

**VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY**

**INTERNATIONAL UNIVERSITY**

**SCHOOL OF COMPUTING, ...**



# **FINAL PROJECT REPORT**

## **WEB APP DEVELOPMENT**

Ho Chi Minh City, Vietnam

December, 2025

# **Table of Contents**

1	Introduction .....	4
1.1	Project Background .....	4
1.2	Programming Language .....	4
1.3	Framework, Database, and Tools .....	4
2	Requirement Analysis .....	5
2.1	Use Case Diagram.....	5
2.2	Functional Requirement & Non-functional.....	6
2.2.1	Functional Requirements .....	6
2.2.2	Non-Functional Requirements .....	8
3.	Design and Implementation .....	10
3.1	Design.....	10
3.1.1	Model View Controller Model (MVC Model) .....	10
3.1.2	Relational Schema.....	11
3.1.3	Sequence Diagram .....	14
3.2	Implementation.....	18
4	Challenges .....	23
4.1	Database Synchronization .....	23
4.2	Insufficient Knowledge in Tools/Technologies .....	24
5	Conclusion.....	24

## **1 Introduction**

### **1.1 Project Background**

The Bookly is a dynamic platform designed to connect readers with their favorite books. Users can explore various categories, view book details, and make purchases directly on the website. Features include account management, shopping cart functionality, and an admin dashboard for content and order management.

### **1.2 Programming Language**

For this project, we selected PHP (Hypertext Preprocessor) as the core server-side scripting language.

Why PHP? PHP is widely used for web development due to its simplicity, efficiency, and strong support for database integration. It allows us to build dynamic web pages that interact seamlessly with the database.

Version: The project is compatible with PHP 8.1 (as indicated in the SQL dump header).

### **1.3 Framework, Database, and Tools**

Instead of using a heavy framework like Spring Boot (as initially planned in previous drafts), we opted for Native PHP to demonstrate a solid understanding of fundamental web logic without relying on third-party abstractions.

Database: MySQL (via phpMyAdmin) is used for storing relational data such as users, products, and orders.

Frontend Technologies: HTML5, CSS3 (Custom styling in style.css), and JavaScript (script.js for frontend interactivity).

XAMPP: For local server environment (Apache + MySQL).

Visual Studio Code: For code editing.

Git/GitHub: For version control and collaboration.

## 2 Requirement Analysis

### 2.1 Use Case Diagram

This Use Case Diagram illustrates the functional requirements of a **BookStore** system, identifying the interactions between different types of users (Actors) and the system's core features (Use Cases).

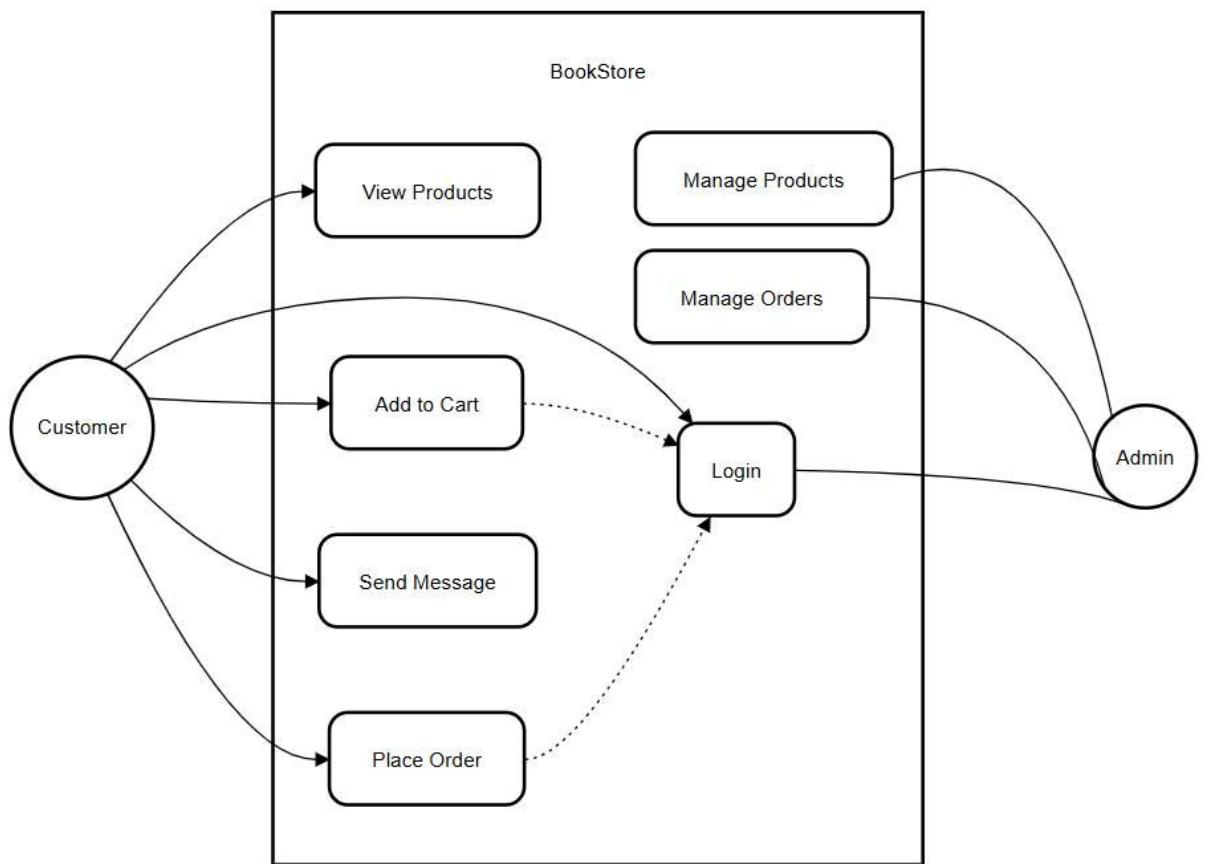


Figure 1: Usecase diagram of the system

#### Actors

The diagram identifies two primary **Actors** who interact with the system:

**Customer:** A front-end user who browses the store and makes purchases.

**Admin:** A back-end user responsible for managing the store's data and operations.

#### System Boundary

The large rectangle labeled "**BookStore**" represents the system boundary. It defines the scope of the application; everything inside the box is a functional requirement of the software, while the actors remain outside as external entities.

## **Use Case Descriptions**

The ovals (or rounded rectangles) represent the specific actions or services the system provides:

Customer Actions:

View Products: Allows the customer to browse the inventory.

Send Message: Provides a way for the customer to contact support or the store owner.

Add to Cart: Allows the customer to select items for potential purchase.

Place Order: The final step in the purchasing process.

Login: A security feature to identify the user.

Admin Actions:

Manage Products: Adding, editing, or deleting books from the inventory.

Manage Orders: Reviewing, updating, or fulfilling customer orders.

Login: Accessing the administrative dashboard securely.

## **Relationships and Logic**

The lines and arrows indicate how the actors interact with the functions:

Associations: The solid lines show which actor can perform which action. For example, both the Customer and the Admin have a direct relationship with the Login use case.

