RAMON MAGSAYSAY MEMORIAL COLLEGES
Information Technology Education Program
1st SEMESTER: AY: 2025 - 2026

NAME: _____ SCHEDULE: _____ SCORE: _____
SUBJECT: **WEB SYSTEMS AND TECHNOLOGIES** INSTRUCTOR: _____ DATE: _____

LABORATORY EXERCISE 6
COURSE ENROLLMENT SYSTEM

Learning Objectives

By the end of this laboratory exercise, students should be able to:

- Design and create a new database table to manage relationships between users and courses.
- Implement server-side logic for handling course enrollments.
- Display user-specific data (enrolled courses) in a dashboard.
- Utilize jQuery and AJAX to create a dynamic, seamless user experience without page reloads.
- Understand and implement basic foreign key relationships in a web application.

Prerequisite student experiences and knowledge

Before starting this exercise, students should have:

- Completed Laboratory Exercise 5 (Admin and Student Dashboards).
- A solid understanding of the MVC architecture in CodeIgniter.
- Proficiency in writing database queries using CodeIgniter's Query Builder.
- Basic knowledge of SQL relationships (one-to-many).
- Familiarity with jQuery syntax and the concept of AJAX.
- Ability to create and style front-end components with Bootstrap.

Background

A core feature of any Learning Management System (LMS) is the ability for students to enroll in available courses. This involves creating a relationship between the **users** table (students) and the **courses** table. This relationship is typically stored in a pivot table. To enhance user experience, the enrollment process should be dynamic, allowing students to join courses without refreshing the page. This is achieved using jQuery AJAX to send a request to the server in the background, providing immediate feedback to the user.

Materials/Resources

- Personal Computer with Internet Access
- XAMPP/WAMP/LAMP server installed
- CodeIgniter Framework (latest version)
- Visual Studio Code or any code editor
- Git and GitHub Account
- Web Browser (Chrome, Firefox, etc.)

Laboratory Activity

Step 1: Create a Database Migration for the Enrollments Table

1. Create a new migration file for the **enrollments** table.
   Run: php spark make:migration CreateEnrollmentsTable

2. Open the newly created file in app/Database/Migrations/.
3. In the up() method, define the table with the following fields:
   - ✓ id (primary key, auto-increment)
   - ✓ user_id (int, foreign key to **users** table)
   - ✓ course_id (int, foreign key to **courses** table)
   - ✓ enrollment_date (datetime)
4. In the down() method, define how to drop the table.
5. Run the migration: php spark migrate.

Step 2: Create the Enrollment Model

1. Navigate to app/Models/ and create a file named EnrollmentModel.php.
2. Create a model class with methods to:
   - ✓ enrollUser($data): Insert a new enrollment record.
   - ✓ getUserEnrollments($user_id): Fetch all courses a user is enrolled in.
   - ✓ isAlreadyEnrolled($user_id, $course_id): Check if a user is already enrolled in a specific course to prevent duplicates.

Step 3: Modify the Course Controller

1. Open your Course.php controller (or create it if it doesn't exist).
2. Add a new method, enroll(), to handle the AJAX request.
   - ✓ This method should:
   - ✓ Check if the user is logged in.
   - ✓ Receive the **course_id** from the POST request.
   - ✓ Check if the user is already enrolled.
   - ✓ If not, insert the new enrollment record with the current timestamp.
   - ✓ Return a JSON response indicating success or failure.

Step 4: Update Student Dashboard View

1. Open/Check the student dashboard view file.
2. Create a section to **Display Enrolled Courses**. Use a Bootstrap list group or cards to iterate over and display the courses returned by **EnrollmentModel::getUserEnrollments()**.
3. Create another section for **Available Courses**. Display a list of courses with an **Enroll** button next to each.

Step 5: Implement AJAX Enrollment

1. In the **Available Courses** section of the dashboard, add a **data_course_id** attribute to each **Enroll** button containing the specific course ID.
2. Include the jQuery library in your view if it's not already included.
3. Write a jQuery script that:
   - ✓ Listens for a click on the **Enroll** button.
   - ✓ Prevents the default form submission behavior.

