COMP3030 - Final Summary Report

Vu Ai Thanh Truong Gia Bao Nguyen Tuan Anh

22-05-2025

Table of contents

1 Conceptual & Physical Design

The project is aimed to provide a yet minimal but still functional of a management application for a education institution. The main actors that interacts to the app contains: `Administrators`, `Instructors`, and `Students`. There activities will go around the `Courses` at school.

1.1 Functional and Non-functional Designs

1.1.1 Functional

1.1.1.1 User Authentication and Authorization:

- The system must allow users (students, instructors, and admins) to log in using email and password credentials.
- Users must be assigned roles (student, instructor, admin) with role-specific access to features (e.g., students can enroll/unenroll, instructors can view/manage courses, admins can manage users and courses).

1.1.1.2 Course Management

- The system must display a list of courses with filters by course name, department, and schedule, supporting pagination (10 courses per page).
- Users can view detailed course information (e.g., ID, name, department, instructor, location, schedule, semester, availability) on a course detail page.
- Instructors and admins can add, edit, and delete courses, while students can only view course details.

- The system must track and display enrolled students for each course.

1.1.1.3 Enrollment Management:

- Students must be able to enroll in a course if availability is greater than zero and they are not already enrolled, with a confirmation dialog before submission.
- Students must be able to unenroll from a course if they are enrolled, with a confirmation dialog before submission.

- The system must update course availability automatically when students enroll or unenroll.
- The system must prevent enrollment if the course is full (availability ≤ 0) or if the student is already enrolled.

1.1.1.4 User Interface and Navigation:

- The system must provide a responsive web interface using Bootstrap for consistent styling across devices.
- Each role (student, instructor, admin) must have a dedicated dashboard with role-specific options.
- Flash messages must be displayed to provide feedback on actions (e.g., success or error messages for login, enrollment).

1.1.1.5 Analytics

- For management roles like `Administrators` and `Instructors`, they will have access to several dashboards to report about the courses at the institution, or the courses that the instructors are involving in.

1.1.2 Non-functional

- **Performance**:
  - Having high uptime, with all errors are being handled to prevent crashes.
  - Queries should execute within a reasonable time, with minimal complexity. Current target is 0.5 seconds for a dataset of 1000 enrollments.
- **Security**:
  - Passwords are encrypted when being stored in the database, with supports of password hashing function `bcrypt`.
  - Access control systems based on roles are designed properly, prevent unauthorized information access or disclosure.
  - Preventions or mitigations of common web security vulnerabilities as well as applications using SQL database system: SQL Injection, Cross-site Scripting, etc.
- **Scalability**:
  - The application is scalable with increasing numbers of users in an acceptable level, against around 300 users under peak time.
- **Usability**:
  - Provide easy-to-navigate interface, with responsive User Interface with `Bootstrap` library.
- **Reliability**:
  - Ensure data integrity with normalized database design (3NF) and constraints.

