# LABORATORY EXERCISE 7

# FILE UPLOADS (COURSE MATERIALS)

**Learning Objectives**

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

* Design and implement a database schema for managing file uploads related to courses.
* Utilize CodeIgniter's File Uploading Library to handle file uploads securely.
* Create an administrative interface for uploading and managing course materials.
* Implement access control to ensure only enrolled students can download materials.
* Enhance the user interface with Bootstrap for a clean and functional file management system.

**Prerequisite student experiences and knowledge**

Before starting this exercise, students should have:

* Completed Laboratory Exercise 6 (Course Enrollment System).
* A solid understanding of CodeIgniter's MVC structure and database operations.
* Experience with HTML forms and Bootstrap styling.
* Basic knowledge of handling file inputs in HTML.
* Familiarity with user authentication and session management in CodeIgniter.

**Background**

A core feature of any Learning Management System (LMS) is the distribution of learning resources. This involves allowing instructors to upload various file types (such as PDFs, PowerPoint presentations, and documents) and making them available to enrolled students. CodeIgniter provides a robust File Uploading Library that simplifies the process of validating, uploading, and securing files. This exercise will integrate this functionality into the LMS, ensuring that file management is seamless and secure.

**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 Materials Table**

1. Create a new migration file for the materials table.

Run: **php spark make:migration CreateMaterialsTable**

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)
* course\_id (INT, Foreign Key referencing the `courses` table)
* file\_name (VARCHAR(255)) to store the original filename
* file\_path (VARCHAR(255)) to store the path to the uploaded file
* created\_at (DATETIME)

1. In the **down() method**, define how to drop the table.

Run the migration: php spark migrate

**Step 2: Create a Model for Materials**

1. Navigate to app/Models/ and create a file named **MaterialModel.php**.
2. Create a model class with methods to:

* **insertMaterial($data)** Insert a new material record.
* **getMaterialsByCourse($course\_id)** Get all materials for a specific course.

**Step 3: Create or Modify a Controller for Materials**

1. You can create a new Materials.php controller or add methods to an existing admin controller.
2. Add the following methods:

* **upload($course\_id)** Displays the file upload form and handles the file upload process.
* **delete($material\_id)** Handles the deletion of a material record and the associated file.
* **download($material\_id)** Handles the file download for enrolled students.

**Step 4: Implement File Upload Functionality**

1. In the **upload($course\_id**) method, check for a POST request.
2. Load CodeIgniter's File Uploading Library and Validation Library.
3. Configure the upload preferences (upload path, allowed file types, maximum file size).
4. Perform the file upload. If successful, prepare data (course\_id, file\_name, file\_path) and save it to the database using the **MaterialModel**.
5. Set a flash message indicating success or failure and redirect back to the course management page.

**Step 5: Create the File Upload View**

1. Create a view file.
2. The view should contain a form with the **enctype="multipart/form-data"** attribute and a file input field.
3. Style the form using Bootstrap classes.

**Step 6: Display Downloadable Materials for Students**

1. In the student dashboard or a dedicated course view, use the **MaterialModel** to fetch materials for the courses the student is enrolled in.
2. Create a view that lists these materials, displaying the file name and a download button/link for each.
3. The download link should point to the **download($material\_id)** method in your controller.

**Step 7: Implement the Download Method**

1. In the **download($material\_id)** method:

* Check if the current user is logged in and enrolled in the course associated with the material.
* Retrieve the file path from the database using the `$material\_id`.
* Use CodeIgniter's download() helper function or the Response class to force the file download securely.