- ✓ Uses **$.post()** to send the **course_id** to the /course/enroll URL.
- ✓ On a successful response from the server:
- ✓ Displays a Bootstrap alert message.
- ✓ Hides or disables the **Enroll** button for that course.
- ✓ Updates the **Enrolled Courses** list dynamically without reloading the page.

Step 6: Configure Routes

1. Update app/Config/Routes.php to include a route for the enrollment action.
   $routes->post('/course/enroll', 'Course::enroll');

Step 7: Test the Application Thoroughly

1. Log in as a student.
2. Navigate to the student dashboard.
3. Click the **Enroll** button on an available course and verify:
   - The page does not reload.
   - A success message appears.
   - The button becomes disabled or disappears.
   - The course appears in the **Enrolled Courses** list.

Step 8: Push to GitHub

1. Commit your changes with a descriptive message.
2. Push your changes to your GitHub repository.

Step 9: Vulnerable Checking

1. Test for Authorization Bypass
   - ❖ Log out of the application and attempt to directly access the enrollment endpoint via Postman or browser console by sending a POST request to /course/enroll with a course_id parameter.
   - ❖ Verify that the server returns an unauthorized error instead of processing the enrollment.
2. Test for SQL Injection
   - ❖ While logged in, use browser developer tools to modify the AJAX request and change the course_id value to 1 OR 1=1.
   - ❖ Check if the application properly validates the input and prevents SQL injection attacks.
3. Test for CSRF (Cross-Site Request Forgery)
   - ❖ Check if your enrollment form includes CSRF protection tokens.
   - ❖ Verify that CodeIgniter's CSRF protection is enabled in app/Config/Security.php.
   - ❖ Attempt to make an enrollment request without a valid CSRF token and confirm it is rejected.
4. Test for Data Tampering

- ❖ As a student, try to enroll another user in a course by modifying the user ID in the request.
- ❖ Verify that the server-side code uses the logged-in user's session ID rather than trusting client-supplied user IDs.

5. Test for Input Validation
   - ❖ Attempt to enroll in non-existent courses by sending invalid course_id values.
   - ❖ Verify that the application properly validates that the course exists before creating an enrollment.

Output / Results

- ✓ Screenshot of your database's **enrollments** table structure (phpMyAdmin or equivalent).
- ✓ A screenshot of the student dashboard showing the **Available** and **Enrolled Courses** sections is attached.
- ✓ A screenshot of the browser's developer tools (Network tab) shows the successful AJAX POST request and response when enrolling in a course.
- ✓ A screenshot of the GitHub repository with the latest commit for this exercise.

RAMON MAGSAYSAY MEMORIAL COLLEGES
Information Technology Education Program
1st SEMESTER: AY: 2025 - 2026

