# 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

1. Open the newly created file in app/Database/Migrations/.
2. 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)
3. In the down() method, define how to drop the table.
4. 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.

**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?

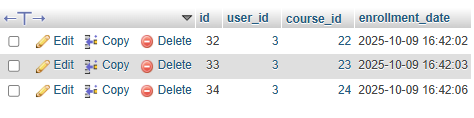
The enrollments table connects students and courses. It saves the user\_id, course\_id, and the time when the student enrolled. This table is needed because one student can join many courses, and one course can have many students. A single course\_id in the users table is not enough to handle this kind of many-to-many relationship or to save details like status and enrollment date.

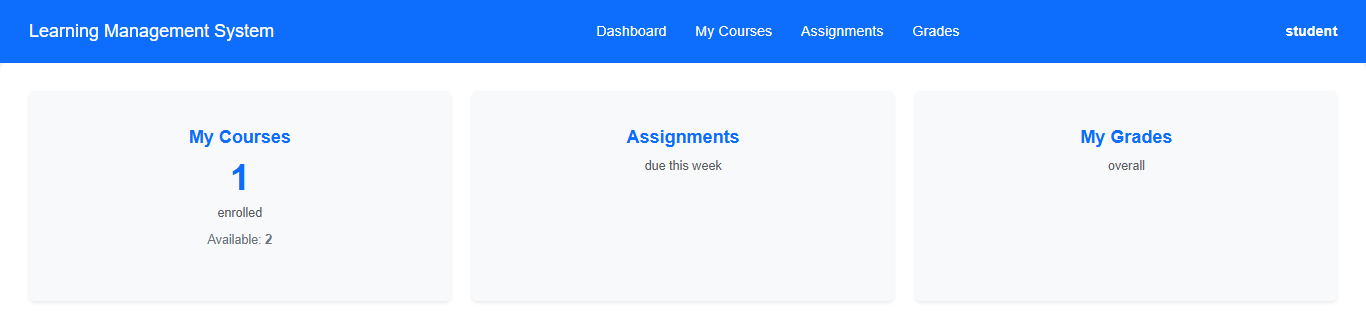
1. Explain the role of the **isAlreadyEnrolled()** method in the Model. What potential issue does it prevent?

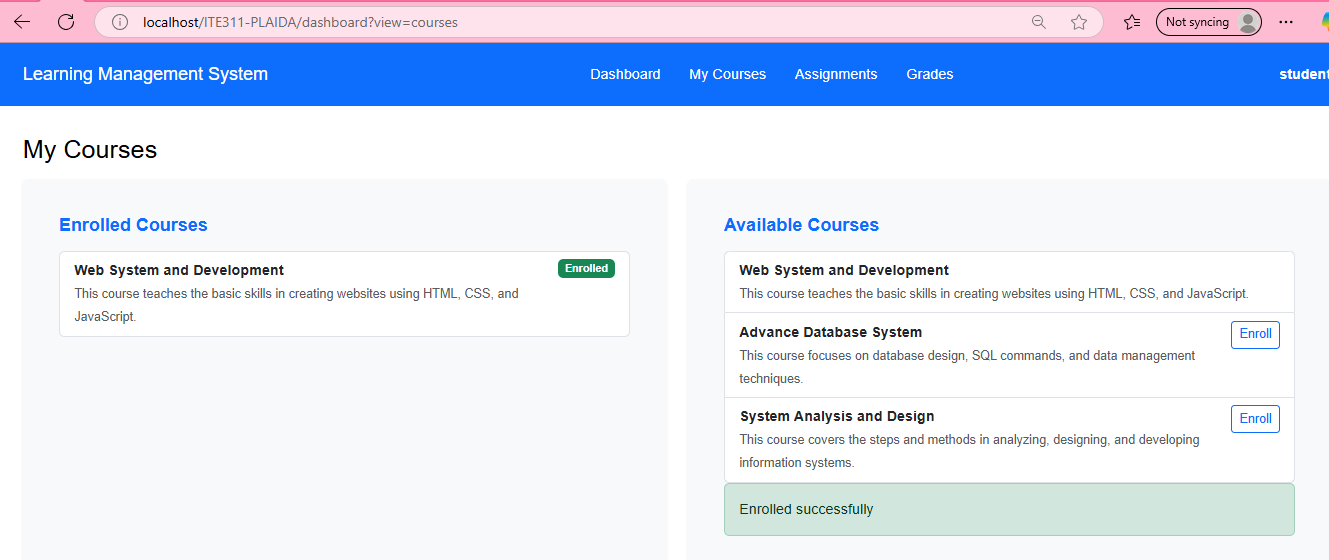
The **isAlreadyEnrolled()** method is used to check if a student is already enrolled in a course before adding a new one. This helps stop problems like enrolling twice when the student clicks the Enroll button many times or when requests happen at the same time. It also lets the system show a clear message like “already enrolled” instead of showing a database error.

