COS30043 PROJECT 2025

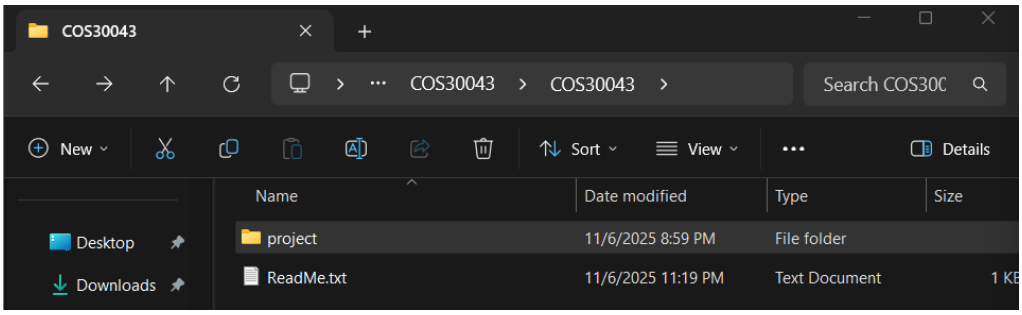
Name: Alison Au Ching Yi

Student ID: 102768672

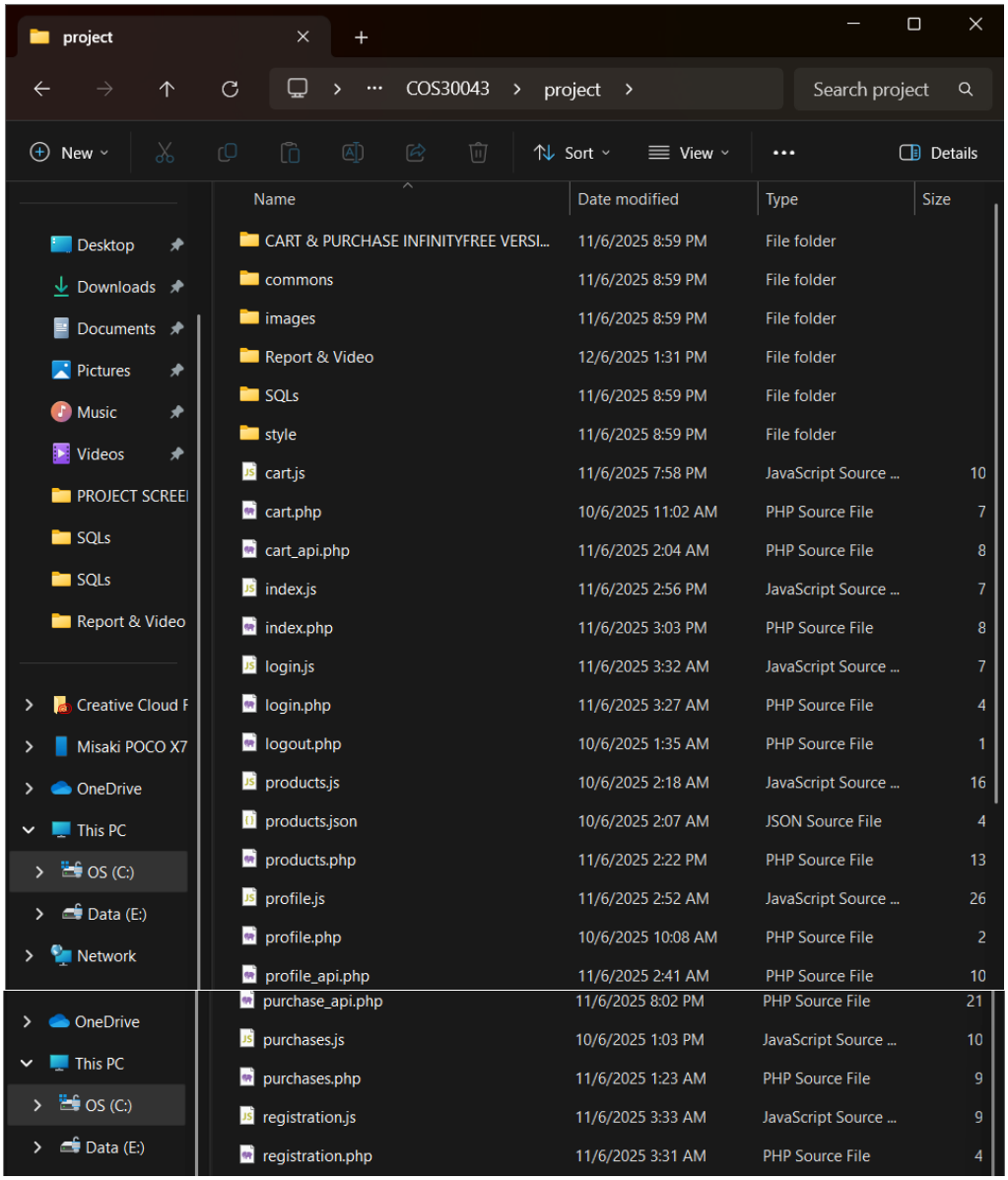
Table of Contents

1. Code Folder Structure

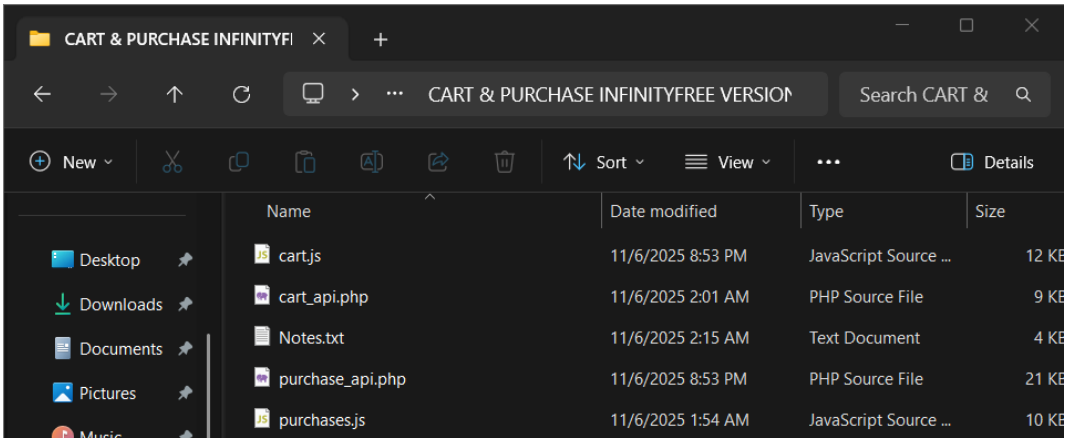
The submitted code is placed in a folder called **COS30043**. This folder should be placed in the xampp/htdocs folder. In **COS30043**, there is a **ReadMe.txt** text file and a folder called **project**.



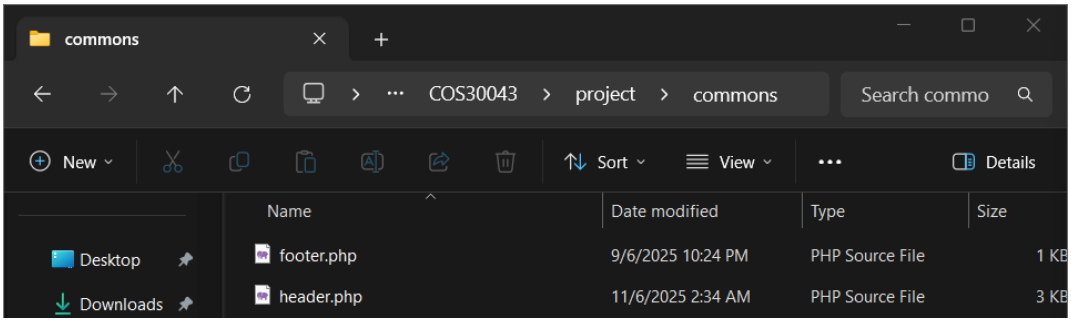
In the **project** folder, it contains all the files needed for the website to work along with other folders that contain the project report and video recording.