Dependencies (Dashed Arrows): The dashed lines pointing from Add to Cart and Place Order toward Login represent an Include or Pre-requisite relationship.

*Interpretation:* In this system, a customer can "View Products" without logging in, but they must log in to "Add to Cart" or "Place Order." This ensures that transactions are tied to a specific user account.

## **2.2 Functional Requirement & Non-functional**

### **2.2.1 Functional Requirements**

#### **1) Account Management**

- Supports user register / login / logout
- Uses session-based authentication to store user info (user ID, name, email)
- Admin pages are protected by an admin authentication gate (admin session)

- Provides an account dropdown for admins (view account + logout)
- Provides a header user box toggle for users (view session info + logout)

## **2) Catalog Management**

Customers can view products:

- Homepage shows latest products (limited set)
- Shop page shows full product list
- Admin can manage products with full CRUD:
- Add / edit / delete products
- Upload product images
- Product images are served from an upload directory based on stored image paths

## **3) Shopping Cart and Order Management**

Users can add products to cart from product listing pages

Cart data is stored in a cart database table

Cart page supports view cart items, update quantity, remove items, calculate subtotal and total

Checkout process supports: Calculate grand total, check duplicate orders, create order records in orders table, clear cart after successful ordering

Users can view their orders

Admin order management supports: View all orders, update order status, delete orders

## **4) Notification System**

Uses flash messages for user/admin actions

Messages are displayed in the interface after key operations (e.g., add to cart, update, checkout)

Supports: Manuals dismiss (close icon), Auto-dismiss after a short time

## **5) Content Management**

Includes static/informational pages (e.g., About section)

Contact feature:

- Users submit contact messages
- Messages are stored in a message table
- Admin can view and delete messages
- System-wide footer provides consistent links/navigation

## **6) Search and Navigation**

Responsive navigation for users:

- Main navbar + user dropdown
- Mobile menu toggle
- Sticky header on scroll + close-on-scroll behavior

Admin navigation:

- Admin navbar + account box toggle
- Product search:
- Search products by keyword
- Display matching results on a search results page

### **2.2.2 Non-Functional Requirements**

#### **1. System Performance**

Uses simple, direct mysql\_query per request

No caching or pagination for shop and order pages, which may reduce performance with large datasets

Product images are served statically from the upload directory

No asset bundling, minification, or CDN usage except Font Awesome

JavaScript and CSS files are small and loaded separately

#### **2. User Experience**

Responsive toggles for navbar, user dropdown, and admin account box

- Sticky header behavior during scrolling
- Flash messages with auto-dismiss functionality
- Mobile menu support for smaller screens
- Forms rely mainly on basic HTML validation
- No advanced client-side validation for checkout or contact forms

### **3. Device & Browser Compatibility**

- Uses modern ES6 JavaScript features (e.g., querySelector, classList, arrow functions, optional chaining)
- Compatible with modern browsers
- No polyfills for older browsers
- Layout built with flexbox and simple CSS
- UI interactions rely heavily on .active class toggling

### **4. Future Growth**

- No pagination or optimized search queries
- Scaling product and order lists will require query limits, indexing, and pagination
- No front-end build pipeline in place
- Asset growth may require bundling and minification
- Search functionality is basic
- Advanced filtering or faceted search will require additional backend logic

### **5. Data Security**

- Session-based access control for users and admins
- Database queries interpolate variables, even when escaped
- No use of prepared statements, increasing security risk
- No CSRF protection on forms
- Password handling not evaluated in this scope

Product image uploads require file type and size validation

No rate limiting or CAPTCHA for login and contact forms

### 3. Design and Implementation

#### 3.1 Design

##### 3.1.1 Model View Controller Model (MVC Model)

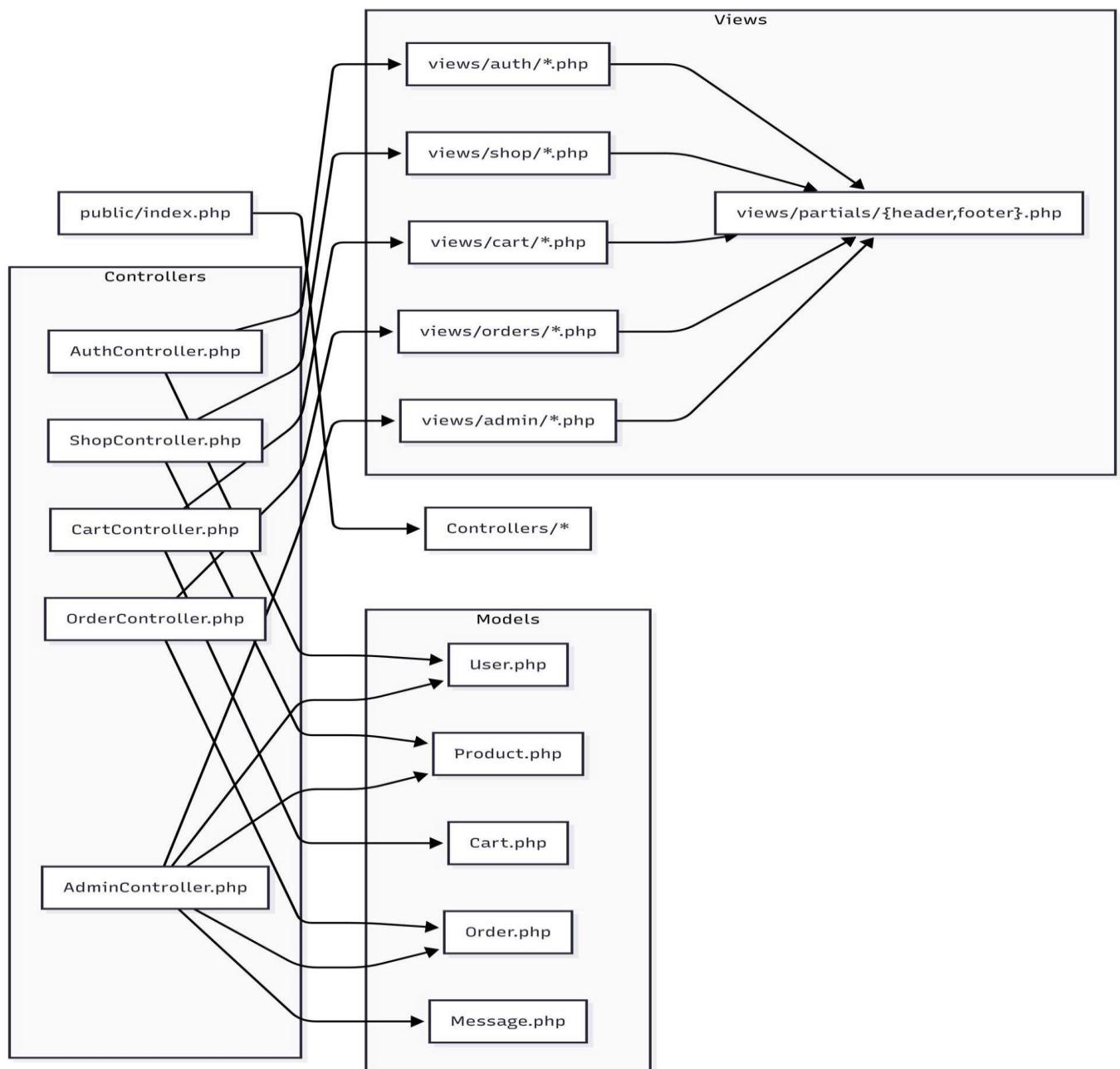


Figure 2: The MVC Diagram of the project

Although the project is built with Native PHP, it follows a logical separation of concerns.

