



KINGDOM OF CAMBODIA
NATION RELIGION KING



**REPORT OF
SOFTWARE ENGINEERING**

GROUP: I4-GIC-A

PROJECT TITLE: *Course Enrollment System*

Name of students	ID	Score
1. HAN Raby	e20220580
2. CHEA Chanminea	e20220335
3. HENG Oulong	e20221390
4. LO Bunleang	e20220722
5. HUN Lyhorng	e20220188

Lecturer: Mr. Roeun Pacharoth

Academic year 2025-2026

Table of Contents

I. Han Raby's Responsibility

1. Configuration File.....	3
a. Cloudinary Configuration.....	
b. Global Exception Handler.....	
c. Model Mapper Configuration.....	
2. Authentication & Security.....	3
a. Authentication Controller.....	
b. Security Configuration.....	
3. Controller.....	4
a. Global Controller.....	
b. Student Controller.....	
c. Student Profile Controller.....	
4. Services.....	4
a. Image Upload Service.....	
b. Student Profile Service.....	
5. Datatransfer Object.....	5
a. Student Profile DTO.....	

II. Lo Bunleang's Responsibility

1. Controllers & Their Functions.....	5
a. Admin Classroom Controller (/admin/classrooms)	
b. Admin Schedule Controller (/admin/schedules).....	
c. Student Classroom Controller (/student/classrooms).....	
d. Student Schedule Controller (/student/schedules).....	
2. Complete System Flow.....	6
a. Admin Creates Schedule.....	
b. Student Views Schedule.....	
3. Conflict Detection.....	6
4. Key Data Object.....	7
5. Service Layer Purpose.....	7

a. Classroom Service.....
b. Schedule Service.....
c. Student Schedule Service.....
III. Heng Oulong's Responsibility	
1. Functional Tasks.....	8
2. UI Task.....	8
3. Controller.....	8
a. Admin Course Controller.....
b. Admin Enrollment Controller.....
c. Student Course Controller.....
d. Student Enrollment Controller.....
4. DTO Data Flow Summary.....	10
IV. Chea Chanminea's Responsibility	
1. Database Schema.....	11
2. Database Code Implementation.....	11
3. Admin Controller.....	14
4. Admin Dashboard.....	15
V. Hun Lyhorng's Responsibility	
1. Controller.....	16
a. Home Controller.....
b. Admin User Controller.....
2. Service Layer.....	17
3. DTO.....	17
4. UI Task.....	18
a. Student HomePage & Admin Dashboard Intergration.....
b. User Management Page (User List).....
c. Create User Interface.....

I. Han Raby's Responsibility

1. Configuration File

a. Cloudinary Configuration

📄 CloudinaryConfig.java

- Configures Cloudinary cloud storage for image uploads
- Reads API credentials from application.properties
- Provides a Cloudinary bean used by image upload services

b. Global Exception Handler

📄 GlobalExceptionHandler.java

- Handles application-wide exceptions
- Returns clean and consistent error responses
- Prevents application crashes when a resource is not found

c. Model Mapper Configuration

📄 ModelMapperConfig.java

- Provides a ModelMapper bean
- Simplifies conversion between Entity and DTO objects
- Reduces repetitive mapping code

2. Authentication & Security

a. Authentication Controller

📄 AuthController.java

- Handles user login and student registration
- Validates signup data and checks for duplicate users
- Encrypts passwords and assigns STUDENT role

b. Security Configuration

📄 SecurityConfig.java

- Configures Spring Security rules
- Defines login, logout, and role-based access control
- Restricts admin and student pages based on user roles

3. Controller

a. Global Controller

 GlobalController.java

- Makes the current logged-in user available to all views
- Retrieves user data from the security context
- Used for showing user information in templates

b. Student Controller

 StudentController.java

- Handles student home page
- Displays available courses
- Marks courses as enrolled or not enrolled for the student

c. Student Profile Controller

 StudentProfileController.java

- Manages student profile page
- Allows updating profile information
- Handles profile image upload and course withdrawal

4. Services

a. Image Upload Service

 ImageUploadService.java

- Uploads images to Cloudinary
- Returns secure image URLs
- Used for student profile image uploads

b. Student Profile Service

 StudentProfileService.java

- Contains business logic for student profiles
- Retrieves the currently logged-in user
- Handles profile updates, image upload, and enrollments