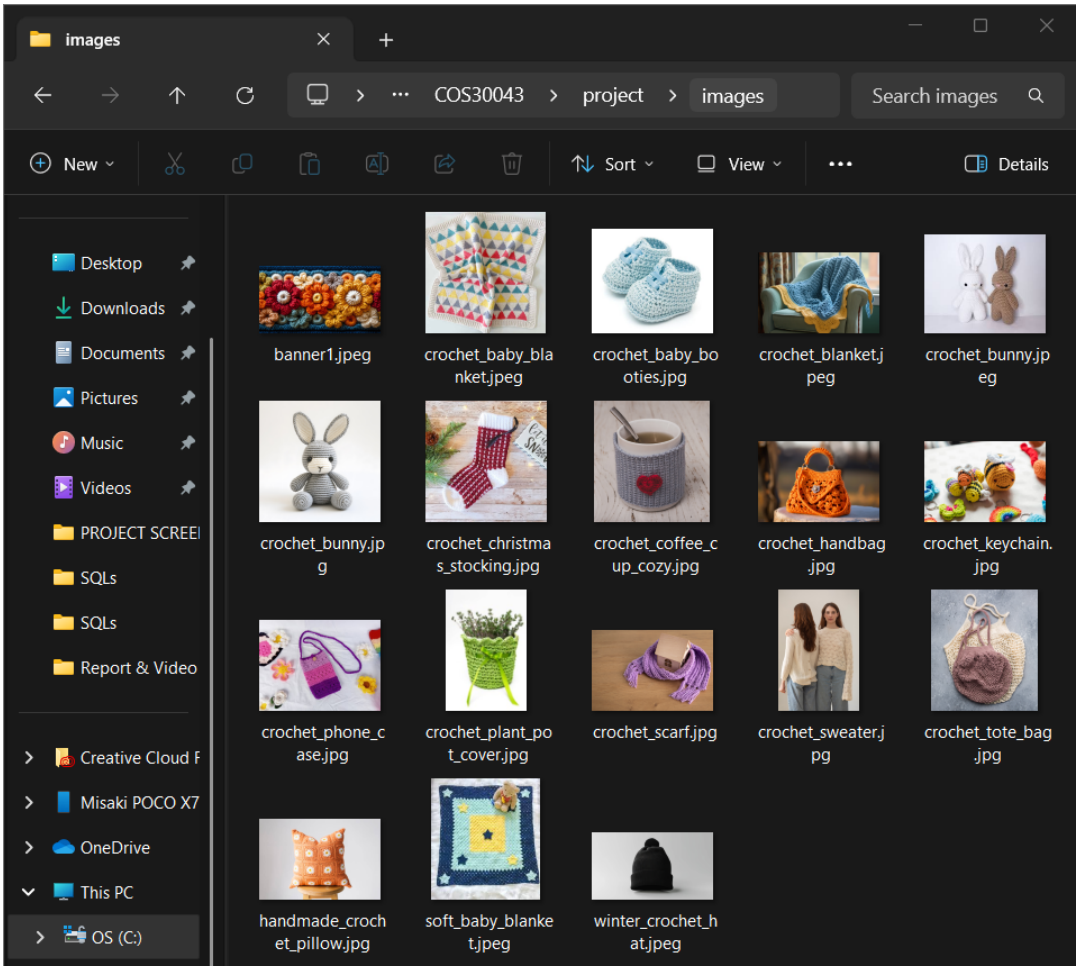
In the **CART & PURCHASE INFINITYFREE VERSION** folder, it contains the code files that are changed to accommodate the limitations of hosting the website on a free website hosting platform called **InfinityFree** - https://www.infinityfree.com. It also contains the Notes.txt text file that describes why these code files have to be modified.



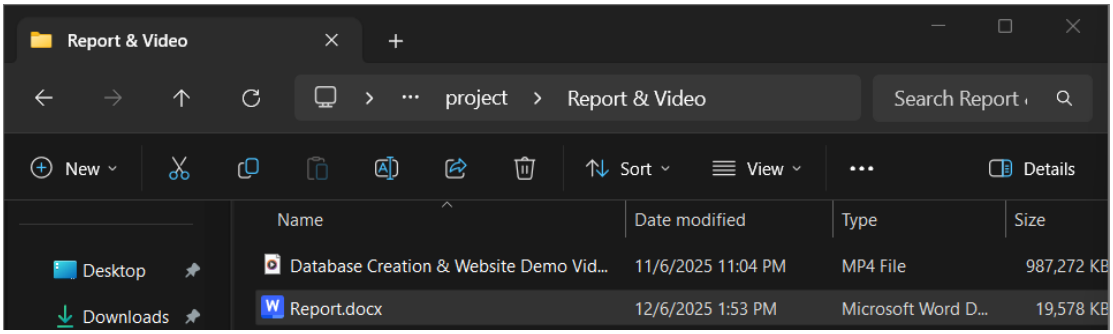
In the **commons** folder, there are the **header.php** and **footer.php** files. These files are the same across all the pages of my website, so it is easier to modify the header and footer through a universal file, that's why the header and footer files are created.



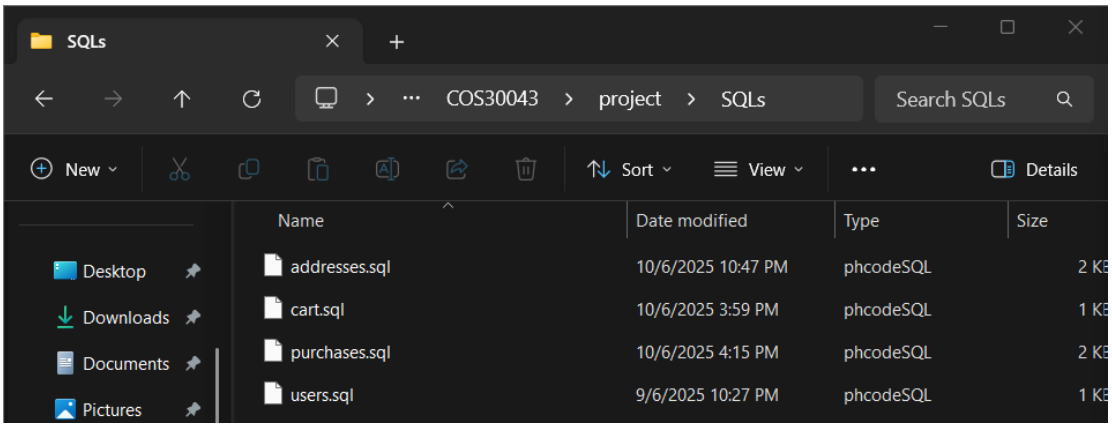
In the **images** folder, there are the various images used on the website such as the product images and the index page banner image. All these images are obtained from **Freepik** - https://www.freepik.com.



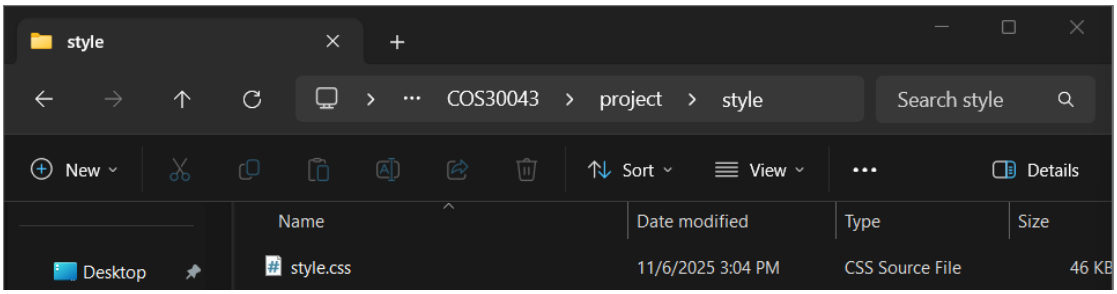
In the **Report & Video** folder, there are the report document and the video recording for this project.



In the **SQLs** folder, it contains the **sql files** needed for table creation in the database.