Such as the MVC architecture:

- Model (Database & Logic): Represented by the MySQL database (shop\_db)

and the connection file config.php. It handles data structure and retrieval.

- View (User Interface): The HTML sections within files (e.g., home.php, shop.php) act as the View.

- Controller (Business Logic): The PHP blocks at the top of each file act as Controllers.

They handle form submissions (`$_POST`), validate input, communicate with the database and determine which View to render.

### **3.1.2 Relational Schema**

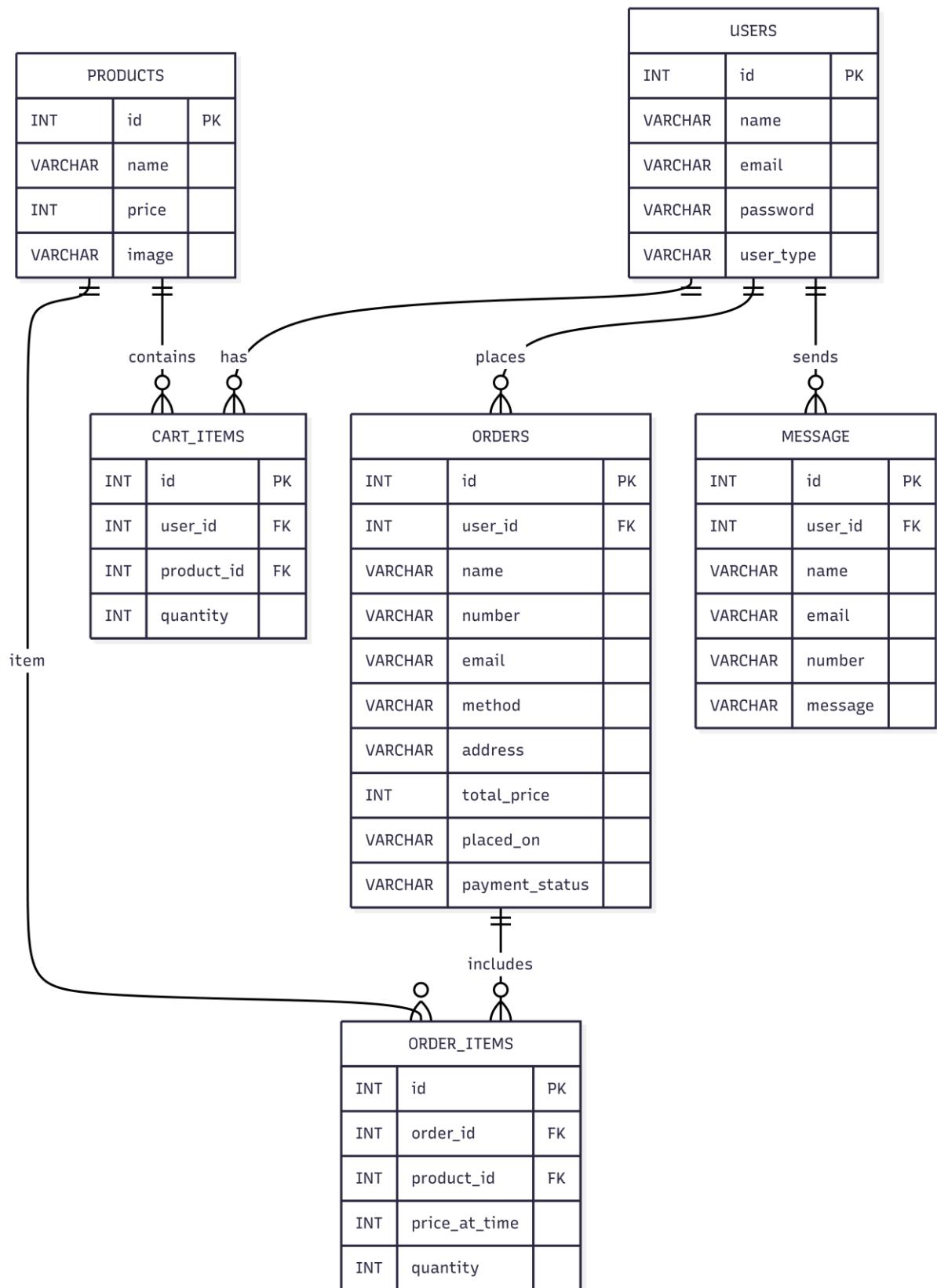


Figure 3: The relational schema of the project

## Core Tables and Data Entities

The schema consists of five primary tables, each serving a specific role in the e-commerce workflow:

- **Users:** The central hub of the database. It stores account details including names, emails, passwords, and a `user_type` (which likely distinguishes between the "Customer" and "Admin" roles seen in your use case diagram).
- **Products:** Contains the inventory data, storing the book name, price, and a reference to the image file.
- **Cart:** Acts as a bridge between users and products. It tracks which items a specific user has selected and the quantity of those items.
- **Orders:** Records finalized transactions. This table is data-heavy, capturing the customer's contact info, address, payment status, and a summary of the total\_products purchased.
- **Message:** Stores communications sent by users, including their contact details and the specific message content.

## Relationships and Cardinality

The dashed lines represent Foreign Key relationships, which maintain data integrity across the system:

### One-to-Many (1:N) Relationships

Most connections here follow a "one-to-many" logic, indicated by the straight line on one side and the "crow's foot" (three-pronged fork) on the other:

- **Users to Orders:** One user can place multiple orders over time, but each order belongs to only one user.
- **Users to Cart:** One user can have multiple distinct items in their cart.
- **Users to Message:** One user can send multiple messages/inquiries to the store.
- **Products to Cart:** One product can be added to the shopping carts of many different users.

## Data Integrity and Keys:

**Primary Keys (Yellow Key Icon):** Each table has an `id` field. This is a unique identifier that ensures no two rows in a table are identical.

**Foreign Keys (Red Diamond Icon):** Fields like `user_id` and `product_id` are used to "point" to the primary key of another table. This allows the system to know exactly which user placed which order without duplicating all the user's personal info in the `Orders` table.

## Technical Specifications

**Data Types:** The schema uses `INT` for whole numbers (like price or quantity) and `VARCHAR` for text (like names or addresses).

**Storage Limits:** For example, the address field allows up to 500 characters, while the total\_products field in the Orders table allows up to 1000 characters to accommodate a long list of purchased items.