**Step 8: Update Routes**

1. Update app/Config/Routes.php to include routes for the new functionalities.

**$routes->get('/admin/course/(:num)/upload', 'Materials::upload/$1');**

**$routes->post('/admin/course/(:num)/upload', 'Materials::upload/$1');**

**$routes->get('/materials/delete/(:num)', 'Materials::delete/$1');**

**$routes->get('/materials/download/(:num)', 'Materials::download/$1');**

**Step 9: Test the Application**

1. Run the application and test the complete flow:

* Log in as an admin/instructor.
* Navigate to a course and upload a file (PDF, PPT).
* Verify the file is saved in the designated folder and a record is added to the `materials` table.
* Log in as a student enrolled in the course.
* Navigate to the course page and verify the material is listed.
* Test the download functionality.
* Test accessing the download link while not enrolled (it should be restricted).

**Step 9: Push to GitHub**

1. Add, commit, and push your changes to your GitHub repository.

Output / Results

* Screenshot of the **materials** table schema from your database (phpMyAdmin or equivalent).
* Screenshots of the file upload form (admin side) and the student view showing the list of downloadable materials.
* Screenshot of the server's file system (upload directory) showing the uploaded file.
* A screenshot of the GitHub repository with the latest commit for this exercise.

**QUESTIONS:**

1. What are the security risks associated with file uploads, and how did you mitigate them using CodeIgniter's File Uploading Library?

**Security Risks:**

* Malicious files: Attackers can upload executable or script files (e.g., .php, .exe) to compromise the server.
* File overwriting: Uploaded files may overwrite existing ones if not properly renamed.
* Large file uploads: Oversized files can cause denial of service (DoS) or fill up server storage.
* Path traversal attacks: Attackers can try to manipulate the file path to store files outside the intended directory.

**Mitigation using CodeIgniter’s File Uploading Library:**

* File type validation: Restrict uploads to specific file extensions using $config['allowed\_types'] = 'jpg|png|pdf|docx';
* File size limitation: Set a maximum size (e.g., $config['max\_size'] = 2048;) to prevent large uploads.
* File name encryption: Use $config['encrypt\_name'] = TRUE; to generate random file names and prevent overwriting or path guessing.
* Upload path restriction: Store files only in safe, predefined directories (e.g., WRITEPATH . 'uploads/').
* Server-side validation: CodeIgniter automatically checks the uploaded file’s MIME type, ensuring it matches the expected type.

1. Explain the purpose of the **enctype="multipart/form-data"** attribute in the form tag for file uploads.

The enctype="multipart/form-data" attribute tells the browser to **encode the form data properly when files are being uploaded**.  
Without this attribute, file input data is not sent to the server correctly.

**Purpose:**

* It divides the form data into multiple parts so both text fields and binary data (like images or PDFs) can be sent.
* It ensures the uploaded file is transmitted in its raw binary format rather than as plain text.

Example:

<form action="/upload" method="post" enctype="multipart/form-data">

<input type="file" name="userfile">

<button type="submit">Upload</button>

</form>

Without multipart/form-data, the file content will **not reach the server**, and the upload will fail.

1. Why is it important to check if a student is enrolled in a course before allowing them to download a material? How does this enforce application security?.