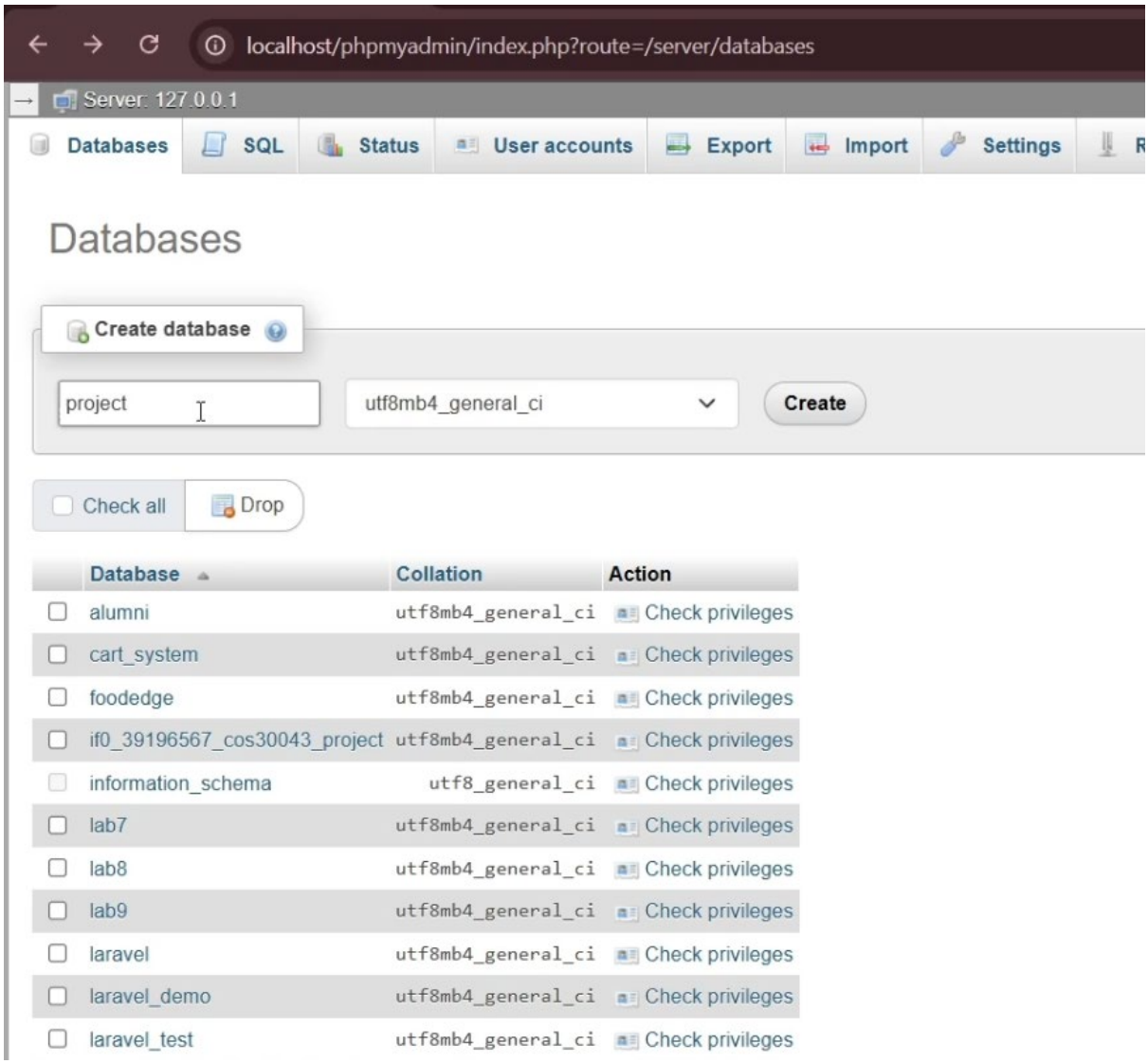


In the **style** folder, it contains the **style.css** file that contains all the styling used for the pages in my website.



2. Database Creation and Setup

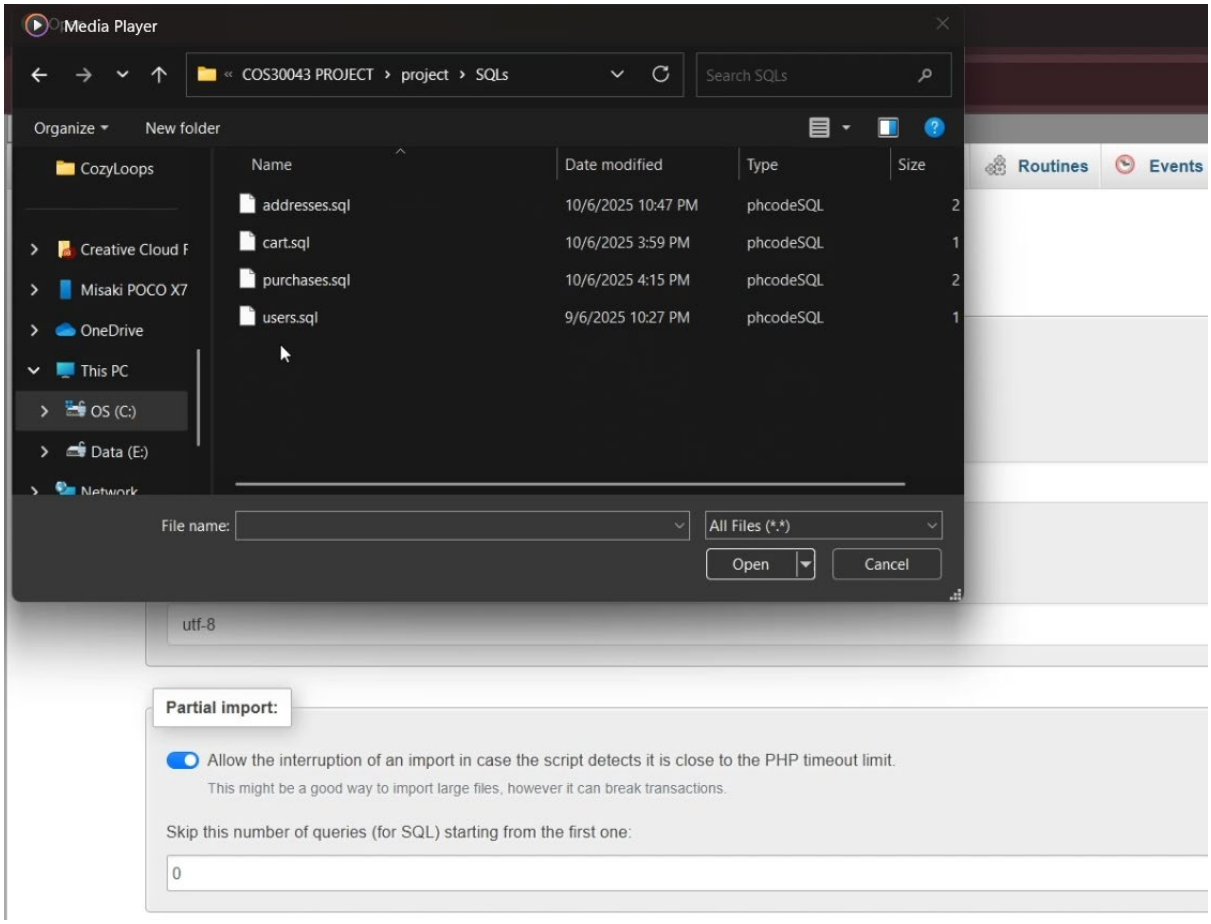
First, access the **phpMyAdmin** page and create a database called **project**.



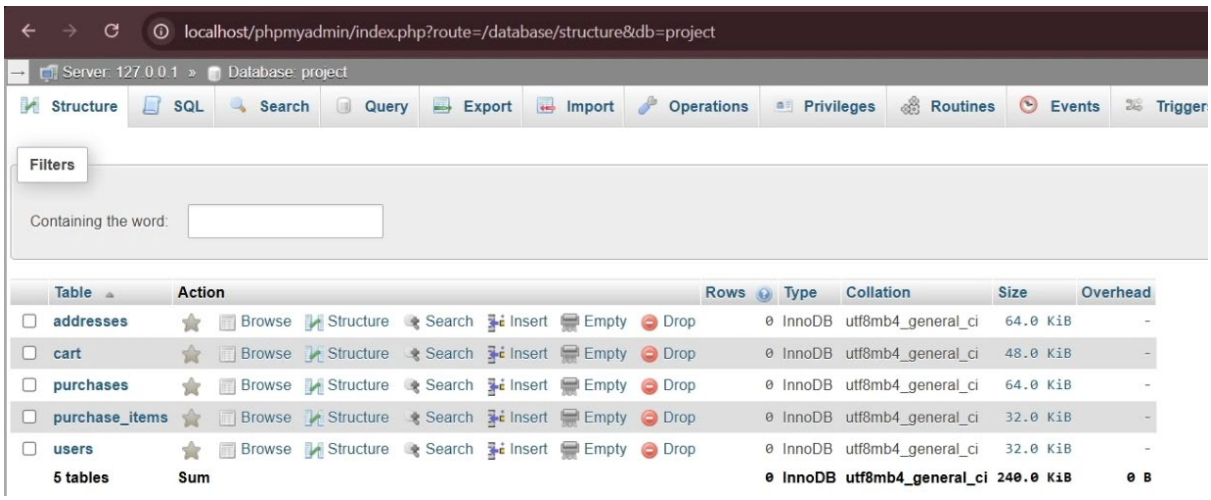
After clicking on the **Create** button, click on **Import**.

Select all **SQL files** to import into the database **project**:

- address.sql
- cart.sql
- purchases.sql
- users.sql



After importing all the SQL files, all the needed tables will be created.



3. Code Functionality

a. Use of header.php and footer.php in the Project

To maintain a consistent layout and improve code reusability, the project uses modular PHP components, particularly header.php and footer.php. These files are included in most other PHP pages to ensure a standardized navigation bar and footer throughout the application.

header.php:

- **Session Start & Authentication Check**:
  The file begins with session_start() to access existing session data. It checks if the user is logged in by verifying the $_SESSION['logged_in'] flag. If the user is not authenticated, they are redirected to login.php.

- **User Information Initialization**:
  If the session is valid, the script extracts the user's ID, name, and email from session variables and stores them in the $user array for use within the page.