### 3.1.3 Sequence Diagram

#### 1. Register account

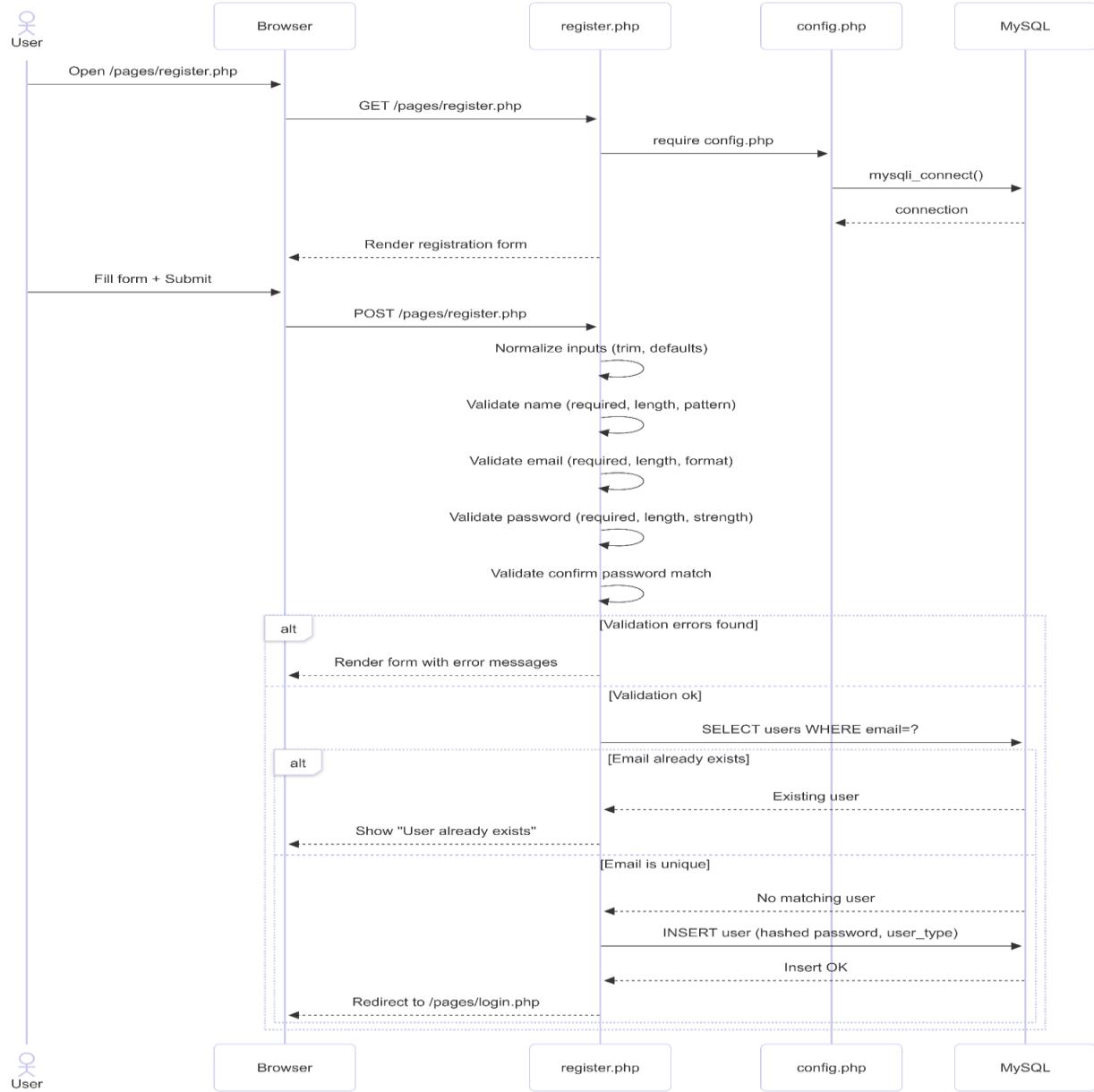


Figure 4: Sequence diagram for register account

The registration flow starts when a user submits the sign-up form. The server normalizes and validates the name, email, password, and confirmation fields. If

any validation fails, error messages are returned to the form. If validation passes, the system checks for an existing account by email. If the email is unique, it creates the user record and redirects to the login page; otherwise, it shows a “User already exists” message.

## 2. Login Account

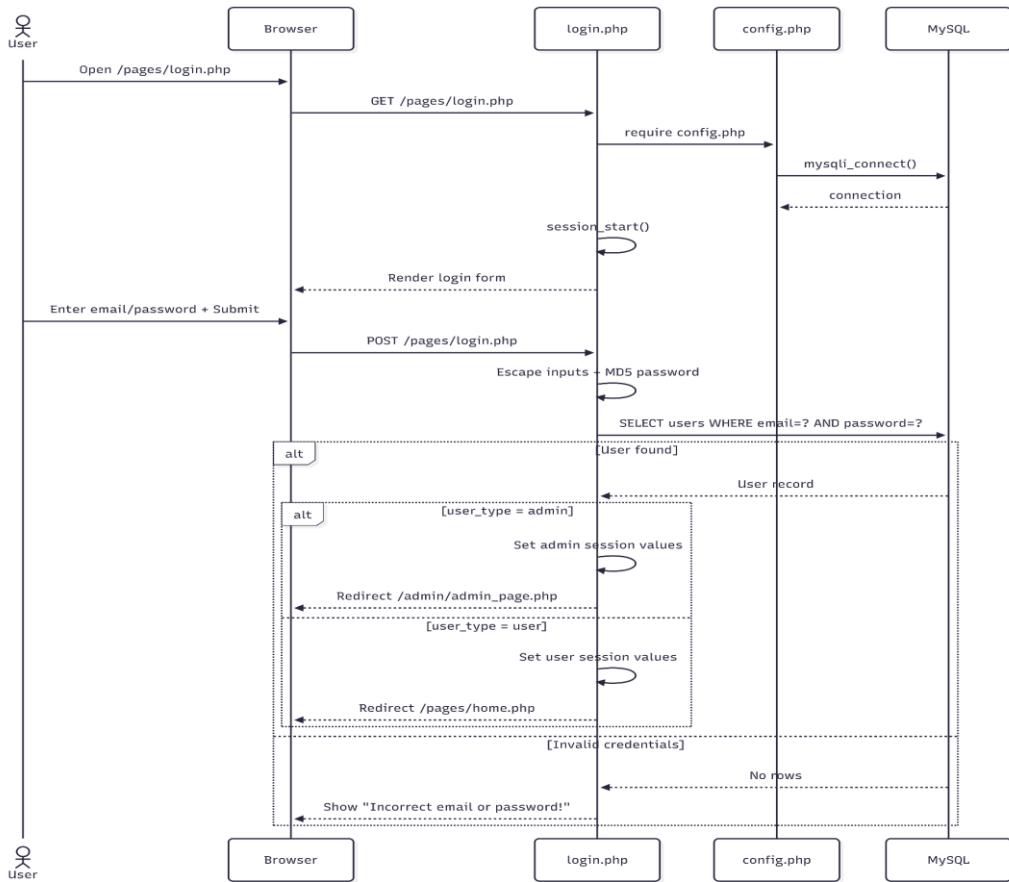


Figure 5: Sequence diagram for login account

The login flow begins when the user submits their email and password. The server hashes the password and queries the database for a matching user record. If a match is found, it starts a session, stores user data based on the role (admin or user), and redirects to the appropriate dashboard. If the credentials are invalid, the user is returned to the login page with an error message.

