LATE Student arrived late
EXCUSED Absence excused

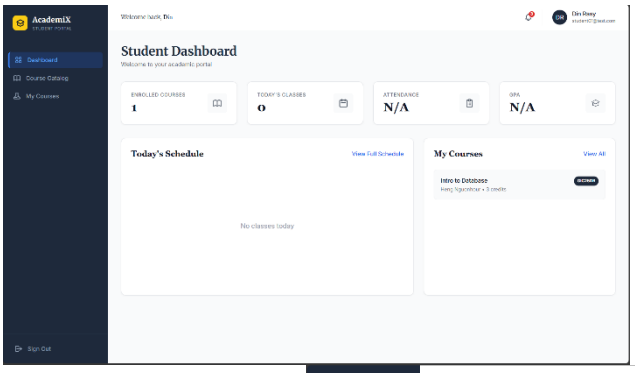

Course Management
Dashboard Overview
TOTAL COURSES
2
Active Courses
TOTAL STUDENTS
1
Enrolled across all courses
UPCOMING CLASSES
0
Scheduled for this week
ATTENDANCE RATE
0.0%
Average attendance
Quick Access
View My Courses
Mark Attendance
View Schedule

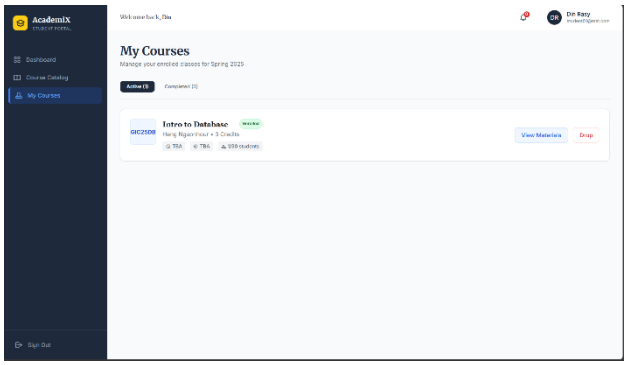

Course Management
Attendance Management
Schedule
N/A
Date
Dec 19, 2025
Total Records
0
Mark Attendance
No attendance records
Record New Attendance
Attendance Status Legend

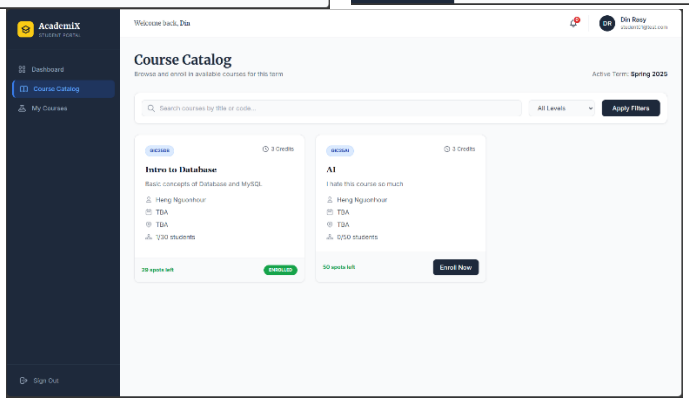

Course Management
Reports & Analytics
AVERAGE ATTENDANCE
PASS RATE
ACTIVE ENROLLMENTS
TOTAL CLASSES
Attendance Trends
Course Performance
Course Performance Details

👨‍💼 Student Dashboard

- **Course Catalog**: Grid of available courses in active term
- **My Courses**: List of enrolled courses (status: “Enrolled”)
- Empty states: “No courses yet” illustration with CTA