NAME: _____ SCHEDULE: _____ SCORE: _____
SUBJECT: **WEB SYSTEMS AND TECHNOLOGIES** INSTRUCTOR: _____ DATE: _____

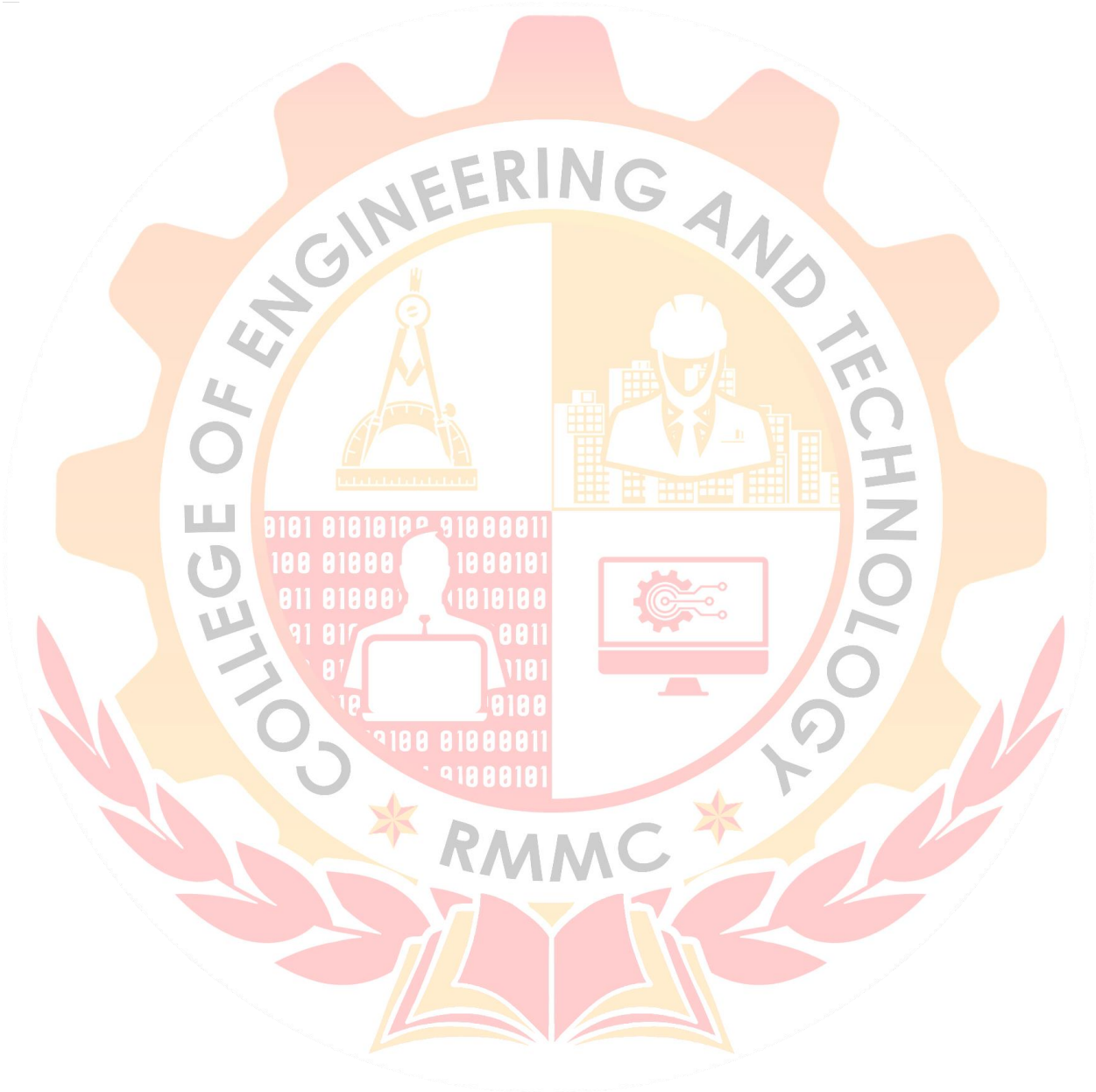
QUESTIONS:

1. What is the purpose of the **enrollments** table? Why is it necessary, instead of just adding a **course_id** column to the **users** table?
2. Explain the role of the **isAlreadyEnrolled()** method in the Model. What potential issue does it prevent?
3. Describe the client-side and server-side steps when students click the **Enroll** button until they receive confirmation.

RAMON MAGSAYSAY MEMORIAL COLLEGES
Information Technology Education Program
1st SEMESTER: AY: 2025 - 2026

NAME: _____ SCHEDULE: _____ SCORE: _____
SUBJECT: **WEB SYSTEMS AND TECHNOLOGIES** INSTRUCTOR: _____ DATE: _____

Output / Results



RAMON MAGSAYSAY MEMORIAL COLLEGES
Information Technology Education Program
1st SEMESTER: AY: 2025 - 2026

NAME: _____ SCHEDULE: _____ SCORE: _____
SUBJECT: **WEB SYSTEMS AND TECHNOLOGIES** INSTRUCTOR: _____ DATE: _____

Conclusion