# LABORATORY EXERCISE 3

# Content Posting and Interaction

**Learning Objectives**

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

* Implement content posting and interaction functionalities (creating posts, commenting, and liking) in a social media platform using Laravel, AngularJS, and MySQL.
* Develop interactive features using AngularJS for dynamic frontend updates and Laravel for handling backend logic.
* Manage database operations with MySQL and ensure secure interaction management across the platform.

**Prerequisite student experiences and knowledge**

Students should have completed Laboratory Exercises 1 and 2, be familiar with MVC architecture, and have experience with Laravel, AngularJS, and MySQL. Students should also have experience with user authentication, profile management, and basic CRUD operations.

**Background**

In previous exercises, you implemented user authentication and profile management. This laboratory exercise focuses on adding core social media functionality: content posting and interaction. These features allow users to create, view, and interact with posts through likes and comments. In this exercise, you will work with **Laravel** for server-side logic, **AngularJS** for a dynamic front-end, and **MySQL** for persistent storage.

**Materials/Resources**

* **Web Browsers:** Google Chrome, Mozilla Firefox, or any modern web browser.
* **Code Editor:** Visual Studio Code, Sublime Text, or any preferred code editor.
* **Command Line Interface:** Git Bash, Terminal, or Command Prompt.
* **Development Tools:**
  + Composer (for PHP dependencies)
  + npm (Node Package Manager)
  + Git (version control)
* **Frameworks:**
  + Laravel
  + AngularJS
  + MySQL(Database)

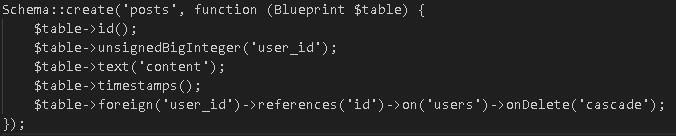
**Laboratory Activity**

**Step 1: Set up the Laravel Backend for Content Posting**

1. Create a Post Model in Laravel
   * Use Laravel's Eloquent ORM to create a Post model:

****

* + In the migration file, define the necessary fields for a post (e.g., user\_id, content, created\_at, updated\_at):

****

* + Run the migration:

****

1. Develop a Controller to Handle Posts
   * Create a PostController

****

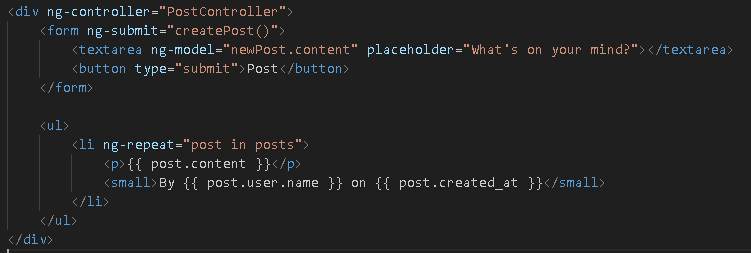
* + Implement methods for creating, retrieving, and deleting posts:

****

**Step 2: Set up the Frontend with AngularJS**

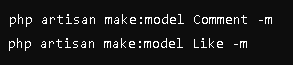
1. Create AngularJS Components for Post Creation and Display
   * Set up an AngularJS component to handle the post creation form

****

1. Create the HTML Structure for the Post Form and List
   * ****Add an AngularJS-based form and a list to display posts

**Step 3: Add Commenting and Liking Features**

1. Laravel Backend: Add Comment and Like Models
   * Create Comment and Like models with migrations

****

* + Define their relationships in the migrations and models

****

1. Add Like and Comment Functionality in AngularJS
   * ****Add methods to handle commenting and liking posts
2. ****Update the HTML Structure for Likes and Comments

**Step 4: Testing and Securing the Application**

1. **Test the Content Posting and Interaction Features**
   * Ensure users can create posts, like, and comment.
   * Test security by preventing unauthenticated users from interacting with posts.
2. **Secure the Application**
   * Use Laravel middleware to ensure only authenticated users can create, like, or comment on posts.
   * Implement Cross-Site Request Forgery (CSRF) protection in form submissions.

**Step 5: Deploy the Application**

1. **Deploy the Project**
   * Push the project to GitHub.
   * Deploy to a server or hosting platform, ensuring the database is properly configured with MySQL.Ensure that user roles are correctly enforced and that password reset flows are functioning securely.

**QUESTIONS:**

1. How do Laravel and AngularJS handle real-time data display differently when implementing comments and likes?

- **Laravel:** Typically relies on server-side rendering or AJAX requests to update data. For real-time interactions like comments or likes, Laravel would use polling or (via packages like Laravel Echo) to push updates to the client and make it working

- **AngularJS:** Uses client-side JavaScript to dynamically update the view without needing to reload the page.