- **Navigation Bar**:
  The header.php contains a Bootstrap-based navigation bar that includes links to index.php, products.php, cart.php, and purchases.php.
- **Active Link Highlighting**:
  Each navigation link uses PHP to dynamically assign the active class to the link that matches the currently loaded page using basename($_SERVER['PHP_SELF']).
- **User Dropdown**:
  On the right-hand side of the navbar, a dropdown menu displays the logged-in user's name (escaped with htmlspecialchars() for security) and provides links to the user's profile (profile.php) and the logout page (logout.php).

purchases.js | purchase_api.php | footer.php | header.php

```php
<nav class="navbar navbar-expand-lg sticky-top px-3">
    <a class="navbar-brand" href="index.php">CozyLoops</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav"
        aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse justify-content-end" id="navbarNav">
        <ul class="navbar-nav align-items-center">
            <li class="nav-item">
                <a class="nav-link <?php echo basename($_SERVER['PHP_SELF']) == 'index.php' ? 'active' : ''; ?>" href="index.php">Home</a>
            </li>
            <li class="nav-item">
                <a class="nav-link <?php echo basename($_SERVER['PHP_SELF']) == 'products.php' ? 'active' : ''; ?>" href="products.php">Products</a>
            </li>
            <li class="nav-item">
                <a class="nav-link <?php echo basename($_SERVER['PHP_SELF']) == 'cart.php' ? 'active' : ''; ?>" href="cart.php">Cart</a>
            </li>
            <li class="nav-item">
                <a class="nav-link <?php echo basename($_SERVER['PHP_SELF']) == 'purchases.php' ? 'active' : ''; ?>" href="purchases.php">Purchases</a>
            </li>
            <li class="nav-item dropdown">
                <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">
                    <?php echo htmlspecialchars($user['name']); ?>
                </a>
                <ul class="dropdown-menu dropdown-menu-end">
                    <li><a class="dropdown-item" href="profile.php">Profile</a></li>
                    <li><a class="dropdown-item" href="logout.php">Logout</a></li>
                </ul>
            </li>
        </ul>
    </div>
</nav>
```

footer.php:

Simple Page Footer:
The footer.php provides a simple, centered footer for all pages that include it. It helps maintain branding and professionalism by showing the company name and a copyright.

purchases.js | purchase_api.php | footer.php

```php
<footer class="text-center mt-4">
    <div class="container">
        <p class="mb-0">&copy; 2025 CozyLoops. All rights reserved.</p>
    </div>
</footer>
```

Integration with Other Pages:

Most PHP pages in the system (e.g., index.php, products.php, cart.php, profile.php, purchases.php) include header.php and footer.php using the PHP include or require functions, usually like this:

```
<!-- Header -->
<?php include 'commons/header.php'; ?>
```

```
<!-- Footer -->
<?php include 'commons/footer.php'; ?>
```

This setup has several benefits:

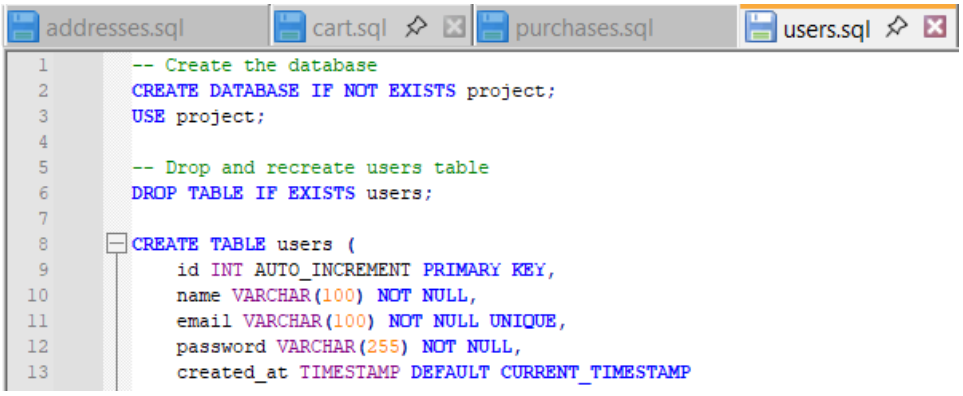
- **Consistency**: Ensures all pages have a consistent navigation bar and footer.
- **Security**: header.php performs an authentication check, so including it on pages that require login automatically protects them.
- **Code Reuse**: Prevents duplication of the same HTML/CSS/PHP code across multiple files.
- **Maintainability**: If the navbar or footer design changes, the update only needs to be made in one file.

b. Use of Sql codes in the Project

The website is supported by a structured MySQL database called project (created in phpMyAdmin), which consists of multiple related tables: users, addresses, cart, purchases, and purchase_items. Each SQL file is responsible for creating and initializing one or more of these tables. Below is a breakdown of how each file works and how the tables are interconnected.

users.sql:

- **Purpose**: This file creates the foundational users table, which stores basic account details for each registered user.
- **Key Columns**:
  - id: Unique identifier for each user (used as a foreign key in other tables).
  - email: Must be unique to prevent duplicate accounts.
  - password: Stores the hashed password.
  - created_at: Automatically records when the account was created.
- **Relation to Other Tables**: Other tables such as cart, addresses, and purchases reference users.id through foreign keys.



addresses.sql:

- **Purpose**: Stores the shipping addresses associated with each user.
- **Key Features**:
  - user_id: Intended to reference the user who owns the address (foreign key constraint commented out).
  - state and country: Required fields with indexing for faster lookup and filtering.
  - created_at and updated_at: Timestamps for tracking changes.
- **Usage in the App**: When users manage their shipping details in the profile page, these are stored in this table.

addresses.sql | cart.sql | purchases.sql | users.sql

```sql
-- Create the database
CREATE DATABASE IF NOT EXISTS project;
USE project;

-- Drop and recreate addresses table with new fields
DROP TABLE IF EXISTS addresses;

CREATE TABLE addresses (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    name VARCHAR(100) NOT NULL,
    phone VARCHAR(20) NOT NULL,
    unit_number VARCHAR(50), -- Optional unit number
    street VARCHAR(255) NOT NULL,
    city VARCHAR(100) NOT NULL,
    state VARCHAR(100) NOT NULL, -- New required field
    postcode VARCHAR(20) NOT NULL,
    country VARCHAR(100) NOT NULL DEFAULT 'Malaysia', -- New required field with default
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    -- Add foreign key constraint if users table exists
    -- FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,

    -- Add indexes for better performance
    INDEX idx_user_id (user_id),
    INDEX idx_country (country),
    INDEX idx_state (state)
);
```

cart.sql:

- **Purpose**: Stores items that users have added to their cart before checking out.
- **Key Features**:
  - Linked to users via user_id.
  - Stores product info redundantly (name, price, image) to preserve item details even if the original product listing changes later.
  - Indexes for quick user-specific cart lookups.
- **Usage in the App**: Data from this table is shown in the cart.php page and used during the checkout process.

addresses.sql | cart.sql | purchases.sql | users.sql

```sql
-- Create the database
CREATE DATABASE IF NOT EXISTS project;
USE project;

-- Drop and recreate users table
DROP TABLE IF EXISTS cart;

CREATE TABLE cart (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    product_id INT NOT NULL,
    name VARCHAR(255) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    image VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    INDEX idx_user_id (user_id),
    INDEX idx_product_user (product_id, user_id)
);
```

purchases.sql:

- **Purpose**: Handles orders and the items within each order.
- **purchases Table**:
  - Tracks the overall purchase: who made it (user_id), the total cost, shipping cost, order number, and status.
  - order_status uses an ENUM to enforce valid states (e.g., pending, shipped).
- **purchase_items Table**:
  - Each row corresponds to an individual product within an order.
  - Links to purchases via purchase_id.
  - Stores full product data redundantly (e.g., name, price, image) for consistency over time.
- **Usage in the App**: This is used in the purchases.php page to display order history and itemized purchases per user.

addresses.sql | cart.sql | purchases.sql | users.sql

```sql
-- Create the database
CREATE DATABASE IF NOT EXISTS project;
USE project;

-- Drop and recreate tables
DROP TABLE IF EXISTS purchase_items;
DROP TABLE IF EXISTS purchases;

-- Create purchases table to store order information
CREATE TABLE IF NOT EXISTS purchases (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    order_number VARCHAR(50) UNIQUE NOT NULL,
    total_amount DECIMAL(10,2) NOT NULL,
    shipping_cost DECIMAL(10,2) DEFAULT 0.00,
    order_status ENUM('pending', 'processing', 'shipped', 'delivered', 'cancelled') DEFAULT
    'pending',
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    INDEX idx_user_id (user_id),
    INDEX idx_order_number (order_number)
);

-- Create purchase_items table to store individual items in each order
CREATE TABLE IF NOT EXISTS purchase_items (
    id INT AUTO_INCREMENT PRIMARY KEY,
    purchase_id INT NOT NULL,
    product_id INT NOT NULL,
    name VARCHAR(255) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    quantity INT NOT NULL,
    image VARCHAR(255),
    FOREIGN KEY (purchase_id) REFERENCES purchases(id) ON DELETE CASCADE,
    INDEX idx_purchase_id (purchase_id)
);
```

How They All Link Together:

- users.id is the primary key that connects all user-related data.
- addresses.user_id, cart.user_id, and purchases.user_id ensure all records are linked to the corresponding user.
- purchases.id is referenced by purchase_items.purchase_id to tie each product to a specific order.

c. Use of styling codes in the Project

The style.css file is a **custom stylesheet** that defines the visual design and responsive behavior of the e-commerce website built with PHP and Vuetify. It applies consistent and customized styles across all HTML elements and PHP pages using shared layout components like header.php, footer.php, and content pages such as index.php, login.php, registration.php, profile.php, products.php, cart.php, and purchases.php.