### 3. Order book

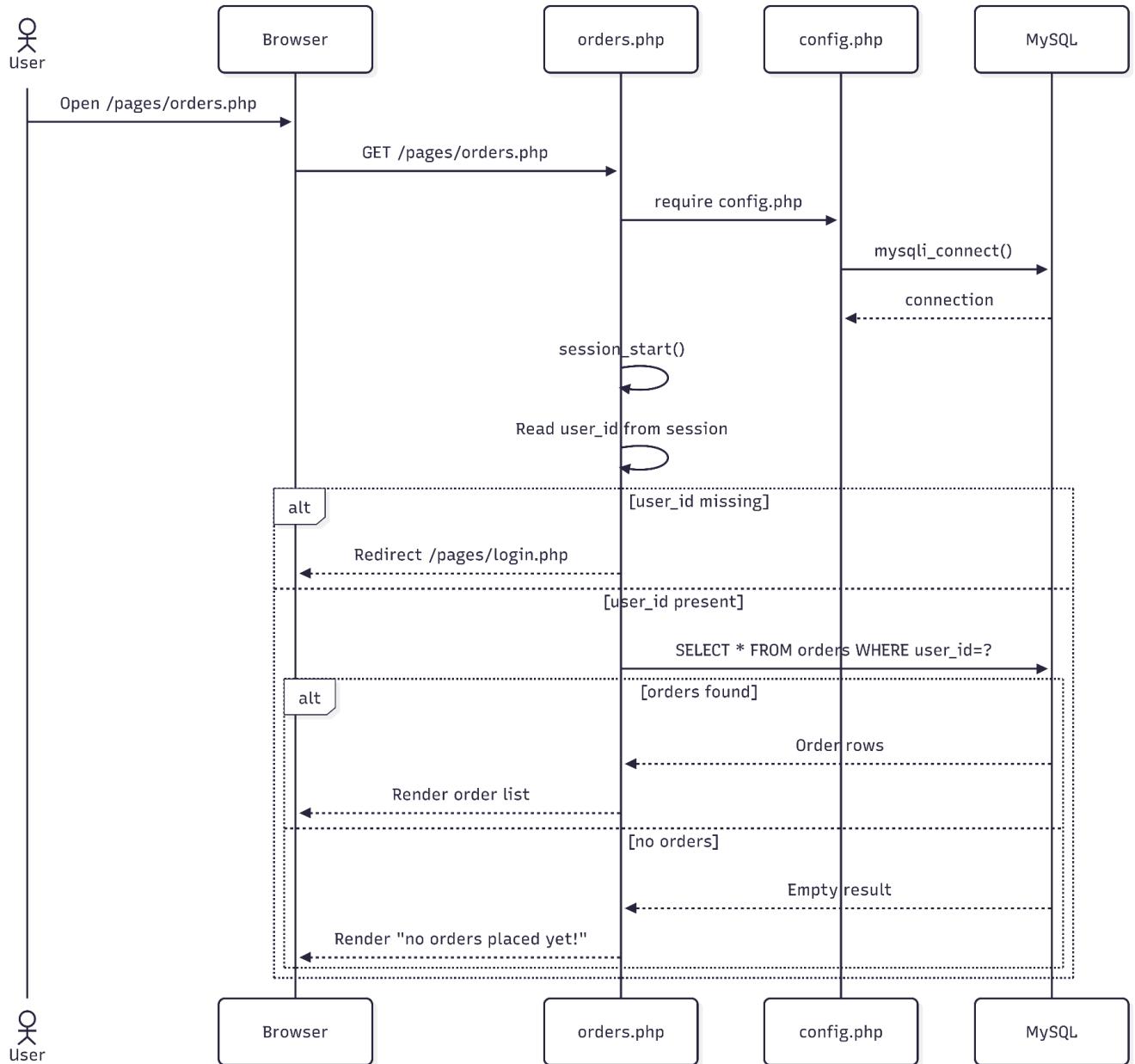


Figure 6: Sequence diagram for order book

The orders flow starts when a logged-in user opens the orders page. The server checks the session for a valid user ID; if missing, it redirects to the login page. If authenticated, it queries the database for the user's orders and renders either the order list or a “no orders placed yet” message.