1. What challenges did you encounter in securing user interactions (e.g., preventing unauthorized likes or comments), and how were they addressed?

**-Challenges:** Preventing me to unauthorized likes or comments can be tricky, as users might try to bypass restrictions or perform actions they aren’t authorized for.

**- Solutions:** Implementing middleware for authentication and authorization checks in Laravel ensures that only logged-in users can like or comment. And validating user actions on both the client side (AngularJS) and server side (Laravel) helps secure these interactions.

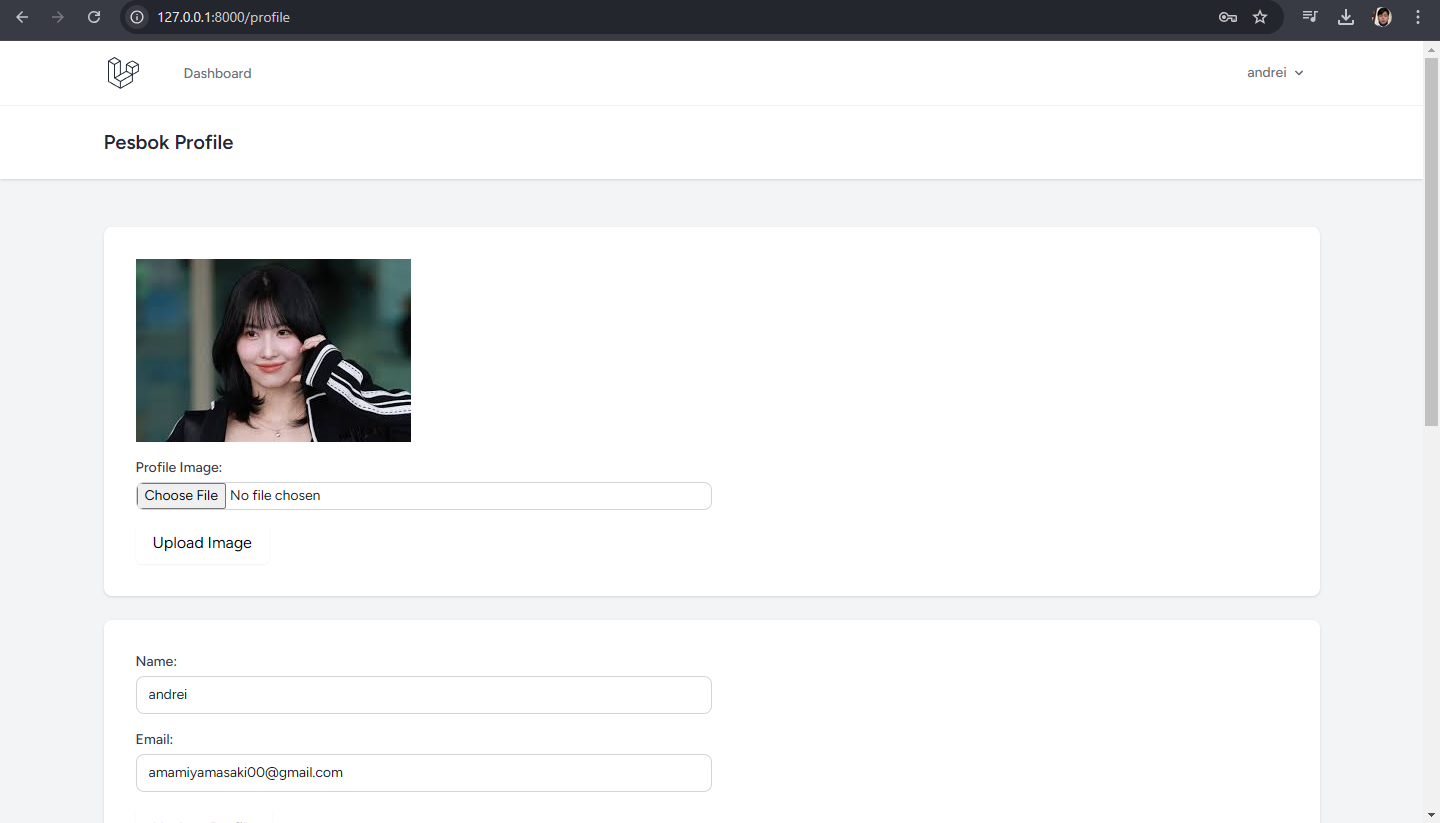
1. Analyze the role of MySQL in efficiently storing and retrieving user interactions on the platform.