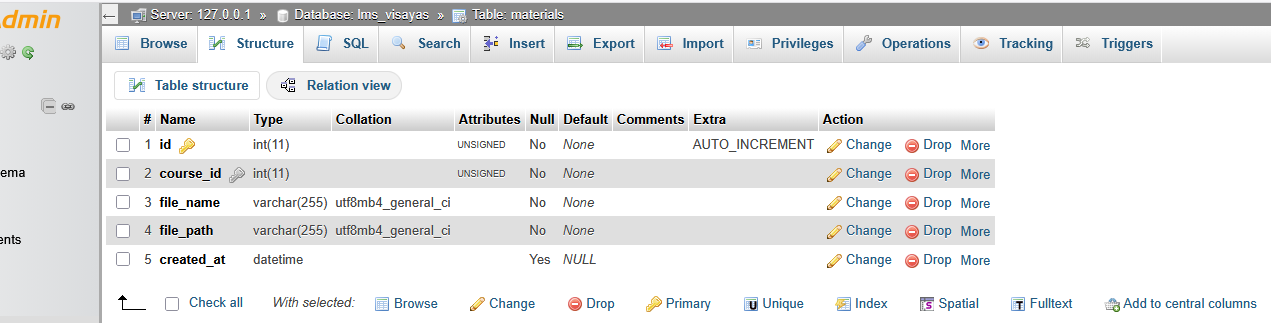
**It is important because:**

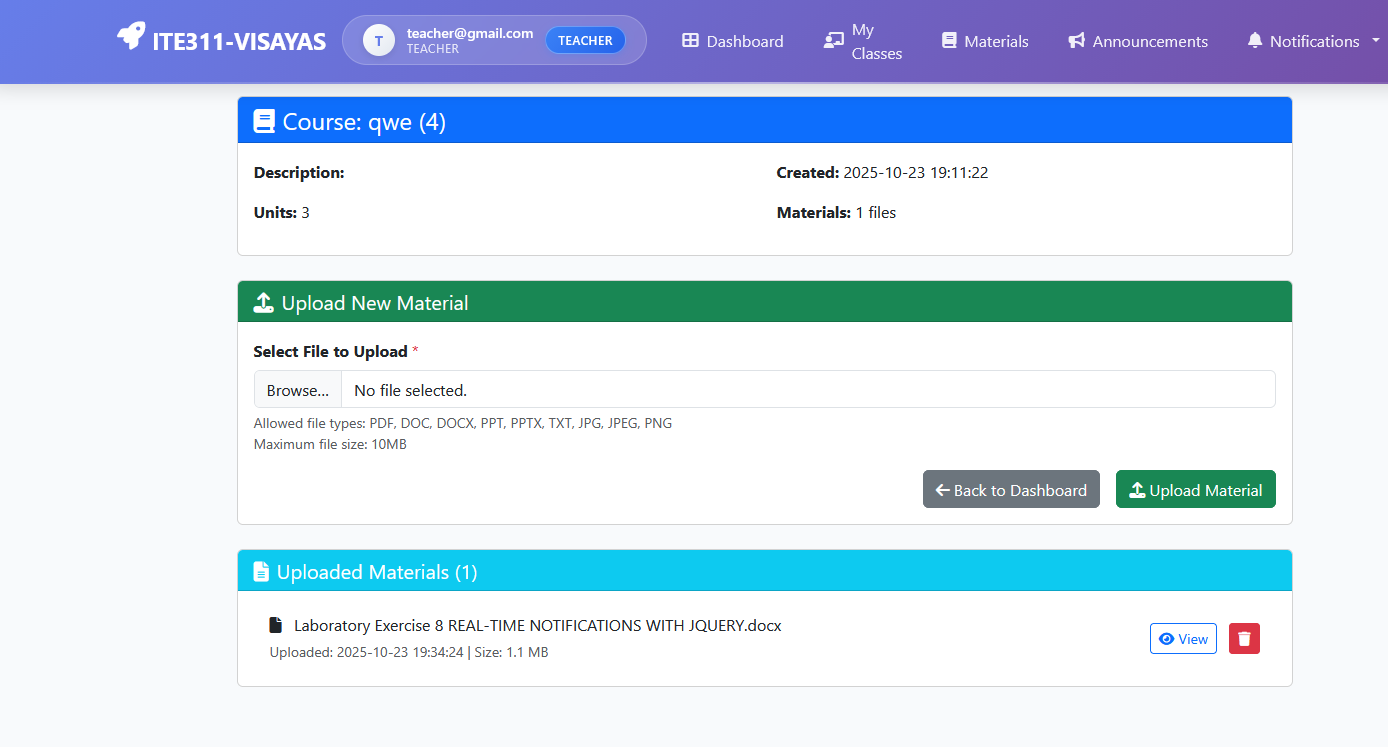
* Access control: Only authorized (enrolled) students should have access to course materials.
* Prevents data leaks: Unauthorized users could access and distribute course content illegally.
* Protects intellectual property: Course materials belong to instructors or the institution.
* Maintains system integrity: Prevents users from exploiting download endpoints or guessing file URLs to access restricted files.