Below is a breakdown of how it works and what key sections do.

1. Global Layout and Page Structure

- Styling on html and body class ensures full-page height and a flexible layout so that content can stretch and the footer stays at the bottom.
- Applies a soft pink gradient background site-wide.
- Uses a clean, modern sans-serif font.

2. Navigation Bar Styling (header.php)

- Styling on classes such as .navbar, .navbar-nav .nav-link colors and styles the navigation bar (used in header.php) with a soft gradient and brown-pink theme.
- .navbar-nav .nav-link:active and .navbar-nav .nav-link:hover highlights the active page and hover state for better user navigation feedback.

3. Footer Styling (footer.php)

- Footer class provides a fixed footer with matching colors to the header.
- Used consistently across all pages by including footer.php.

4. Login and Registration Page Styling

- Styling .login-wrapper, .login-form-section and .welcome-section implements a **split-screen layout** with a styled login form and a welcoming right panel.
- Adds animations, shimmer effects, and visually distinct success/error messages.
- Ensures consistency between login and registration interfaces using shared .large-input and .large-btn classes.

5. Product and Cart Page Enhancements

- Product cards are styled with shadows, hover animations, and responsive images using classes like .product-card, .image-container, and .btn-custom.
- Cart items are neatly boxed and styled using .cart-container, .cart-item, and .remove-all-btn.

6. Purchases and Profile Page Styling

- Purchase records are styled with colored status badges like .status-pending, .status-shipped, etc.
- The profile page and address cards use .custom-card and .address-card with soft shadows and button hovers for editing and deleting.

7. Responsive Design

- Multiple responsive breakpoints are used to adapt layouts for tablets and mobile devices by using the @media rule.
- Ensures all UI components are mobile-friendly, such as the login form, navbar, and product grid.

8. Animations and Effects

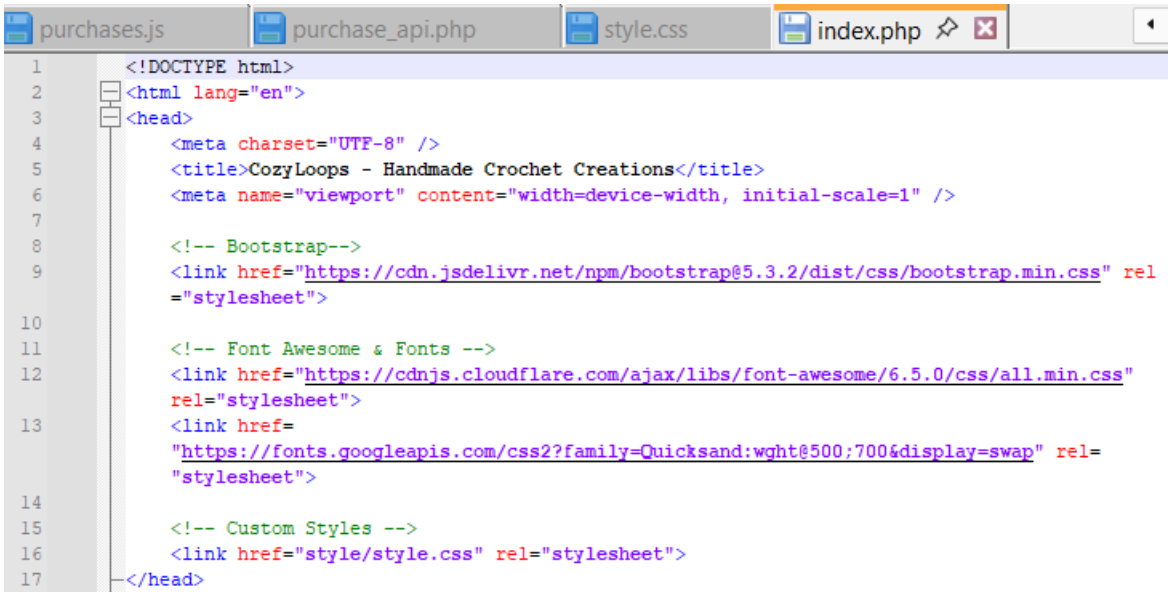
- Uses @keyframes for shimmer and fade-in effects, enhancing user experience.
- Applies hover transformations to elements like category cards (.index_category) and buttons.

9. Badges and Visual Labels

- Badges like .top-products-badge and .latest-items-badge are used to highlight featured items.
- Custom colors and icons help differentiate between product types and states.

How It Connects to the PHP Files:

The style.css is linked in each PHP page using a <link> tag in the <head> section (usually included in header.php), like this:



- header.php and footer.php include site-wide layout components styled by style.css.
- Pages like login.php, register.php, products.php, and profile.php use specific classes (e.g., .login-wrapper, .product-card, .custom-card) defined in this stylesheet.
- Vuetify elements are enhanced by targeting their classes (e.g., .v-field, .v-alert) for consistent integration.

d. Use of login.php, login.js and logout.php in the Project

The login system in this application is implemented using a **PHP backend (login.php)** and a **Vue 3 + Vuetify frontend (login.js) and also logout.php**. Together, they provide a **secure and user-friendly** login interface with real-time form validation and session-based authentication.

login.php:

This file handles **session control**, **validates login credentials**, and **responds with JSON** for the frontend to interpret.

1. Initial Configuration

- Starts a session and suppresses PHP errors to avoid output before JSON responses.
- Uses ob_start() to prevent accidental output that could corrupt JSON.

```php
<?php
// Prevent any output before JSON response
ini_set('display_errors', 0);
error_reporting(0);
ob_start();
```

2. Handling POST Requests (AJAX Login)

- Checks for POST requests with JSON content.
- Parses the email and password fields from the request body.

```php
// Handle POST requests for login
if (
    $_SERVER['REQUEST_METHOD'] === 'POST' &&
    isset($_SERVER['CONTENT_TYPE']) &&
    strpos($_SERVER['CONTENT_TYPE'], 'application/json') === 0
) {
    // Clear any previous output
    ob_clean();
    header("Content-Type: application/json");

    try {
        // Parse JSON input
        $input = json_decode(file_get_contents('php://input'), true);
        if (!$input || !isset($input['email'], $input['password'])) {
            http_response_code(400);
            echo json_encode(['message' => 'Invalid input']);
            exit;
        }

        $email = trim($input['email']);
        $password = $input['password'];
```

3. Database Verification

- Connects to the MySQL database.
- Uses **prepared statements** to prevent SQL injection when looking up the user by email.

```php
        // Database connection using prepared statements
        $conn = mysqli_connect('localhost', 'root', '', 'project');
        if (!$conn) {
            http_response_code(500);
            echo json_encode(['message' => 'Database connection failed: ' .
            mysqli_connect_error()]);
            exit;
        }

        mysqli_set_charset($conn, 'utf8');

        // Use prepared statement to prevent SQL injection
        $query = "SELECT * FROM users WHERE email = ?";
        $stmt = mysqli_prepare($conn, $query);

        if (!$stmt) {
            http_response_code(500);
            echo json_encode(['message' => 'Database query failed']);
            mysqli_close($conn);
            exit;
        }
```

4. Password Validation and Session Assignment

- Validates the password using password_verify() (for hashed passwords).
- If correct, stores user info in $_SESSION to mark them as logged in.
- Sends a JSON response back to the frontend including a redirect URL.

```php
        mysqli_stmt_bind_param($stmt, "s", $email);
        mysqli_stmt_execute($stmt);
        $result = mysqli_stmt_get_result($stmt);

        if ($user = mysqli_fetch_assoc($result)) {
            if (password_verify($password, $user['password'])) {
                // Store user data in session
                $_SESSION['user_id'] = $user['id'];
                $_SESSION['user_name'] = $user['name'];
                $_SESSION['user_email'] = $user['email'];
                $_SESSION['logged_in'] = true;

                echo json_encode([
                    'message' => 'Login successful',
                    'user' => [
                        'id' => $user['id'],
                        'name' => $user['name'],
                        'email' => $user['email']
                    ],
                    'redirectUrl' => 'index.php'
                ]);
            } else {
                http_response_code(401);
                echo json_encode(['message' => 'Incorrect password']);
            }
        } else {
            http_response_code(404);
            echo json_encode(['message' => 'User not found']);
        }

        mysqli_stmt_close($stmt);
        mysqli_close($conn);
    } catch (Exception $e) {
        http_response_code(500);
        echo json_encode(['message' => 'Server error: ' . $e->getMessage()]);
    }

    exit;
}
```

5. Redirect Logged-in Users

- If the user is already logged in (based on session), they are redirected automatically to the homepage.

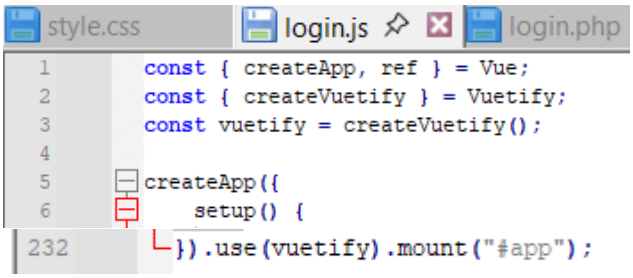
```php
// Check if user is already logged in via session
if (isset($_SESSION['logged_in']) && $_SESSION['logged_in'] === true) {
    header('Location: index.php');
    exit;
}
?>
```

login.js:

This file builds a **modern, reactive login form** and interacts with login.php via AJAX. It also provides real-time validation and UI feedback.

