**Landing, Login, and Enrollment Pages Development**

Xenia Reid

The University of Arizona Global Campus

CST499: Capstone for Computer Software Technology

Professor Joseph Rangitsch

July 22, 2025

**Landing, Login, and Enrollment Pages Development**

Developing secure and efficient landing, login, and enrollment pages is critical in any web application, especially when handling sensitive user information. In this project, PHP and MySQL were used to implement a dynamic system that securely registers and stores user data. The project focused on using MySQLi functions to work with the database, creating a reusable connection class, and building a secure registration process with proper form handling and validation. This paper will explain how MySQLi was implemented in the project, how user registration data is processed and stored, and the security measures taken to protect sensitive user information.

**MySQL Database Functions Used**

When developing a web application with PHP and MySQL, effective and secure interaction with the database is essential. In this project, I used the MySQLi extension in PHP to manage database operations such as connecting to the database, executing queries, and retrieving results. To improve code maintainability and reusability, I developed a custom database connection class that encapsulates the connection logic.

The PHP MySQLi extension provides an improved interface to interact with MySQL databases. The key functions and methods used in the project include:

* **new mysqli(host, username, password, dbname)**  
  This constructor creates a new connection object to the MySQL server with the specified host, user credentials, and database name.
* **$conn->connect\_error**This property stores any error message that occurs during connection. Checking this allows for error handling.
* **$conn->prepare($sql)**  
  Prepares an SQL statement for execution. Prepared statements increase security by separating query structure from data, helping to prevent SQL injection.
* **$stmt->bind\_param("ssss", $param1, $param2, $param3, $param4)**  
  Binds variables to the prepared statement as parameters. The string argument defines the data types for the variables (s for string, i for integer, etc.).
* **$stmt->execute()**  
  Executes the prepared statement.
* **$stmt->get\_result()**  
  Retrieves the result set from a SELECT query executed by a prepared statement.
* **$result->num\_rows**  
  Returns the number of rows in the result set, useful to verify if any records matched a query.
* **$result->fetch\_assoc()**  
  Fetches a row from the result set as an associative array for easy access by column names.
* **$stmt->close() and $conn->close()**  
  Properly close the prepared statement and the database connection to free resources.

Using prepared statements is important for security because it prevents SQL injection vulnerabilities by ensuring that user inputs are treated as data, not executable code. This separation safeguards the database from malicious queries (Security Journey, n.d.).

To avoid repeating connection code across multiple PHP scripts and to improve code organization, I developed a custom database connection class. This encapsulation promotes reuse and simplifies maintenance. Encapsulation ensures that each class hides its internal details and only exposes necessary information (Dooley, 2017).

The Database class establishes a connection to the database when created and offers a method to access the connection object.

Instead of creating a new MySQLi connection in every script, the Database class is initialized once, and the connection is retrieved.

The benefits of using the Database class include centralized configuration, as database credentials and connection parameters are defined in one place. This promotes code reusability, allowing the connection logic to be reused across different parts of the application. It also simplifies maintenance because updates to connection handling need to be made only once (Jaiswal, 2024).

Utilizing MySQLi functions with prepared statements ensures secure and effective database interactions in PHP applications. Creating a custom database connection class improves code clarity, reduces redundancy, and prepares the project for easier scaling and future enhancements.

**Steps to Create the Registration Page and Save User Information in the Database**

The first step was to create a user-friendly registration form using HTML. This form collects essential user details such as:

* First Name
* Last Name
* Email Address
* Password

The form uses the POST method to securely send data to a server-side PHP script for processing.

Example form fields include:

Register.php:



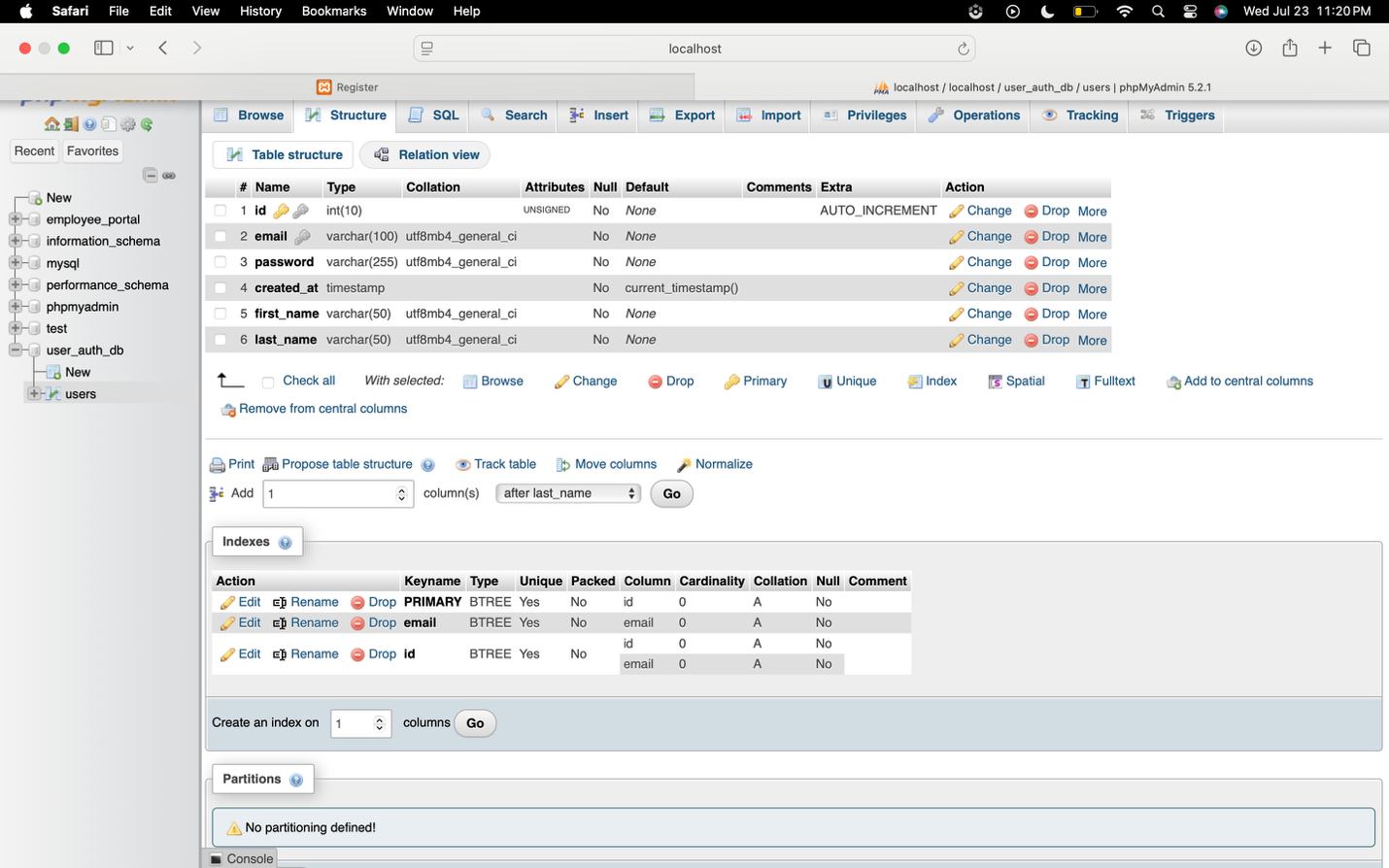
A screenshot of a computer

AI-generated content may be incorrect.

Before saving any data, the MySQL database table must be created to store user information. The users table contains columns that correspond to the form fields:

* id: Unique identifier (auto-incremented)
* first\_name: User’s first name
* last\_name: User’s last name
* email: User’s email address (unique)
* password: Hashed password for security
* created\_at: Timestamp of registration

This table ensures data integrity and provides a structured way to save and retrieve user data.



The registration form submits data to register\_process.php, which handles the following tasks:

1. Connecting to the Database - A connection to the MySQL database is established using MySQLi or a custom database class.

A screen shot of a computer program

AI-generated content may be incorrect.

1. Retrieving and Validating Form Data- User inputs are captured from the $\_POST superglobal. Basic validation, such as ensuring required fields are not empty, can be performed here.
2. Securing the password- Passwords are hashed using PHP’s password\_hash() function before storage. This secures user credentials by storing an irreversible hash rather than plain text.

A computer screen shot of text

AI-generated content may be incorrect.

1. Preparing and executing the SQL Query - A prepared statement is created to insert user data safely into the database, protecting against SQL injection attacks.

A black background with text

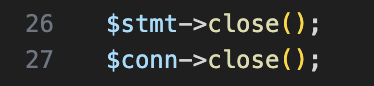
AI-generated content may be incorrect.

1. Handling Success or Errors - If the insertion is successful, a confirmation message is displayed, or the user may be redirected to the login page. If an error occurs, an appropriate error message is shown.