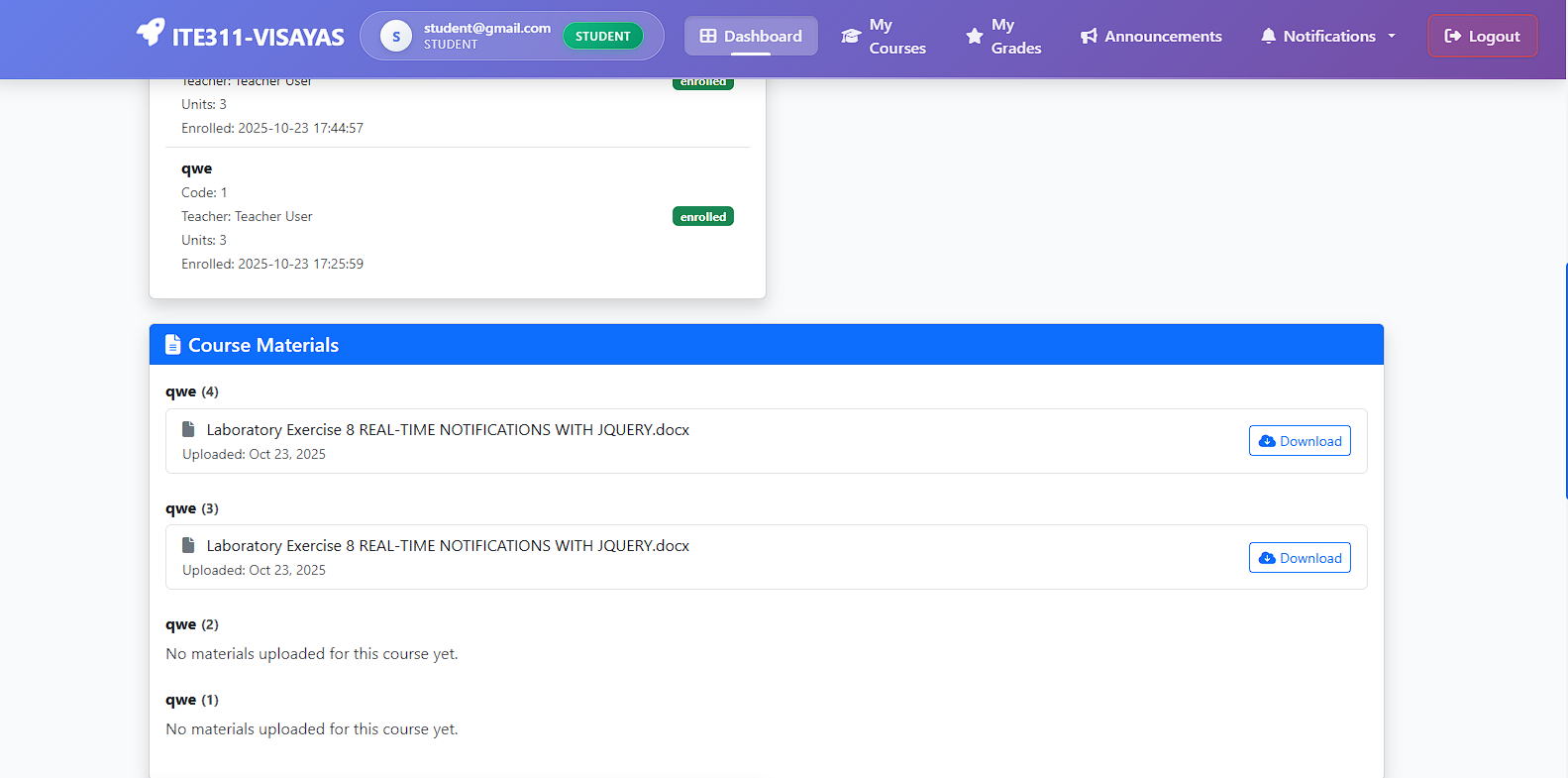
**Security Enforcement:**

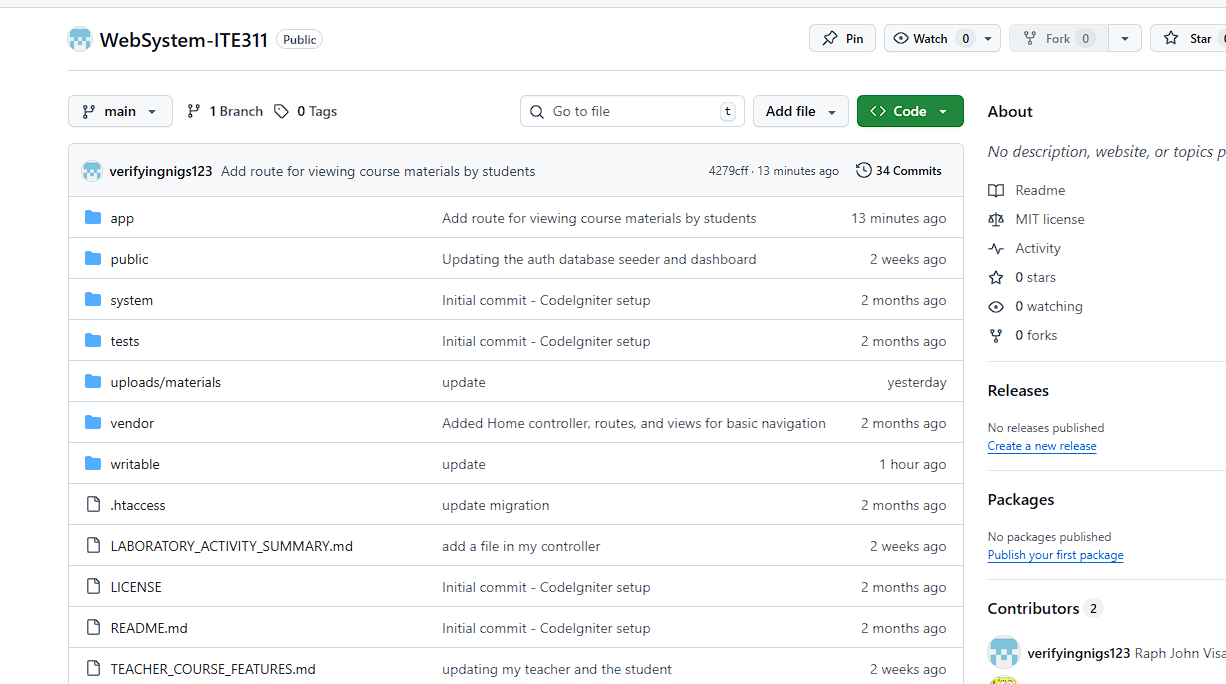
* Before allowing a file download, the application checks if the student\_id exists in the enrollments table for that course\_id.
* If not enrolled, the system denies access (e.g., redirects or shows a “403 Forbidden” message).
* This ensures authorization—verifying that users not only have valid accounts but also have permission to access that specific resource.

**Output / Results**









**Conclusion**

implementing secure file uploads and proper access control is essential in maintaining the integrity and safety of any web application. By using CodeIgniter’s File Uploading Library, developers can effectively mitigate common risks such as malicious uploads, file overwrites, and unauthorized access. The use of enctype="multipart/form-data" ensures that files are correctly transmitted to the server, enabling reliable file handling. Furthermore, validating student enrollment before allowing access to course materials enforces strict authorization policies, protecting confidential resources and ensuring that only legitimate users can download educational content. Together, these measures strengthen application security and promote responsible data management.