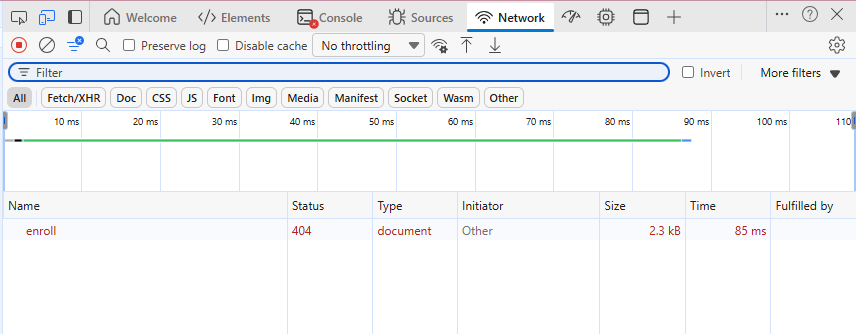
1. Describe the client-side and server-side steps when students click the **Enroll** button until they receive confirmation.

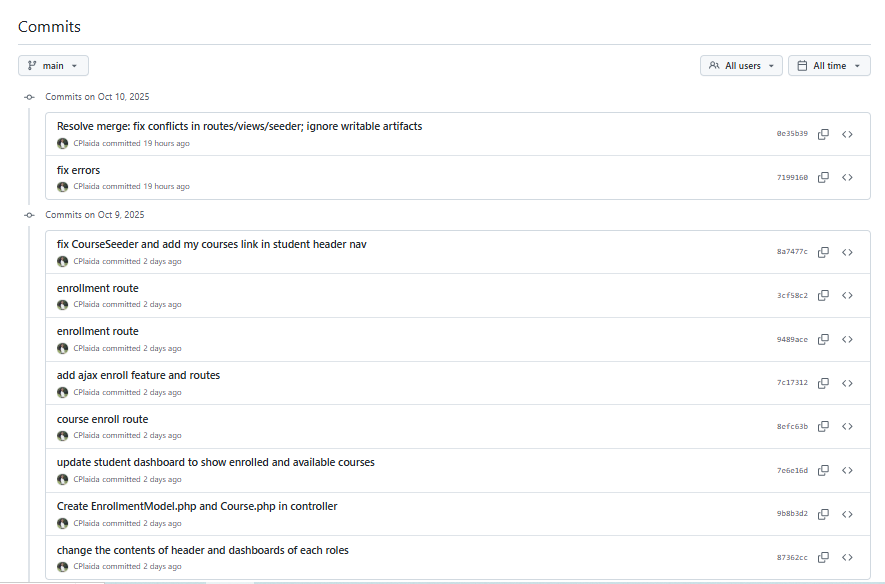
When a student clicks Enroll, the button turns off, and a safe POST request with the course\_id is sent to the enroll route. The controller checks if the student is logged in, makes sure the course\_id is valid, and that the course exists. It then uses isAlreadyEnrolled() to stop duplicate enrollments. If the student is not yet enrolled, it adds a new record in the enrollments table and sends a success message. The page then shows a confirmation and changes the button to “Enrolled”.

**Output / Results**







****

**Conclusion**

In Lab 6, I learned how to manage course enrollment using an enrollments table that shows the connection between students and courses. We used isAlreadyEnrolled() to stop double enrollment and a unique (user\_id, course\_id) rule to keep the database correct. When a student clicks Enroll, the system checks if the student is logged in, if the course is valid, and if they are already enrolled before adding a new record. It then gives a clear message.