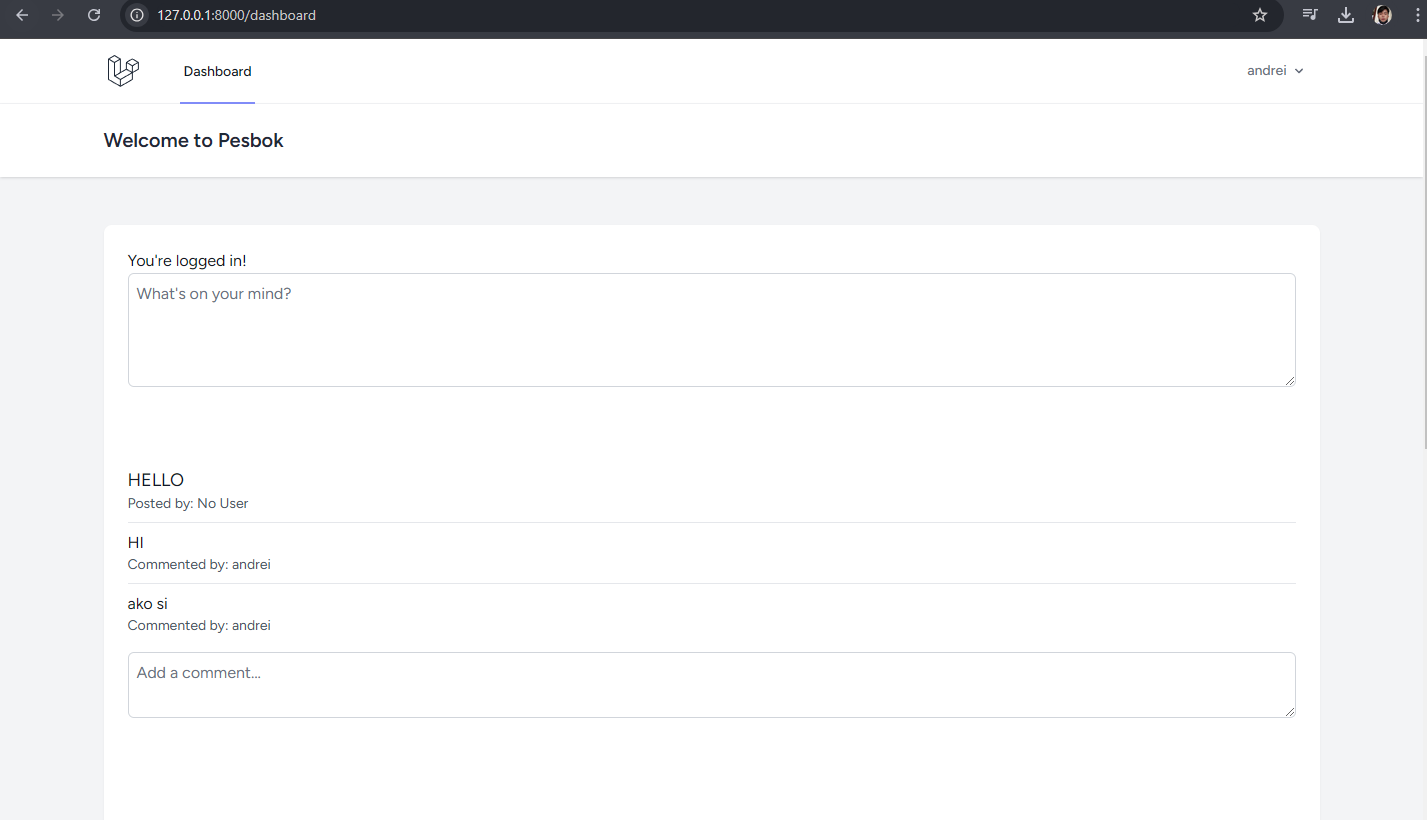
**-Efficiency:** MySQL helps me efficiently to store and retrieve user interactions through structured tables and indexes. Comments and likes are stored in separate tables with foreign keys linking them to posts and users.

**Output / Results**

*Provide screenshots of your user authentication and profile management implementations.*

*Include URLs to your deployed projects and GitHub repositories.*

**

**

**Conclusion**

**Laravel** manages real-time data through server-side rendering and AJAX, with WebSockets or polling for live updates. **AngularJS** excels in real-time data display by leveraging two-way data binding and direct manipulation of the DOM, making the user interface more responsive.

https://github.com/Dreiwantscode/Social\_media\_name\_app3/tree/main