## 4. Purchase

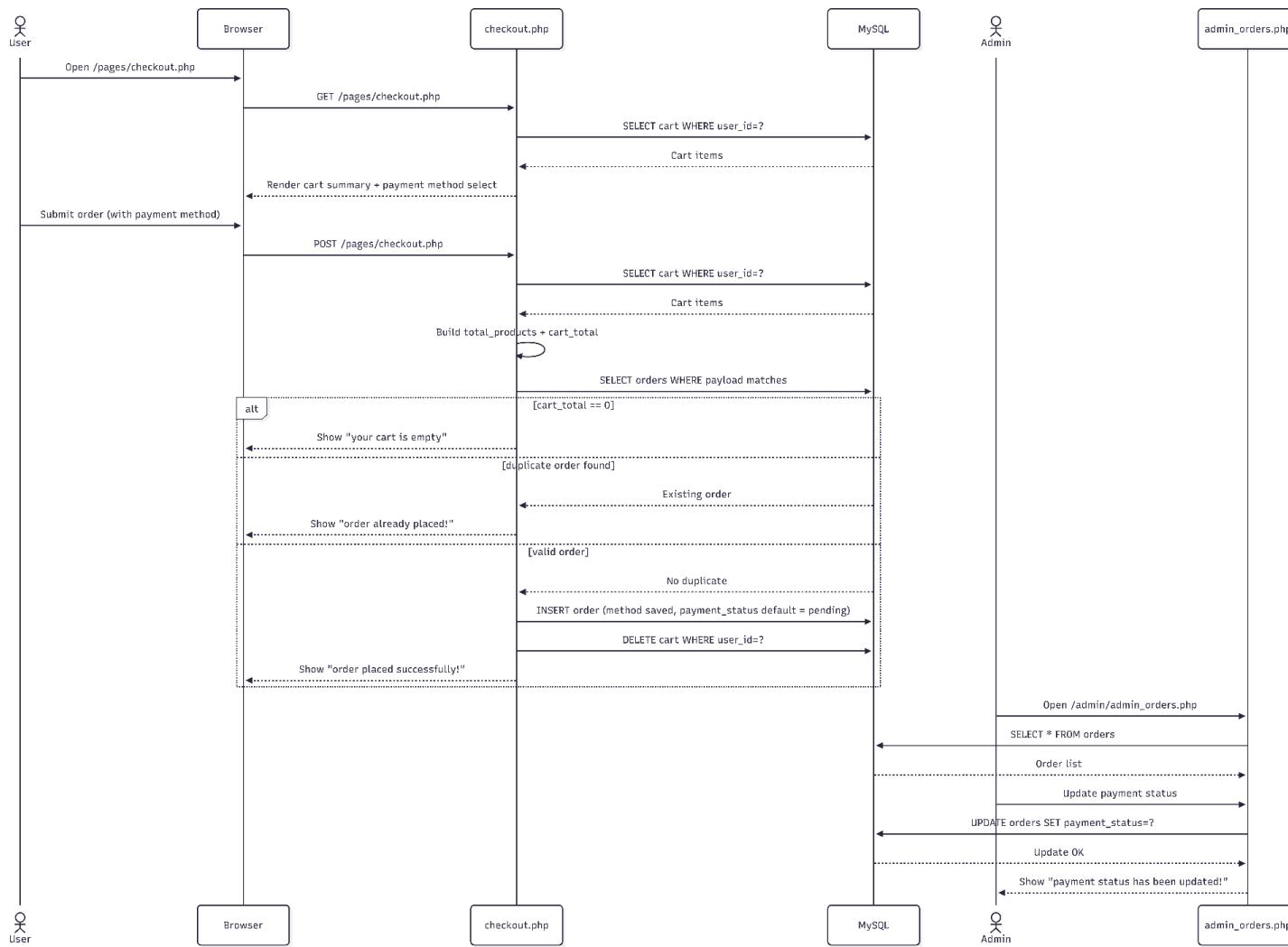


Figure 7: Sequence diagram for purchase

The payment flow is recorded during checkout. The user selects a payment method and submits the order. The server calculates cart totals, prevents duplicate submissions, and inserts a new order with the selected method and a default payment status of “pending,” then clears the cart. An admin can later update the payment status from the admin orders page.

## 5. Shipping

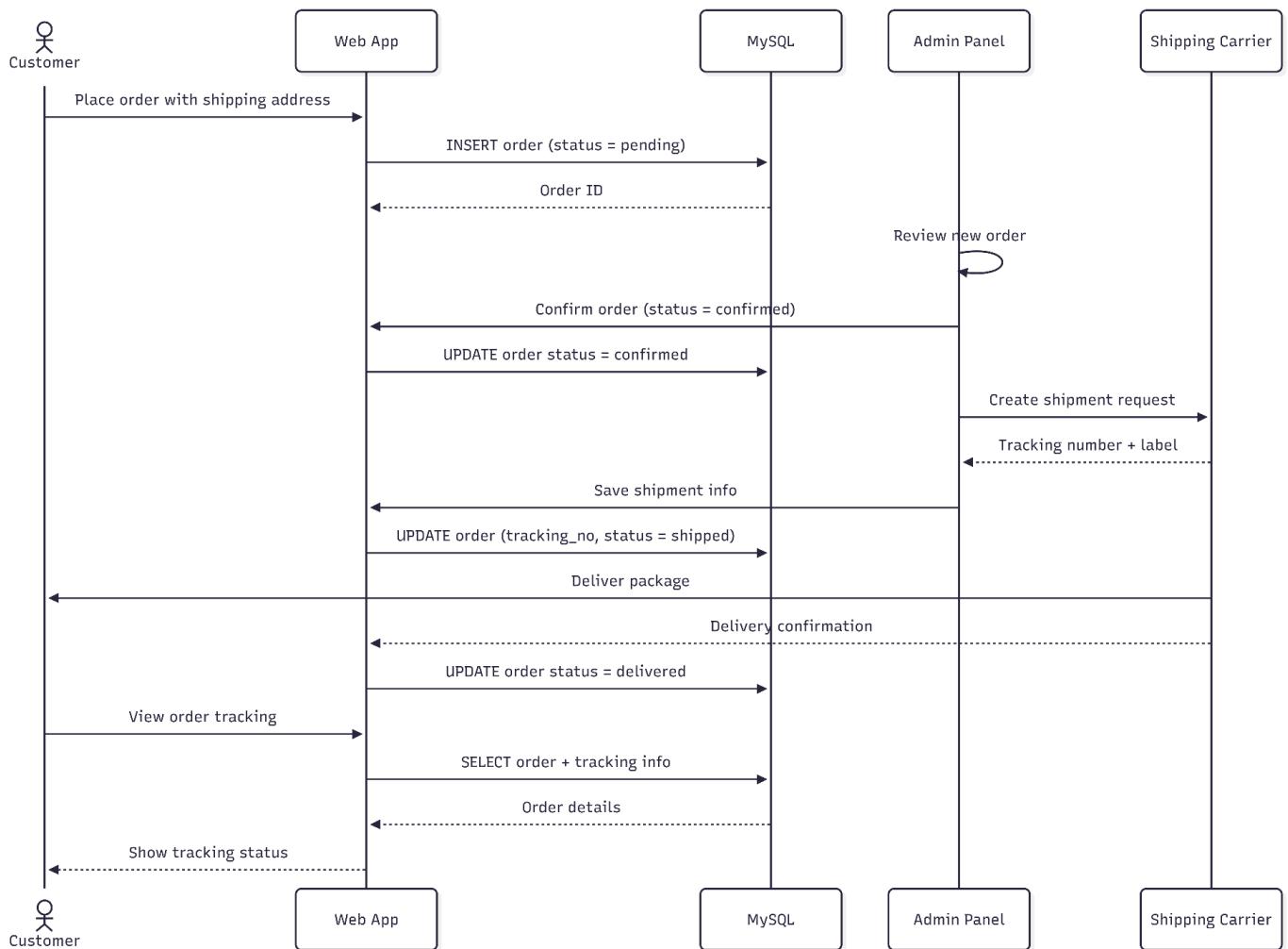


Figure 8: Sequence diagram for shipping

After an order is confirmed, the admin creates a shipment with a carrier and stores the tracking number. The order status moves from confirmed to shipped, and the customer can view tracking information. When the carrier confirms delivery, the system updates the order to delivered.