1.2 Entity-Relationship Diagram

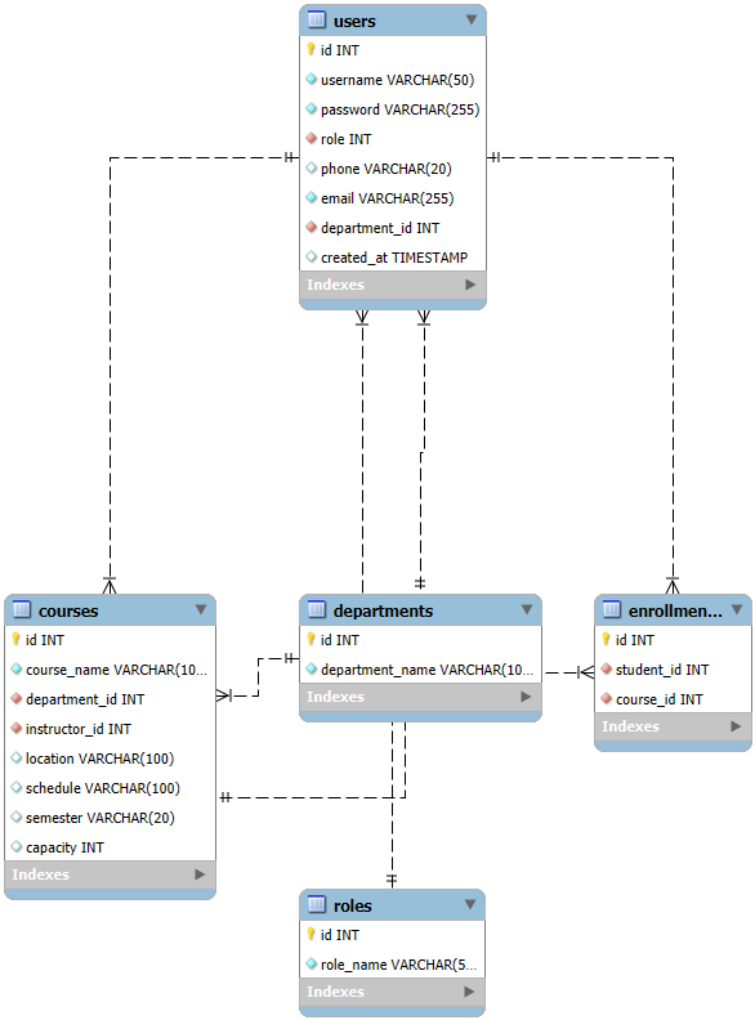

Figure 1: Entity-Relationship Diagram of the project

1.3 Proof of Normalization Form

1.3.1 First Normal Form (1NF):

- All tables have primary keys
- All attributes are atomic
- No repeating groups

1.3.2 Second Normal Form (2NF):

- All non-key attributes are fully functionally dependent on their primary keys
- No partial dependencies exist

1.3.3 Third Normal Form (3NF):

- No transitive dependencies
- Proper use of foreign keys for references
- Related data is properly normalized into separate tables

2 Implement of DB Entities

2.1 Database schema

The full script is in our GitHub repository at this link.

2.2 Views

Currently, there is one view `view_course_details` in the database. It provides a simplified and denormalized view of course information with instructor details. This view joins `courses` and `users` tables to provide course information along with the instructor's name, making it easier to retrieve complete course information without writing complex joins in application queries.

```
CREATE VIEW view_course_details AS
SELECT
    c.id AS course_id,
    c.course_name,
    c.department_id,
    c.instructor_id,
    u.username AS instructor_name,
    c.location,
    c.schedule,
    c.semester,
    c.availability
FROM
    courses c
    LEFT JOIN users u ON c.instructor_id = u.id;
```

2.3 Stored Procedures

There are 03 stored procedures within the database, which are used as quick refrence for most used queries within the lifecycle of the application:

Stored Procedures	Purposes
`get_user_role_counts()`	Collect number of users across roles within the databases, adaptive with future new roles.
`get_course_department_counts()`	Collect number of courses across departments, adaptive with future new departments & courses in the institute.
`get_student_department_counts()`	Collect number of students across departments, adaptive with future new departments, or changing numbers of students
`get_course_count_by_semester()`	Filter number of courses in each semesters

2.4 Triggers

In the functionalities of the application, there are features that automatically update available slots in each course after enrollment. To efficiently operate this feature, we introduced two triggers:

- `before_enrollment_insert()`: This trigger is used to check if there are enough seats left in the course. Whenever a student registered a course, this trigger will reduce the available slots by one. When there is no seat available, it will raise an exception for the business logic to know and handle.
- `after_enrollment_delete()`: In several cases, student may want to drop the course. This trigger simply increase one seat for the course whenever a student unenroll it.

Due to the atomic & synchronize features of MySQL, there are no race con