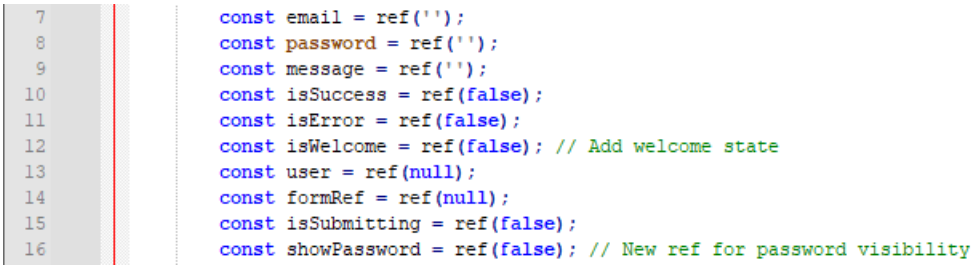
1. Vue App Initialization

- Mounts a Vue app on the #app div in login.php.
- Uses Vuetify UI components for styling and layout.



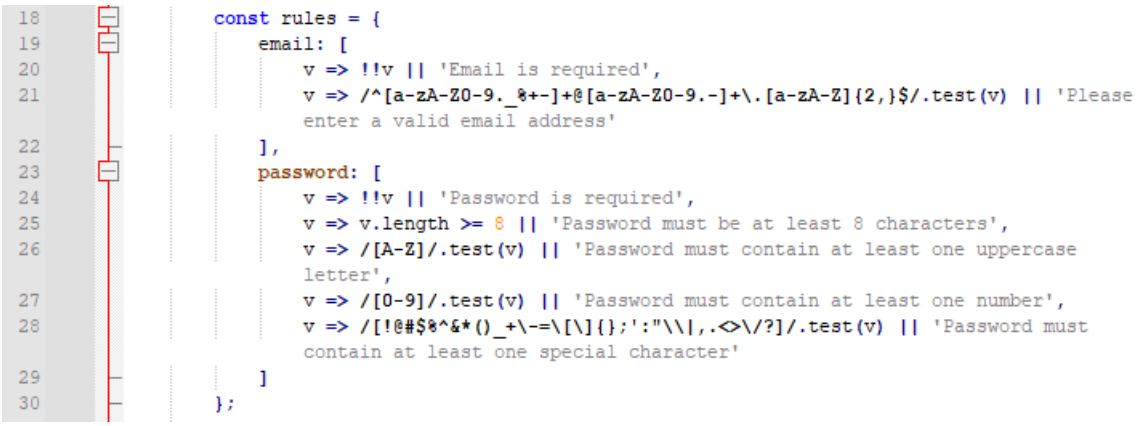
2. Reactive Form Inputs

- Data bindings for user input.
- Linked to v-text-field components for live updates and validation.



3. Form Validation Rules

- Ensures:
  - Email is valid and required.
  - Password is at least 8 characters and includes a number, uppercase letter, and special character.



4. Login Submission

- On form submit:
  - Validates form input.
  - Sends a POST request to login.php with email and password as JSON.
  - Handles the response:
    - If successful: shows a welcome message, then redirects.
    - If failed: shows error messages and triggers a shake animation.

```
32      const loginUser = async () => {
33          const { valid } = await formRef.value.validate();
34          if (!valid) {
35              isError.value = true;
36              isSuccess.value = false;
37              isWelcome.value = false;
38              message.value = "Please correct all form errors before submitting.";
39              // Add shake animation to form
40              const loginContainer = document.querySelector('.login-container');
41              if (loginContainer) {
42                  loginContainer.classList.add('error-shake');
43                  setTimeout(() => {
44                      loginContainer.classList.remove('error-shake');
45                  }, 500);
46              }
47              return;
48          }
49
50          isSuccess.value = false;
51          isError.value = false;
52          isWelcome.value = false;
53          message.value = '';
54          isSubmitting.value = true;
55
56          try {
57              const response = await fetch('login.php', {
58                  method: 'POST',
59                  headers: { 'Content-Type': 'application/json' },
60                  body: JSON.stringify({
61                      email: email.value,
62                      password: password.value
63                  })
64              });
65
66              const result = await response.json();
67
68              if (response.ok) {
69                  user.value = result.user;
70                  message.value = result.message || 'Login successful!';
71                  isSuccess.value = true;
72
73                  // After 1 second, replace with welcome message and change styling
74                  setTimeout(() => {
75                      message.value = `Welcome, ${result.user.name || 'User'}!`;
76                      // Change to welcome styling
77                      isSuccess.value = false;
78                      isError.value = false;
79                      isWelcome.value = true; // Set welcome state
80
81                      // After another 2 seconds, redirect
82                      setTimeout(() => {
83                          window.location.href = result.redirectUrl || 'index.php';
84                      }, 2000);
85                  }, 1000);
```

```
86
87                  } else {
88                      message.value = result.message || 'Login failed';
89                      isError.value = true;
90                      isSuccess.value = false;
91                      isWelcome.value = false;
92
93                      // Add shake animation to form on error
94                      const loginContainer = document.querySelector('.login-container');
95                      if (loginContainer) {
96                          loginContainer.classList.add('error-shake');
97                          setTimeout(() => {
98                              loginContainer.classList.remove('error-shake');
99                          }, 500);
100                     }
101                 }
102             } catch (error) {
103                 message.value = 'Network error: ' + error.message;
104                 isError.value = true;
105                 isSuccess.value = false;
106                 isWelcome.value = false;
107
108                 // Add shake animation to form on network error
109                 const loginContainer = document.querySelector('.login-container');
110                 if (loginContainer) {
111                     loginContainer.classList.add('error-shake');
112                     setTimeout(() => {
113                         loginContainer.classList.remove('error-shake');
114                     }, 500);
115                 }
116             } finally {
117                 isSubmitting.value = false;
118             }
119         };
```

5. User Feedback Mechanisms

- message, isSuccess, isError, and isWelcome manage alert display.
- getAlertClass() dynamically sets CSS classes for alerts.
- Uses timers (setTimeout) to delay welcome message and redirect.

```
125         // Function to get custom alert class based on message type
126         const getAlertClass = () => {
127             if (isError.value) return 'login-error-message';
128             if (isSuccess.value) return 'login-success-message';
129             if (isWelcome.value) return 'welcome-message-elegant'; // Use existing welcome
                styling
130             return '';
131         };
```

6. Password Visibility Toggle

- Allows users to toggle password visibility by clicking the eye icon.

```
196                                 :append-inner-icon="showPassword ? 'mdi-eye' :
                                    'mdi-eye-off'"
197                                 @click:append-inner="showPassword = !showPassword"
```

7. Navigation to Registration

- When users click “Sign Up,” they are redirected to the registration form.

```
121         const redirectToRegister = () => {
122             window.location.href = 'registration.php';
123         };
```

8. Template for Login Page

- The forms on the login page are coded using Vuetify.

```
template: `
    <v-app>
        <v-main style="padding: 0 !important;">
            <div class="login-wrapper">
                <!-- Left side - Login Form -->
                <div class="login-form-section">
                    <div class="login-container">
                        <h1 class="login-title">Login</h1>

                        <!-- Custom Alert for messages -->
                        <div
                            v-if="message"
                            :class="getAlertClass()"
                            class="alert-message"
                        >
                            {{ message }}
                        </div>

                        <v-form ref="formRef" lazy-validation @keyup.enter="loginUser"
                        class="login-form">
                            <div class="form-group">
                                <label for="email" class="form-label">EMAIL</label>
                                <v-text-field
                                    v-model="email"
                                    :rules="rules.email"
                                    type="email"
                                    required
                                    variant="outlined"
                                    density="comfortable"
                                    placeholder="Enter your email"
                                    hide-details="auto"
                                    class="large-input"
                                ></v-text-field>
                            </div>

                            <div class="form-group">
                                <label for="password"
                                class="form-label">PASSWORD</label>
                                <v-text-field
                                    v-model="password"
                                    :rules="rules.password"
                                    :type="showPassword ? 'text' : 'password'"
                                    required
                                    variant="outlined"
                                    density="comfortable"
                                    placeholder="Enter your password"
                                    hide-details="auto"
                                    class="large-input"
                                    :append-inner-icon="showPassword ? 'mdi-eye' :
                                    'mdi-eye-off'"
                                    @click:append-inner="showPassword = !showPassword"
                                ></v-text-field>
                            </div>

                            <v-btn
                                @click="loginUser"
                                :loading="isSubmitting"
                                :disabled="isSubmitting"
                                class="btn-brown large-btn"
                                block
                            >
                                <span v-if="isSubmitting">Signing In...</span>
                                <span v-else>Continue ></span>
                            </v-btn>
                        </v-form>
                    </div>
                </div>

```

```
                    <!-- Right side - Welcome/Register Section -->
                    <div class="welcome-section">
                        <div class="welcome-content">
                            <h1 class="welcome-title">Join Our Community</h1>
                            <p class="welcome-text">Don't have an account?</p>
                            <v-btn
                                @click="redirectToRegister"
                                class="btn-brown large-btn signup-btn"
                            >
                                Sign Up
                            </v-btn>
                        </div>
                    </div>
                </div>
            </v-main>
        </v-app>
    `
```

logout.php:
The logout.php file handles **secure user logout** by clearing session data and redirecting the user to the login page. It ensures that once a user logs out, they can no longer access protected pages unless they log in again.