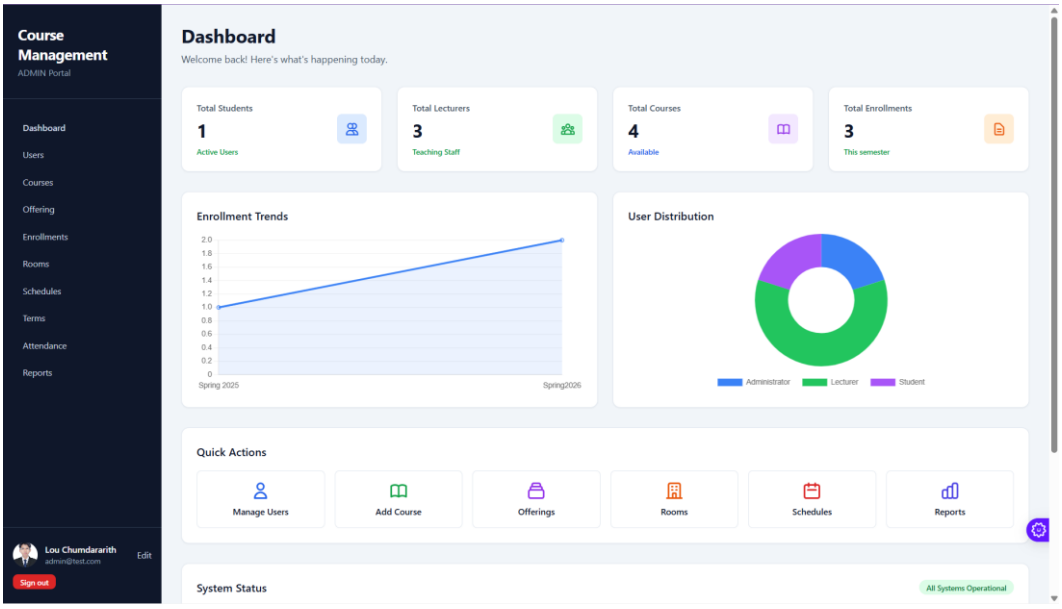






👔 Admin Dashboard

- Metrics cards: **Total Students**, **Lecturers**, **Courses**, **Active Terms**



- Secure access (403 if non-admin)
- Navigation stubs for future term/room management

8. Development Process

Our team followed an **agile, collaborative workflow** over **Sprint 1 (8 – 22 December 2025)**.

8.1. Team Structure & Responsibilities

Role	Member	Focus Area
Security & Authentication Lead	David	Method-level security, JWT implementation, CSRF, Login/Logout logic.
Main Entity CRUD Developer	Rasy	Enrollment rules, Drop logic, Waitlist Auto-Promotion engine.
Secondary Module CRUD Developer	Hour	rooms, class_schedules, attendance (Sprint 2 prep)
Admin Developer	Seyha	Admin dashboard, role-based UI, system metrics, Thymeleaf layout
Database & DevOps Lead	Rith	MySQL schema, Flyway migrations, seeding, dev setup

8.2. Git & Code Review Workflow

We adopted a **dual-commit review strategy** to ensure quality before merging:

a. Draft Commit

- Developer implements a logical unit of work (like "build admin dashboard UI")
- Pushes commit with `[REVIEW] draft: ...` message
- Notifies team for inspection

b. Peer Review

- At least one teammate tests the feature and checks code quality
- Feedback is given via Git comments or direct discussion

c. Official Commit

- After approval, developer amends the draft commit with a conventional message
- Example:
  - Draft: `[REVIEW] draft: create admin dashboard with user counts`
  - Official: `feat(admin): implement admin dashboard with system metrics`

d. Merge to **main**

- Only **official commits** are merged via pull request

- **No draft commits** remain in the final history

Result: Clean, professional Git history with full team participation.

8.3. Branching Strategy

Branch Type	Example	Purpose
main	main	**Protected** — stable, deployable code
Developer's Name	Lou-chumdararith	**For each user tasks** — one per task card
develop	release/sprint-1	Final integration branch (optional)

- All work done in **feature branches**
- Merged to develop **only after review and approval**
- main always passes mvn spring-boot:run with empty DB

8.4. Tools & Technologies

Category	Tools
IDE	IntelliJ IDEA, VS Code, MySQL Workbench
Backend	Spring Boot 3.3, Spring Security, Spring Data JPA
Frontend	Thymeleaf, Tailwind CSS (via CDN)
Database	MySQL 8.0, Flyway, HikariCP
Build	Maven
Version Control	Git + GitHub
Diagrams	dbdiagram.io, Mermaid, Plant UML

Appendices

Appendix A: Full ER Diagram (Sprint 1 Schema)