### 3.2 Implementation

#### Register account

Go to the homepage

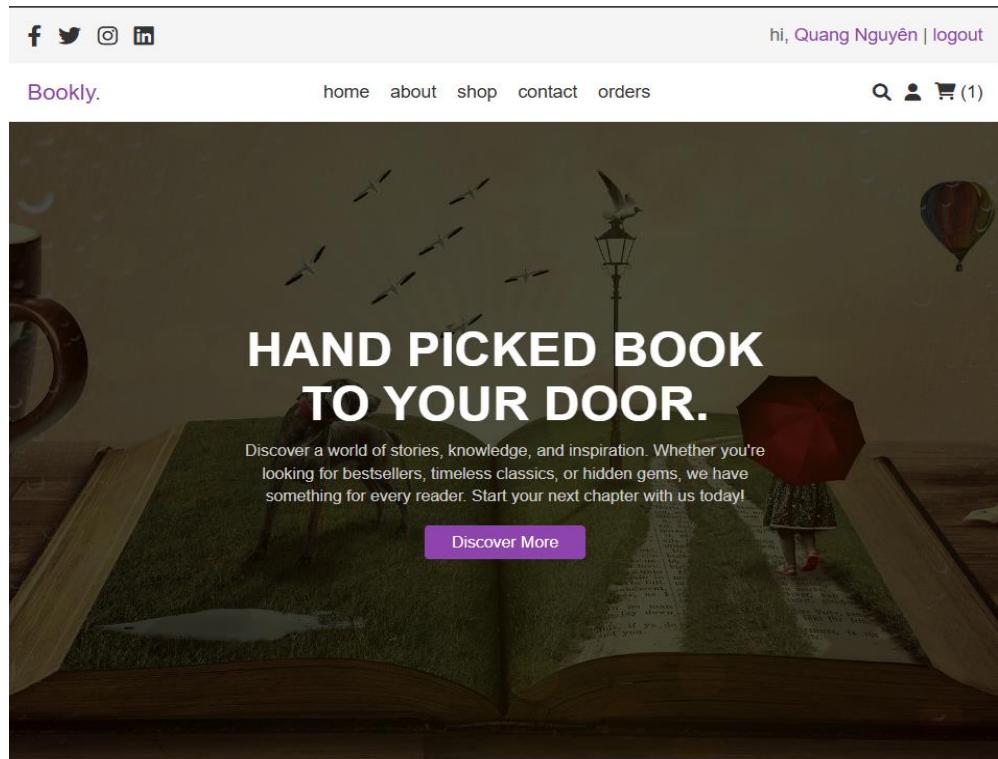


Figure 13: Homepage website

Press button Login to create login account.

A screenshot of the login page. It features a central form with a light gray background. At the top center is a bold, black "LOGIN NOW" button. Below it are two input fields: the top one is labeled "enter your email" and the bottom one is labeled "enter your password", both in small black text. At the bottom of the form is a purple "Login Now" button. Below the button, a link in purple text reads "don't have an account? register now".

Figure 14: login page

Press creates new account if you didn't have ones.

# REGISTER NOW

enter your name

enter your email

enter your password

confirm your password

**Register Now**

already have an account? [login now](#)

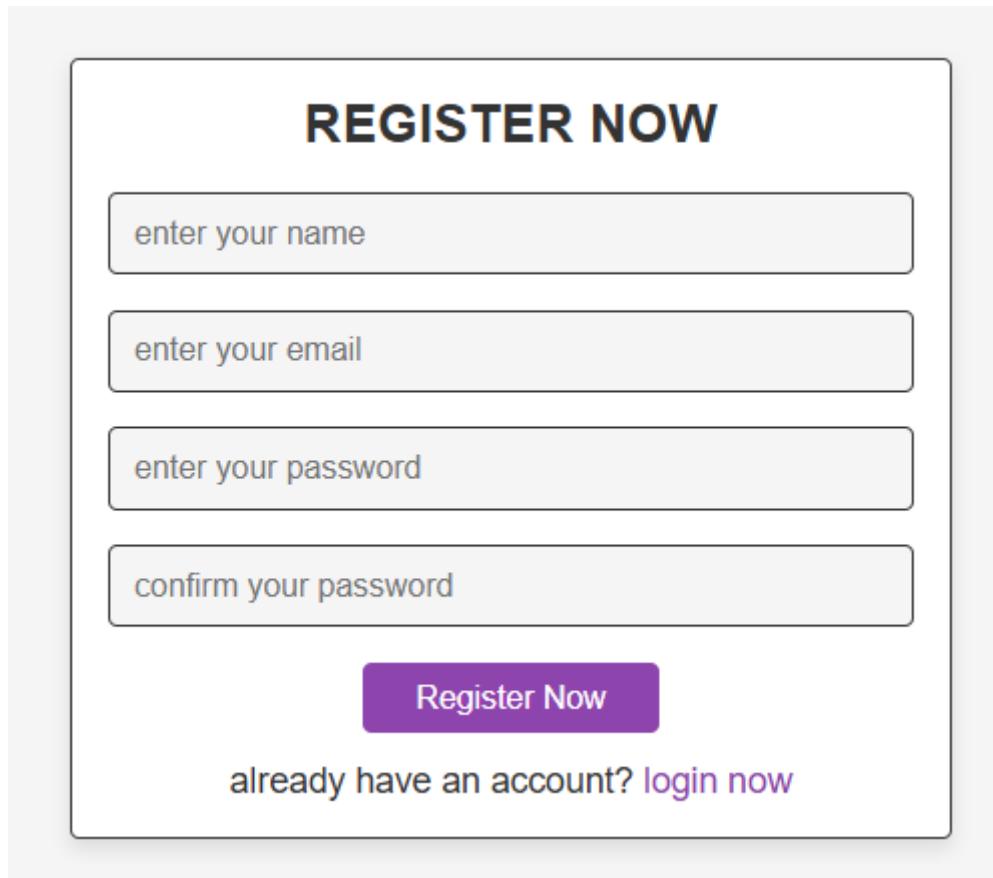


Figure 15: Create account

# LOGIN NOW

lequangnguyen@gmail.com

.....

**Login Now**

Don't have an account? [Register now](#)

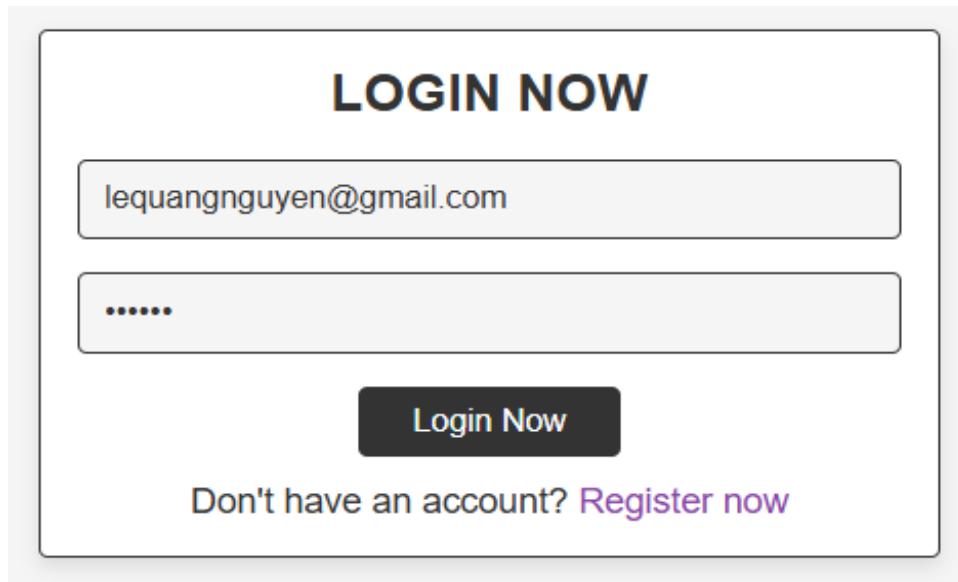


Figure 16: Login (check the box remember me to quick login for the later time)

After login website will bring us to the home page like this. There are many books type so you can click to find your favorite type of books