1. Start the Session

- Initializes the session so it can be cleared and destroyed.

```php
<?php
session_start();
```

2. Clear Session Data

- Removes all data stored in the current session (e.g., user ID, email, name, login status).

```php
// Clear all session variables
$_SESSION = array();
```

3. Destroy Session Cookie

- Removes the PHP session cookie by setting its expiration to a past time.
- Ensures the browser no longer holds a valid session identifier.

```php
// Destroy the session cookie
if (isset($_COOKIE[session_name()])) {
    setcookie(session_name(), '', time() - 42000, '/');
}
```

4. Destroy the Session

- Completely ends the session on the server.
- Prevents any further access to session data.

```php
// Destroy the session
session_destroy();
```

5. Redirect to Login Page

- Sends the user back to login.php, effectively logging them out and requiring them to sign in again.

```php
// Redirect to login page
header('Location: login.php');
exit;
?>
```

How It Connects to Other Files:

- **Trigger**: The logout link is typically found in a dropdown menu in header.php:
- Effect: When clicked, it:
    - Clears the user session.
    - Redirects them to login.php.
    - Prevents access to any protected pages (which check $_SESSION['logged_in']).

[v-form] - UI form using Vuetify, bound to Vue refs

↓

[login.js] - handles form input, validation, sends login data via AJAX

↓

[login.php] - verifies credentials, starts session, returns JSON response

↓

(login successful) - session set → user redirected to homepage (index.php)

[logout.php] - clears session data, invalidates cookie, redirects to login

- The user interacts with a Vuetify-based login form (v-form) rendered by login.js.
- Upon submission, login.js:
    - Validates the form.
    - Sends the credentials via a POST request in JSON format to login.php.
- login.php:
    - Verifies the user's email and password using a secure, hashed comparison.
    - If successful, it stores user data in PHP $_SESSION.
    - Returns a JSON response with the login message and redirect URL.
- The frontend (login.js) then shows feedback (e.g., welcome message) and redirects the user after a delay.
- When the user clicks "Logout" (usually from the dropdown menu in header.php), they are taken to logout.php.
- logout.php:
    - Starts the session.
    - Clears all session variables ($_SESSION = []).
    - Invalidates the session cookie.
    - Calls session_destroy() to fully end the session.
    - Redirects the user to login.php.
- After logout, any protected pages (that check for $_SESSION['logged_in']) will redirect the user back to the login screen.

e. Use of registration.php, registration.js and registration.php in the Project

registration.php

This file processes user registration by receiving form data, validating it, and saving the new user to the database.

1. Receiving JSON Data

- The server expects POST requests with application/json content type.
- Upon receiving data, it decodes the JSON, validates required fields (name, email, password), and checks if the email is already registered.

```php
// Decode JSON input
$input = json_decode(file_get_contents('php://input'), true);
if (!$input || !isset($input['name'], $input['email'], $input['password'])) {
    http_response_code(400);
    echo json_encode(['message' => 'Invalid input']);
    exit;
}

// Extract values
$name = $input['name'];
$email = $input['email'];
$password = $input['password'];
```

2. Checking if Email Exists

- It queries the database to check if the email already exists.
- If found, it returns an error message with HTTP status 409 (Conflict).

```php
// Connect to MySQL
$conn = mysqli_connect('localhost', 'root', '', 'project');
if (!$conn) {
    http_response_code(500);
    echo json_encode(['message' => 'Database connection failed']);
    exit;
}

// Ensure UTF-8 encoding
mysqli_set_charset($conn, 'utf8');

// Check if email already exists
$stmt = $conn->prepare("SELECT id FROM users WHERE email = ?");
$stmt->bind_param("s", $email);
$stmt->execute();
$stmt->store_result();

if ($stmt->num_rows > 0) {
    http_response_code(409);
    echo json_encode(['message' => 'Email already registered']);
    exit;
}
```

3. Password Hashing

- The password is hashed using password_hash() to ensure security before saving it to the database.

```php
// Hash the password
$passwordHash = password_hash($password, PASSWORD_DEFAULT);
```

4. Inserting the New User

- After validating and hashing the password, the new user is inserted into the database.
- If successful, it responds with a success message and a redirect URL (login.php).
- If insertion fails, an error message is returned.

```php
    // Insert new user
    $stmt = $conn->prepare("INSERT INTO users (name, email, password) VALUES (?, ?, ?)");
    $stmt->bind_param("sss", $name, $email, $passwordHash);

    if ($stmt->execute()) {
        http_response_code(201);
        echo json_encode(['message' => 'User registered successfully', 'redirectUrl' =>
        'login.php']);
    } else {
        http_response_code(500);
        echo json_encode(['message' => 'Failed to register user']);
    }

    // Clean up
    $stmt->close();
    mysqli_close($conn);
    exit;
}
?>
```

registration.js

This file provides the client-side logic for a registration form, including validation, AJAX submission, and user feedback.

1. Vue App Initialization

- Initializes a Vue app with reactive form fields (name, email, password, confirmPassword) using Vuetify components.
- Mounts the app on the #app div in registration.php.

```javascript
createApp({
    setup() {
        const name = ref('');
        const email = ref('');
        const password = ref('');
        const confirmPassword = ref('');
        const message = ref('');
        const isSuccess = ref(false);
        const isError = ref(false);
        const isWelcome = ref(false);
        const formRef = ref(null);
        const isSubmitting = ref(false);
        const showPassword = ref(false); // New ref for password visibility
        const showConfirmPassword = ref(false); // New ref for confirm password visibility
```

2. Real-Time Validation

- Defines validation rules for each field using Vuetify's form validation system.
- Validation is applied to ensure:
    - Name contains only letters.
    - Email is in the correct format.
    - Password is strong (at least 8 characters, one uppercase letter, one number, and one special character).
    - Passwords match.

```
const rules = {
    name: [
        (v) => !!v || 'Name is required',
        (v) => /^[A-Za-z\s]+$/.test(v) || 'Only letters allowed'
    ],

    email: [
        (v) => !!v || 'Email is required',
        (v) => /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(v) || 'Invalid email'
    ],

    password: [
        (v) => !!v || 'Password is required',
        (v) => v.length >= 8 || 'Password must be at least 8 characters',
        (v) => /[A-Z]/.test(v) || 'Password must contain at least one uppercase
        letter',
        (v) => /[0-9]/.test(v) || 'Password must contain at least one number',
        (v) => /[!@#$%^&*()_+\-=\[\]{};':"\\|,.<>\/?]/.test(v) || 'Password must
        contain at least one special character',
    ],

    confirmPassword: [
        (v) => !!v || 'Confirm Password is required',
        (v) => v === password.value || 'Passwords do not match'
    ]
};
```

3. Form Submission and User Feedback

- On form submission, the registerUser function validates the form.
- If validation fails, an error message is shown, and the form shakes to indicate the issue.
- On success:
    - A success message is shown, and after a brief delay, the user is redirected to login.php.

```
const registerUser = async () => {
    const { valid } = await formRef.value.validate();
    if (!valid) {
        isError.value = true;
        isSuccess.value = false;
        isWelcome.value = false;
        message.value = "Please correct all form errors before submitting.";
        // Add shake animation to form
        const registrationContainer = document.querySelector('.login-container');
        if (registrationContainer) {
            registrationContainer.classList.add('error-shake');
            setTimeout(() => {
                registrationContainer.classList.remove('error-shake');
            }, 500);
        }
        return;
    }

    isSuccess.value = false;
    isError.value = false;
    isWelcome.value = false;
    message.value = '';

    try {
        isSubmitting.value = true;
        const response = await fetch("registration.php", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({
                name: name.value,
                email: email.value,
                password: password.value
            })
        });
```

```
            const result = await response.json();

            if (response.ok) {
                message.value = result.message;
                isSuccess.value = true;

                // After 1 second, replace with welcome message and change styling
                setTimeout(() => {
                    message.value = `Welcome, ${name.value}!`;
                    // Change to welcome styling
                    isSuccess.value = false;
                    isError.value = false;
                    isWelcome.value = true;

                    clearForm();
                    formRef.value.resetValidation();

                    // After another 2 seconds, redirect
                    setTimeout(() => {
                        if (result.redirectUrl) {
                            window.location.href = result.redirectUrl;
                        } else {
                            window.location.href = 'login.php';
                        }
                    }, 2000);
                }, 1000);

            } else {
                message.value = result.message || "Registration failed.";
                isError.value = true;
                isSuccess.value = false;
                isWelcome.value = false;

                // Add shake animation to form on error
                const registrationContainer = document.querySelector('.login-container'
                );
                if (registrationContainer) {
                    registrationContainer.classList.add('error-shake');
                    setTimeout(() => {
                        registrationContainer.classList.remove('error-shake');
                    }, 500);
                }
            }
        } catch (error) {
            message.value = "Network error: " + error.message;
            isError.value = true;
            isSuccess.value = false;
            isWelcome.value = false;

            // Add shake animation to form on network error
            const registrationContainer = document.querySelector('.login-container');
            if (registrationContainer) {
                registrationContainer.classList.add('error-shake');
                setTimeout(() => {
                    registrationContainer.classList.remove('error-shake');
                }, 500);
            }
        } finally {
            isSubmitting.value = false;
        }
    };
```