5. Data transfer Object

a. Student Profile DTO

 StudentProfileDto.java

- Transfers student profile data between view and controller
- Validates user input fields
- Prevents direct modification of User entity

II. Lo Bunleang's Responsibility

1. Controllers & Their Functions

a. Admin Classroom Controller (/admin/classrooms)

Purpose: Manage physical classrooms

Functions:

- CRUD operations for classrooms
- Track building, room number, capacity, equipment (projector/computer)
- Set status (AVAILABLE, MAINTENANCE, CLOSED)
- Uses modal forms for create/edit

b. Admin Schedule Controller (/admin/schedules)

Purpose: Manage course schedules with conflict prevention

Functions:

- Assign courses to classrooms at specific times
- Automatic conflict detection - prevent double scheduled
- Set status (SCHEDULED, CANCELLED)
- Validates time overlaps before saving
- Loads course and classroom data for dropdowns
- Uses modal forms with error handling

c. Student Classroom Controller (/student/classrooms)

Purpose: Display classrooms to students

Functions:

- View-only access to all classrooms
- See capacity, equipment, and availability status

d. Student Schedule Controller (/student/schedules)

Purpose: Show personalized weekly timetable

Functions:

- Get logged-in student from Spring Security
- Find courses student is enrolled in
- Display only active schedules (filters out CANCELLED)
- Build weekly timetable (time rows × day columns)
- Show course, classroom, and time for each class

2. Complete System Flow

a. Admin Creates Schedule

Admin clicks "Add Schedule"

- Form loads courses & classrooms
- Admin selects: Course, Classroom, Day, Time
- System validates for conflicts
- If conflict exists: Show error "Classroom already in use"
- If no conflict: Save schedule with status "SCHEDULED"
- Schedule now visible to enrolled students

b. Student Views Schedule

Student navigates to schedules

- System gets current user from login
- Finds courses student is enrolled in
- Gets schedules for those courses
- Filters out CANCELLED schedules
- Builds weekly timetable grid
- Displays personalized schedule

3. Conflict Detection

How It Works:

- Checks if classroom is already booked at overlapping times
- For CREATE: Check all existing schedules
- For UPDATE: Check all schedules except current one

- Logic: existingStart < newEnd AND existingEnd > newStart

Example:

- Existing: Monday 9:00-11:00 in Room 101
- New: Monday 10:00-12:00 in Room 101
- Result: CONFLICT DETECTED

4. Key Data Object

ClassroomDTO: Building, Room, Capacity, Equipment, Status

ScheduleDTO: Course, Classroom, Day, Time (start/end), Semester, Year, Status

TimeTableRowDTO: Time slot + Map of schedules by day (for grid display)

5. Service Layer Purpose

The service layer contains business logic and acts as a bridge between Controllers (user requests) and Repositories (database).

a. Classroom Service

Used for: Managing classroom CRUD operations

- Create, read, update, delete classrooms
- Convert ClassroomDTO ↔ Classroom entity
- Set default status to "AVAILABLE"

b. Schedule Service

Used for: Managing schedules with conflict prevention

- Create, read, update, delete schedules
- Validate conflicts - prevent double creation of classrooms
- Build weekly timetable grid structure
- Provide dashboard statistics (count schedules by status)
- Filter active schedules (exclude CANCELLED)

Key Feature: Automatically detects if a classroom is already created at the same time

c. Student Schedule Service

Used for: Providing personalized schedules for students

- Get logged-in student's identity from Spring Security

- Find courses student is enrolled in
- Filter schedules to show only student's enrolled courses
- Remove CANCELLED schedules from view
- Build personalized weekly timetable

Key Feature: Shows students only their relevant active schedules

III. Heng Oulong's Responsibility

1. Functional Tasks

- Course CRUD (Create, Read, Update, Delete)
- Student enrollment & enrollment removal
- Capacity control and duplicate enrollment prevention
- Course visibility based on status (ACTIVE / DRAFT)
- Image upload and management using Cloudinary
- Pagination for course
- Searching for courses in Admin

2. UI Tasks

- Admin course list with pagination
- Admin create/edit course forms (professional UI)
- Enrollment management UI for admin such as assigning, views, drop.

