# LABORATORY EXERCISE 4

# USER AUTHENTICATION (LOGIN/REGISTRATION)

**Learning Objectives**

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

* Create and run database migrations in CodeIgniter 4 to manage schema changes.
* Build secure user registration and login functionalities.
* Implement form validation using CodeIgniter's built-in Validation library.
* Manage user sessions to control access to protected pages.
* Structure views using Bootstrap for a responsive and styled interface.
* Utilize GitHub to track and manage project versions.

**Prerequisite student experiences and knowledge**

Before starting this exercise, students should have:

* Completed Laboratory Exercise 3 (Routing and MVC Structure).
* Understanding of PHP, HTML forms, and basic SQL.
* Familiarity with CodeIgniter's MVC architecture and routing.
* Experience using a local web server (XAMPP/WAMP/LAMP).
* Understanding of basic Git commands and GitHub usage.
* Ability to use a text editor/IDE such as Visual Studio Code.

**Background**

User authentication is a fundamental part of most web applications. It involves verifying a user's identity, typically through a login process, and then maintaining that user's state (session) across multiple pages. CodeIgniter provides powerful libraries for **Validation**, **Sessions**, and **Database** interaction, making it efficient to build secure authentication systems. **Migrations** allow for version control of your database schema, ensuring consistency across different development environments. This exercise will combine these components to create a complete login and registration system.

**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**

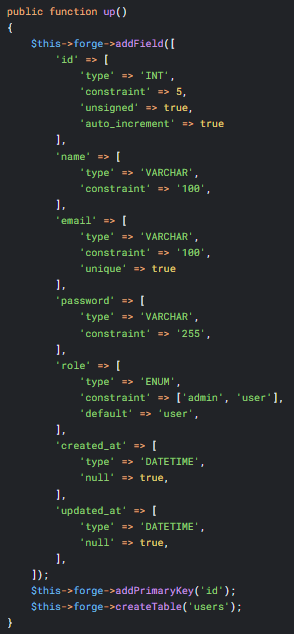
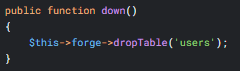
**Step 1: Project Setup**

1. Open your previously created CodeIgniter project **ITE311-LASTNAME**.
2. Ensure your local server and database are running.
3. Open a terminal/command prompt in your project root.

**Step 2: Create a Database Migration for Users Table**

1. Create a new migration file to define the **users** table schema.

Run: **php spark make:migration CreateUsersTable**

1. Open the newly created file in **app/Database/Migrations/**.
2. In the **up()** method, define the table with fields: ***id, name, email, password, role, created\_at, updated\_at***.
3. In the **down()** method, define how to drop the table.
4. Run the migration to create the table in your database: **php spark migrate**

**Step 3: Create an Auth Controller**

1. Navigate to **app/Controllers/** and create a file named **Auth.php**.
2. Add the following methods:

* **register()** Displays the registration form and processes form submission.
* **login()** Displays the login form and processes form submission.
* **logout()** Destroys the user's session and redirects them.
* **dashboard()** A protected page that only logged-in users can see.

**Step 4: Implement Registration Functionality**

1. In the **register()** method, check if the form was submitted (POST request).
2. Set validation rules for the **name, email, password,** and **password\_confirm** fields.
3. If validation passes, hash the password using the **password\_hash()** function.
4. Save the user data (**name, email, hashed\_password, role**) to the users table.
5. On success, set a flash message and redirect to the login page.
6. Load a view (**app/Views/auth/register.php**) that contains a Bootstrap-styled registration form.

**Step 5: Implement Login Functionality**

1. In the **login()** method, check for a POST request.
2. Set validation rules for **email** and **password**.
3. If validation passes, check the database for a user using the provided email.
4. Verify the submitted password against the stored hash using **password\_verify()**.
5. If credentials are correct, create a user session (store: **userID, name, email, role**).
6. Set a welcome flash message and redirect to the dashboard.
7. Load a view (**app/Views/auth/login.php**) that contains a Bootstrap-styled login form.

**Step 6: Implement Session Management and Logout**

1. In the **logout()** method, destroy the current session using **session()->destroy()**.
2. Redirect the user to the homepage or login page.
3. In the **dashboard()** method, check if a user is logged in at the start of the method. If not, redirect them to the login page.

**Step 7: Create Views and Configure Routes**

1. Create a directory auth inside app/Views/.
2. Create the views **register.php** and **login.php** inside this directory. Use Bootstrap classes for styling.
3. Update **app/Config/Routes.php** to include routes for your new authentication pages:

$routes->get('/register', 'Auth::register');

$routes->post('/register', 'Auth::register');

$routes->get('/login', 'Auth::login');

$routes->post('/login', 'Auth::login');

$routes->get('/logout', 'Auth::logout');

$routes->get('/dashboard', 'Auth::dashboard');

**Step 8: Test the Application**

1. Run the application and test the complete flow:
   * Register a new user account.
   * Log in with the new credentials.
   * Access the protected dashboard.
   * Log out and try to reaccess the dashboard.

**Step 9: Push to GitHub**

1. Push your changes to your GitHub repository every time you have changes in your syntax.

Output / Results

* Screenshot of the successful migration run in the terminal.
* Screenshots of the registration form, login form, and user dashboard.
* Screenshot of the users table in your database (phpMyAdmin or equivalent) showing the hashed password.
* A screenshot of the GitHub repository with the latest commit.

**QUESTIONS:**

1. Why is it critical to hash passwords before storing them in a database? Which PHP function did you use for this?
2. Explain the purpose of the Session library in CodeIgniter in the context of user authentication.
3. What is the advantage of migrations (as in Step 2) over manually creating the database table?

**Output / Results**

**Conclusion**