A computer screen with text

AI-generated content may be incorrect.

After completing the operation, the prepared statement and database connection are closed to free up resources.



By combining an HTML form for input, a secure MySQL table for storage, and PHP for processing and validation, the registration page effectively collects and saves new user information. The use of password hashing and prepared statements ensures that the application follows good security practices to protect user data.

**Developed Pages:**

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

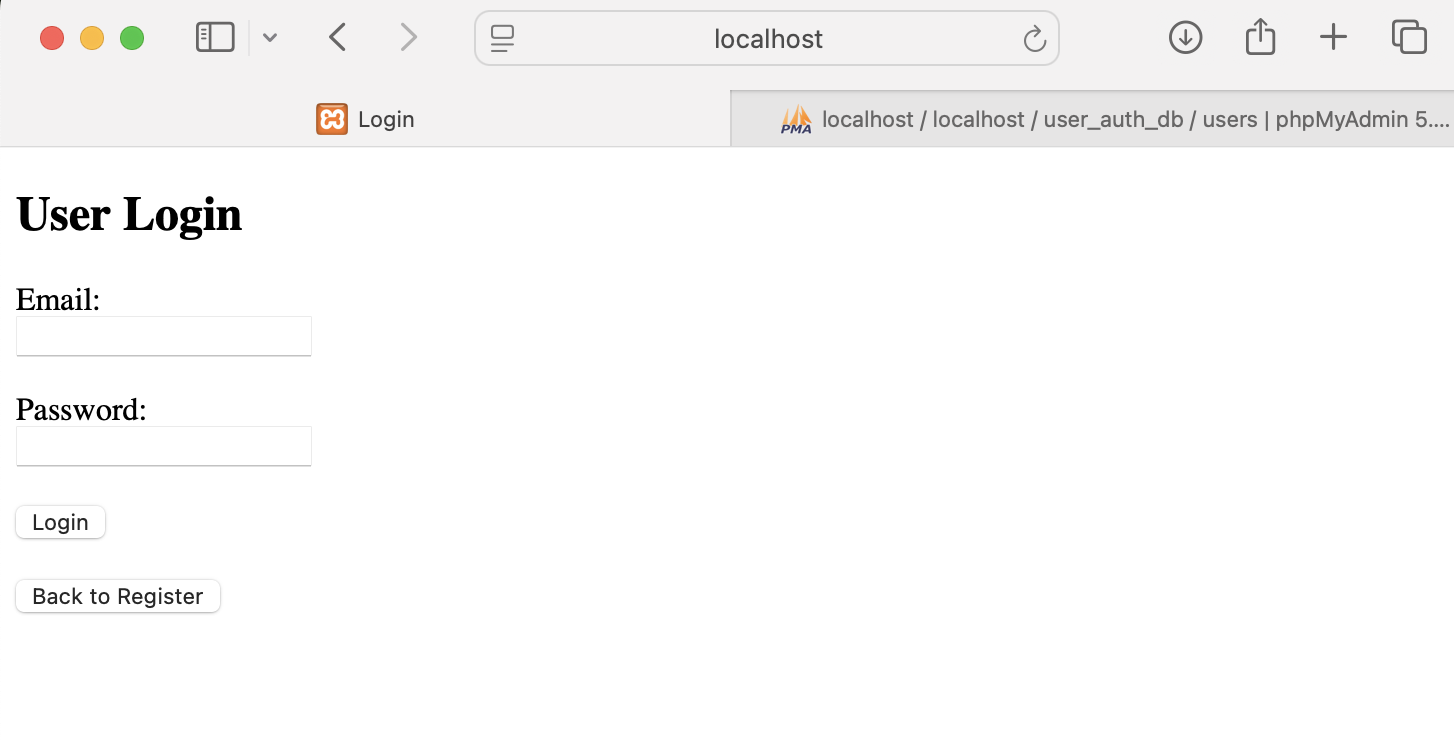
AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.



A screenshot of a computer

AI-generated content may be incorrect.