3. Controller

a. Admin Course Controller

Purpose: Allows ADMIN to fully manage courses:

- View all courses
 - Form contains: Course name, Course code, Description, Credits, Max capacity, Lecturer (dropdown), Status (ACTIVE / DRAFT) , Image upload (Cloudinary)
- Create courses
 - Business logic: Lecturer must exist, Status defaults to DRAFT if not selected, Image uploaded & URL saved, Course stored safely, Capacity <= 500, Credit <= 10

- Edit courses
 - Old image kept if no new upload
 - New image replaces old if uploaded
- Delete courses
 - Delete course from database
- Manage capacity, status, lecturer, and image

b. Admin Enrollment Controller

Purpose: Allows ADMIN to:

- View all enrollments
 - Enrollment ID
 - Student name
 - Course name
 - Status
 - Delete option
- Assign enrollments manually
 - Select student
 - Select course
 - Assign enrollment
- Delete enrollments

💡 Why Admin enrollment exists

- Manual assignment
- Administrative override
- Real school scenario

c. Student Course Controller

Purpose: Allows STUDENT to:

- View available courses: Only ACTIVE courses shown
- View course details
 - Image
 - Description
 - Lecturer

- Capacity
- Enroll button

d. Student Enrollment Controller

Purpose: Allows STUDENT to:

- Enroll themselves
 - Course name
 - Enrollment date
 - Status
 - Drop button
- View their enrollments
- Drop enrollment
 - Student can only drop their own enrollment

4. DTO Data Flow Summary

CourseCreateUpdateDto → Entity

Used for:

- Create course
- Update course

Contains:

- Basic fields
- Status
- Image file

Course → CourseDto

Used for:

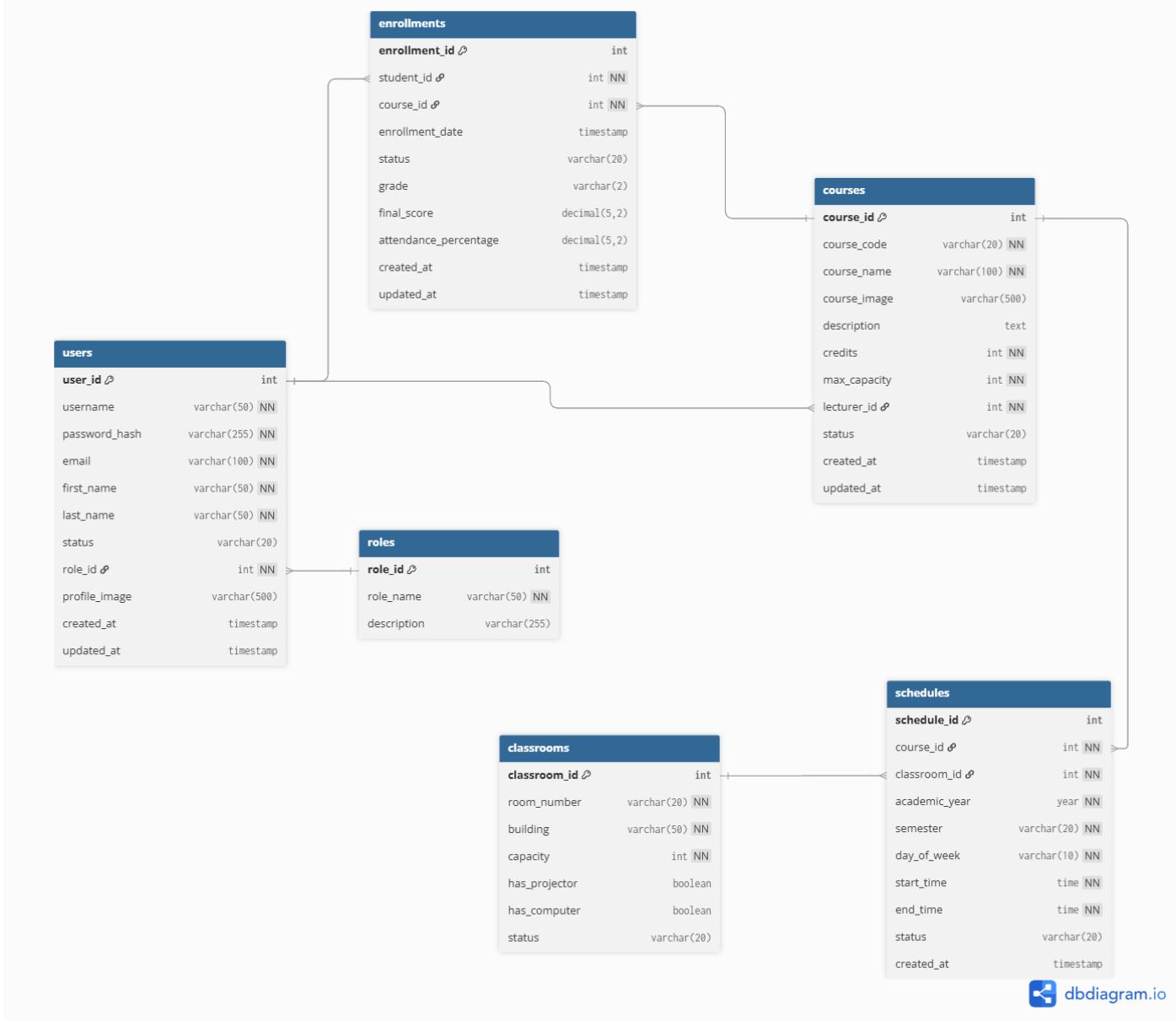
- Admin UI
- Student UI

Contains:

- Display-safe fields
- Computed fields (capacity)

IV. Chea Chanminea's Responsibility

1. Database Schema



2. Database Code Implementation

a. Overall Design Approach

- Used **JPA/Hibernate** with modern Jakarta Persistence annotations
- Followed **clean entity structure** with separation of concerns:
 - Core attributes
 - Relationships (bidirectional where appropriate)
 - Business/helper methods
 - Lifecycle callbacks

- Proper indexing & unique constraints
- Implemented **soft status management** instead of hard deletes (most entities have status field)
- Used **enums** for type-safe status/role values (RoleName, UserStatus, EnrollmentStatus, CourseStatus)

b. Entity Summary

➤ User

- Full personal info + DOB validation
- Profile image support
- Bidirectional with Role, Courses (lecturer), Enrollments (student)
- Lifecycle timestamps
- Full name helper

Important Decision

- @Past on DOB
- Unique on email & password_hash
- Indexes on role & status

➤ Role

- Enum-based role names
- Bidirectional with Users
- Simple description field

Important Decision

- Unique constraint on role_name
- Cascade ALL (careful usage)

➤ Course

- Lecturer assignment (ManyToOne)
- Schedules & Enrollments (OneToMany)
- Capacity management
- Current enrollment count calculation

- `isFull()` business method

Important Decision

- Unique `course_code`
- Business logic in entity (`getCurrentEnrollmentCount`, `isFull`)
- Cascade ALL on collections

➤ **Classroom**

- Composite unique constraint (`building + room_number`)
- Equipment flags
- Status management
- Bidirectional with Schedules
- Helper methods: `getFullLocation()`, `isAvailable()`, `hasEquipment()`

Important Decision

- Multiple indexes for search performance
- Composite unique very important for real-world usage

➤ **Schedule**

- Very strong double-booking prevention with two unique constraints
- Time overlap checking method
- Classroom capacity & equipment validation methods (placeholder)
- Many useful business methods (`overlapsWith`, `isValidTimeRange`, `getDurationInHours...`)

Important Optimization

- Most important entity for business rules
- Two critical unique constraints
- `DayOfWeek` enum usage

➤ **Junction Table Enrollment**

- Composite unique (`student + course`)
- Rich grade & performance tracking

- Multiple statuses (PENDING → ENROLLED → COMPLETED)
- Automatic enrollment date
- Bidirectional with User & Course

Good Optimization

- Very good status lifecycle management
- Precision/scale on decimal fields
- Indexes on both sides of relationship

c. Important Technical Decisions & Best Practices Applied

- Proper use of FetchType.LAZY everywhere → performance oriented
- Correct cascade types (mostly CascadeType.ALL – be careful in production)
- Good usage of @PrePersist / @PreUpdate for audit fields
- Meaningful business methods inside entities (recommended for domain-driven design)
- Strong constraint strategy:
 - Unique constraints at database level (not just application)
 - Composite unique constraints where it matters most (classroom booking, student-course)
- Strategic indexing on frequently filtered/searched columns
- Consistent naming convention between Java fields and database columns
- Used String-based enums (@Enumerated(EnumType.STRING)) → more readable in DB
- Helper methods for common operations (getFullName, getFullLocation, isFull, overlapsWith...)

3. Admin Controller

- Mapped routes for dashboard (/admin/adminDashboard), classrooms (/classrooms, /classrooms/create), schedules (/schedules, /schedules/create).