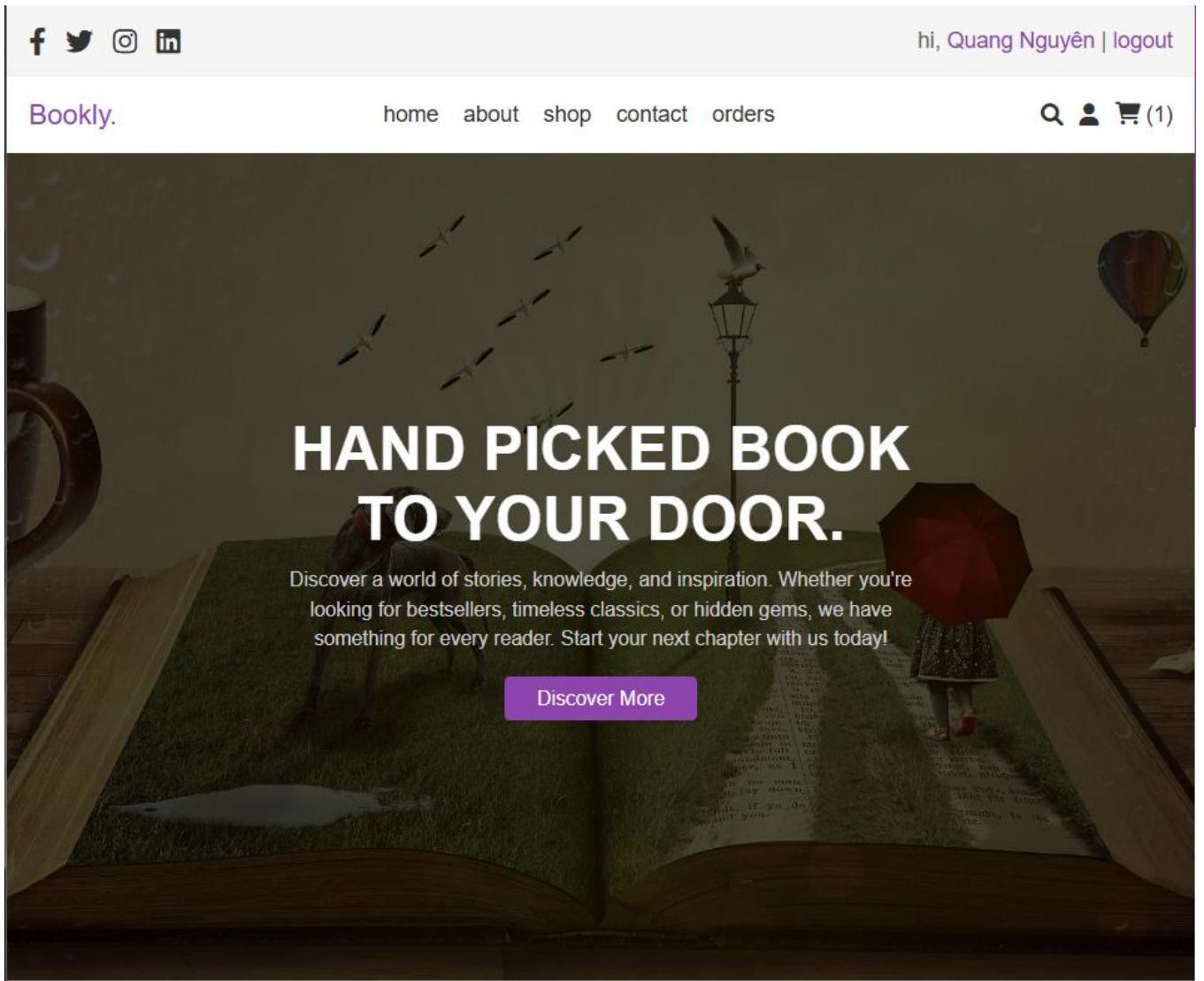
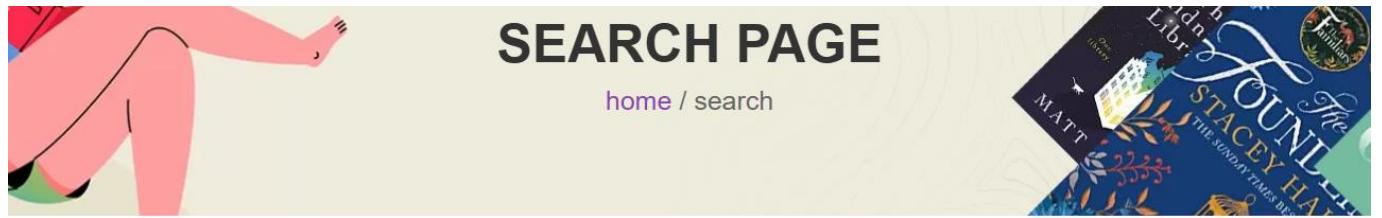


Figure 17: home page

After you click on those type, there are many things to sort like price, types, rating



search products...

Search

\$50/-

Thiên tài bên trái kè điên bên phải

Figure 18: page to find your books

After click on any books, website will bring you to this page to add to wishlist (to buy later if have better price), add to cart (to buy now)

Bookly.

home about shop contact orders

🔍 🚙 ⚡ (1)

## PRODUCTS ADDED

Quá trình hình thành

\$1000/-

1      Update

sub total : \$1000/-

Delete All

grand total : \$1000/-

Continue Shopping

Proceed To Checkout

Figure 19: add to Wishlist or cart

The screenshot shows a mobile application interface. At the top, there is a red box containing the text "Quá trình hình thành (\$1000/- x 1)". Below this, the text "grand total : \$1000/-" is displayed. The main section is titled "PLACE YOUR ORDER". It contains several input fields for user information:

- your name :
- your number :
- your email :
- payment method :
- address line 01 :
- address line 01 :
- city :
- state :
- country :
- pin code :

At the bottom left is a purple button labeled "Order Now".

Firgure 20: checkout

## 4 Challenges

Below are some challenging problems commonly encountered by everyone working on the same project. We acknowledge these issues and aim to address them in more stable future projects.

### 4.1 Database Synchronization

Challenge: Maintaining consistency in the database schema among team members.

Symptoms: Inefficient management of database updates (e.g., adding tables, modifying fields) can lead to discrepancies between the database structure and the application code.

## **4.2 Insufficient Knowledge in Tools/Technologies**

Challenge: Team members may lack adequate knowledge of essential tools like MySQL, leading to challenges in writing efficient queries, designing appropriate schemas, or debugging database issues due to limited prior experience with database management or web development concepts. 4.3 GitHub Version Control Conflicts. And when multiple people work on the same files, conflicts arise when merging code because of lacking experience with Git workflows (e.g., branching, committing, resolving conflicts).

Symptoms: Must resolve conflict when occurring.

## **5 Conclusion**

Through the development of BookStore, our team has successfully built a functional e-commerce web application.

Achievements: We implemented full CRUD (Create, Read, Update, Delete) operations for products and orders, a secure authentication system, and a dynamic shopping cart.

Knowledge Gained: We deepened our understanding of PHP server-side logic, MySQL relational database design, and how to effectively bridge the frontend and backend without relying on heavy frameworks.

Future Work: In the future, we plan to integrate a real payment gateway (like Stripe or PayPal API) instead of just simulating "Credit Card" status, and implement an email notification system for order confirmations.