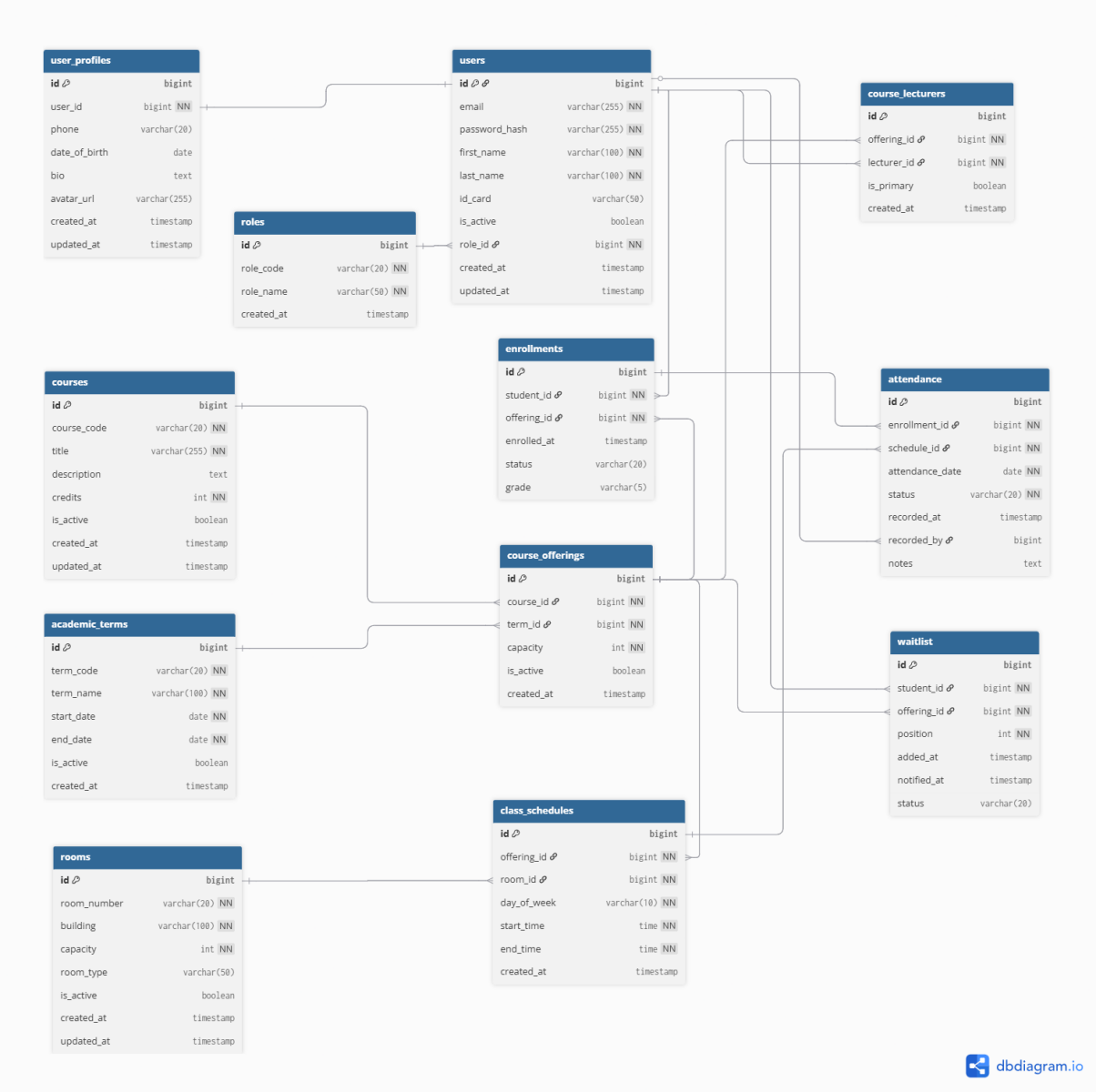


Figure A1: Complete Entity-Relationship Diagram showing all 12 tables and relationships. Generated using dbdiagram.io.

🔗 **View/Edit Online**: https://dbdiagram.io/d/6898a585dd90d17865441dfa

Appendix B: Sample Flyway Migration (V1__init_schema.sql)

```sql
CREATE TABLE users (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    id_card VARCHAR(50) UNIQUE COMMENT 'Student/Lecturer ID',
    is_active BOOLEAN DEFAULT TRUE,
    role_id BIGINT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
    FOREIGN KEY (role_id) REFERENCES roles(id),
    INDEX idx_users_email (email),
    INDEX idx_users_id_card (id_card),
    INDEX idx_users_role (role_id),
    INDEX idx_users_active (is_active),
    INDEX idx_users_name (first_name, last_name),
    INDEX idx_users_active_role (is_active, role_id)
);
```

E

Appendix C: Key API Endpoints

Method	Endpoint	Description	Role
GET	/login	Login page	Public
POST	/login	Authenticate user	Public
GET	/student/catalog	View available courses	STUDENT
POST	/enrollments	Enroll in course	STUDENT
GET	/lecturer/courses	View my courses	LECTURER
GET	/lecturer/courses/new	Show course creation form	LECTURER
POST	/courses	Create course offering	LECTURER
GET	/admin/dashboard	Admin metrics dashboard	ADMIN

Appendix D: Team Member Contributions (week 1-2)

Member	Key Deliverables
David	Spring Security config, role-based redirects, login/logout, sec:authorize integration
Rasy	Course, Enrollment, Waitlist entities/services, capacity logic, duplicate prevention
Hour	Room, ClassSchedule, Attendance entity design; scheduling conflict rules (Sprint 2 prep)
Seyha	Admin dashboard, Thymeleaf layout, role-aware UI fragments, responsive design
RITH	Full Flyway migrations (V1–V2), MySQL setup, seed data, report documentation