- Autowired services (UserDetailsService, CourseService, EnrollmentService, ScheduleService) to fetch stats.
- Added model attributes for stats (e.g., totalUsers, pendingRequests, totalCourses) and activePage for navigation.
- Fetched latest enrollment for display.

4. Admin Dashboard

a. AdminDashboard.html

- Layout with sidebar inclusion, top bar (header with logout), stats grid, and action cards.
- Dynamic stats display using Thymeleaf (e.g., \${totalCourses}, fallback to 0).
- Three stat cards (Courses, Users, Schedules) with icons, colors, and sub-metrics.
- Action cards for quick navigation (Add Classes, Add Course, Add Schedule) with gradients and buttons.
- JavaScript for interactive hover effects on stat cards.

b. AdminSidebar.html

- Reusable fragment with logo, dynamic navigation links (Overview, Users, Courses, Enrollments, Classes, Schedules).
- Active page highlighting using Thymeleaf conditionals.
- Admin profile section with dynamic user data (avatar, name, email) from \${currentUser}.
- Gradient background and responsive flex layout.

c. Important Technical Decisions & Best Practices Applied

- Thymeleaf Integration: Used for dynamic content (e.g., th:text, th:href, th:classappend) to bind backend data securely, preventing XSS via escaping.
- Spring MVC Patterns: Controller handles GET mappings only (for read operations); autowired services promote separation of concerns (controller focuses on routing, services on business logic).

- UI/UX Focus: Tailwind for rapid styling; custom CSS for transitions and hovers; icons enhance usability; gradients create a modern, branded look.
- Data Handling: Null checks and fallbacks (e.g., `${totalCourses != null ? totalCourses : 0}`) ensure robust rendering; stats fetched via services for real-time accuracy.
- Security/Usability: Logout link in header; dynamic user info from session (`${currentUser}`); active page tracking for intuitive navigation.
- Performance: Lazy loading implied via services; no heavy computations in controller; CSS/JS minimized for fast load.
- Error Prevention: Conditional rendering avoids NPEs; enums ensure consistent status queries.

V. Hun Lyhorng's Responsibility

1. Controller

a. Home Controller

After a user logs in successfully, the system:

- Detects the user's role
- Redirects them to the correct homepage
 - Admin → Admin Dashboard
 - Student → Student Home page

b. Admin User Controller

- Implemented an AdminUserController to handle all HTTP requests related to user management.

Defined routes for:

- Displaying the list of all users
- Showing the create user form
- Creating a new user
- Activating or deactivating a user
- Deleting a user

- Applied role-based access control, ensuring that only users with the ADMIN role can access these endpoints.
- Used model attributes to pass user data, role lists, and status lists from the backend to the UI.
- Managed page state (such as active sidebar menu) to improve navigation and user experience

2. Service Layer

- Implemented a UserService to handle all business logic related to users.
- The service performs:
 - Validation to ensure username and email uniqueness
 - Password encryption using BCrypt before saving users to the database
 - Assignment of roles and user status during user creation
 - Safe toggling of user status between ACTIVE and INACTIVE
 - Deletion of users from the database
- Centralized business logic in the service layer to maintain clean separation of concerns and improve maintainability.

3. DTO

- Used DTOs to prevent direct exposure of the User entity to the UI.
- Implemented:
 - UserCreateDto to capture form input data such as username, email, password, role, and status when creating a user.
 - UserListDto to display user information in the admin dashboard without exposing sensitive data such as password hashes.

4. UI Tasks

a. Student HomePage & Admin Dashboard Integration

- Integrated the User Management module into an existing Admin Dashboard layout.
- Reused a common sidebar, header, and footer across admin pages to ensure UI consistency.
- Highlighted the active menu item dynamically based on the current page.

b. User Management Page (User List)

- Designed a responsive user list table using Tailwind CSS.
- Displayed key user information:
 - Profile image
 - Username
 - Full name
 - Email
 - Role
 - Status
- Used visual indicators (colored badges) to clearly represent user roles and statuses.
- Added action buttons to:
 - Activate or deactivate users
 - Delete users with confirmation prompts
- Ensured sensitive information such as passwords is never displayed.

c. Create User Interface

- Implemented a Create User form accessible only to administrators.
- The form allows input of:
 - Username
 - Email
 - Password
 - Role

- Status
 - Applied Tailwind CSS to create a clean, modern, and user-friendly design.
 - Ensured proper form submission and redirection back to the user list after successful creation.
 - Designed the UI to align visually with the admin dashboard theme.