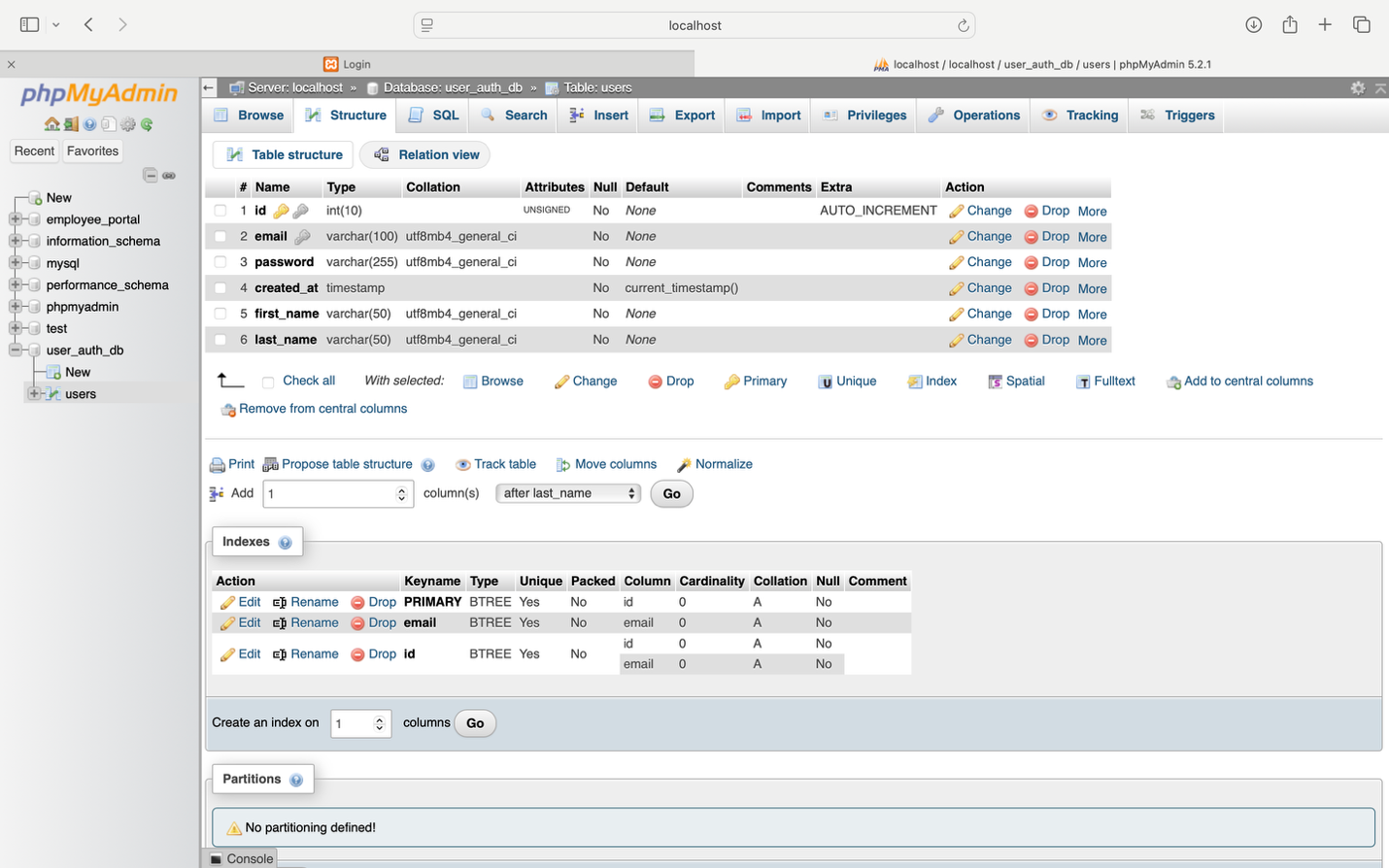
A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Database & Tables**



**Source code**

**Index.php**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**register.php**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**register\_process.php**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**login.phpA screen shot of a computer program

AI-generated content may be incorrect.**

**login\_process.php**

**A screen shot of a computer

AI-generated content may be incorrect.**

**References**

Dooley, J. F. (2017). *Software development, design, and coding: With patterns, debugging, unit testing, and refactoring*. (2nd ed.). Springer Nature.

Jaiswal, S. (2024, June 8). *Unlocking Database Connectivity: Advantages of Using Data Sources in Java Applications*. Medium. https://satyendrakjaiswal.medium.com/unlocking-database-connectivity-advantages-of-using-data-sources-in-java-applications-14b52ef9e050.

Security Journey. (n.d.). *How to prevent SQL Injection Vulnerabilities: How Prepared Statements Work*. https://www.securityjourney.com/post/how-to-prevent-sql-injection-vulnerabilities-how-prepared-statements-work#:~:text=A%20prepared%20statement%20is%20a,safely%2C%20preventing%20SQL%20Injection%20vulnerabilities.