4. User Feedback Mechanisms

- message, isSuccess, isError, and isWelcome manage alert display.
- getAlertClass() dynamically sets CSS classes for alerts.
- Uses timers (setTimeout) to delay welcome message and redirect.

```
    // Function to get custom alert class based on message type
    const getAlertClass = () => {
        if (isError.value) return 'login-error-message';
        if (isSuccess.value) return 'login-success-message';
        if (isWelcome.value) return 'welcome-message-elegant';
        return '';
    };
```

5. Password Visibility Toggle

- Allows users to toggle password visibility by clicking the eye icon.

```
                                        :append-inner-icon="showPassword ? 'mdi-eye' :
                                        'mdi-eye-off'"
                                        @click:append-inner="showPassword = !showPassword"
```

6. Navigation to Login

- When users click “Sign In,” they are redirected to the login form.

```
const redirectToLogin = () => {
    window.location.href = 'login.php';
};
```

7. Template for Registration Page

- The forms on the registration page are coded using Vuetify.

```
template: `
    <v-app>
        <v-main style="padding: 0 !important;">
            <div class="login-wrapper">
                <!-- Left side - Registration Form -->
                <div class="login-form-section">
                    <div class="login-container">
                        <h1 class="login-title">Registration</h1>

                        <!-- Custom Alert for messages -->
                        <div
                            v-if="message"
                            :class="getAlertClass()"
                            class="alert-message"
                        >
                            {{ message }}
                        </div>

                        <v-form ref="formRef" lazy-validation
                        @keyup.enter="registerUser" class="login-form">
                            <div class="form-group">
                                <label for="name" class="form-label">NAME</label>
                                <v-text-field
                                    v-model="name"
                                    :rules="rules.name"
                                    required
                                    variant="outlined"
                                    density="comfortable"
                                    placeholder="Enter your full name"
                                    hide-details="auto"
                                    class="large-input"
                                ></v-text-field>
                            </div>

                            <div class="form-group">
                                <label for="email" class="form-label">EMAIL</label>
                                <v-text-field
                                    v-model="email"
                                    :rules="rules.email"
                                    type="email"
                                    required
                                    variant="outlined"
                                    density="comfortable"
                                    placeholder="Enter your email"
                                    hide-details="auto"
                                    class="large-input"
                                ></v-text-field>
                            </div>
```

```
                                <div class="form-group">
                                    <label for="password"
                                    class="form-label">PASSWORD</label>
                                    <v-text-field
                                        v-model="password"
                                        :rules="rules.password"
                                        :type="showPassword ? 'text' : 'password'"
                                        required
                                        variant="outlined"
                                        density="comfortable"
                                        placeholder="Enter your password"
                                        hide-details="auto"
                                        class="large-input"
                                        :append-inner-icon="showPassword ? 'mdi-eye' :
                                        'mdi-eye-off'"
                                        @click:append-inner="showPassword = !showPassword"
                                    ></v-text-field>
                                </div>

                                <div class="form-group">
                                    <label for="confirmPassword" class="form-label">CONFIRM
                                    PASSWORD</label>
                                    <v-text-field
                                        v-model="confirmPassword"
                                        :rules="rules.confirmPassword"
                                        :type="showConfirmPassword ? 'text' : 'password'"
                                        required
                                        variant="outlined"
                                        density="comfortable"
                                        placeholder="Confirm your password"
                                        hide-details="auto"
                                        class="large-input"
                                        :disabled="!password"
                                        :append-inner-icon="showConfirmPassword ? 'mdi-eye'
                                        : 'mdi-eye-off'"
                                        @click:append-inner="showConfirmPassword =
                                        !showConfirmPassword"
                                    ></v-text-field>
                                </div>

                                <v-btn
                                    @click="registerUser"
                                    :loading="isSubmitting"
                                    :disabled="isSubmitting"
                                    class="btn-brown large-btn"
                                    block
                                >
                                    <span v-if="isSubmitting">Creating Account...</span>
                                    <span v-else>Continue ></span>
                                </v-btn>
                            </v-form>
                        </div>
                    </div>

                    <!-- Right side - Welcome/Login Section -->
                    <div class="welcome-section">
                        <div class="welcome-content">
                            <h1 class="welcome-title">Welcome to Registration</h1>
                            <p class="welcome-text">Already have an account?</p>
                            <v-btn
                                @click="redirectToLogin"
                                class="btn-brown large-btn signup-btn"
                            >
                                Sign In
                            </v-btn>
                        </div>
                    </div>
                </div>
            </v-main>
        </v-app>
    `
}).use(vuetify).mount("#app");
```

f. Use of index.php and index.js in the Project

The homepage is built using a **Vue 3-powered frontend** integrated into a PHP template. This structure allows dynamic rendering of products and categories, with Bootstrap used for styling and layout.

index.php:

This is the **HTML and PHP structure** for the homepage. It includes key components:

1. HTML Head Section

- Loads Bootstrap, Font Awesome, and Google Fonts.
- Links to style.css for custom design.
- Loads Vue 3 and index.js at the bottom of the page.

style.css | index.js | index.php | login.js | login.php

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8" />
5       <title>CozyLoops - Handmade Crochet Creations</title>
6       <meta name="viewport" content="width=device-width, initial-scale=1" />
7
8       <!-- Bootstrap-->
9       <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel
        ="stylesheet">
10
11      <!-- Font Awesome & Fonts -->
12      <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.0/css/all.min.css"
        rel="stylesheet">
13      <link href=
        "https://fonts.googleapis.com/css2?family=Quicksand:wght@500;700&display=swap" rel=
        "stylesheet">
14
15      <!-- Custom Styles -->
16      <link href="style/style.css" rel="stylesheet">
17  </head>
213     <!-- Load Bootstrap JS first -->
214     <script src=
        "https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>
215
216     <!-- Load Vue and custom cart script -->
217     <script src="https://unpkg.com/vue@3/dist/vue.global.prod.js"></script>
218     <script src="index.js"></script>
```

2. Shared Components

- Reuses the navigation bar and footer across the site using modular PHP includes.

3. Vue App Mount Point

- The Vue app from index.js is mounted here and controls everything inside.

```
22  <div id="app">
```

4. Dynamic Sections

- **Hero Section**: Visually engaging banner with a slogan.

```
24      <!-- Hero Section with Grid -->
25      <!-- Hero -->
26      <section class="hero text-center" style="background-image:
        url('images/banner1.jpeg'); background-size: cover;">
27          <h1 class="display-4">Handmade Crochet with Love</h1>
28      </section>
```

- **Category Grid**: Displays 5 categories with emojis and live product counts.

```
<!-- Categories Overview Section -->
<section class="container mt-5">
    <div class="row mb-4">
        <div class="col-12 text-center">
            <h2>Shop by Category</h2>
            <p class="text-muted">Explore our handcrafted collections</p>
        </div>
    </div>

    <!-- Category Grid - Replace your existing category grid section with this -->
    <div class="row g-4 mb-5 justify-content-center">
        <div v-for="category in visibleCategories" :key="category.id" class=
        "col-lg-2 col-md-4 col-sm-6 col-6">
            <div class="card index_category h-100"
                :class="category.id"
                @click="showCategoryInfo(category)"
                tabindex="0"
                @keydown.enter="showCategoryInfo(category)"
                @keydown.space="showCategoryInfo(category)">
                <div class="category_card-body">
                    <div>{{ category.icon }}</div>
                    <h5 class="card-title">{{ category.name }}</h5>
                    <p class="text-muted small">{{ getCategoryCount(category.id) }}
                    items available</p>
                </div>
            </div>
        </div>
    </div>
</section>
```

- **Best Sellers**: Shows the first 3 products dynamically.

```
<!-- Best Sellers Section -->
<section class="container mt-5">
    <div class="row mb-4">
        <div class="col-md-8">
            <h2><i class="fas fa-star text-warning me-2"></i>Best Sellers</h2>
            <p class="text-muted">Our most popular handcrafted items</p>
        </div>
    </div>

    <div class="row g-4 mb-5">
        <div
            v-for="(product, index) in products.slice(0, 3)"
            :key="product.id"
            class="col-lg-4 col-md-6"
        >
            <div class="card product-card h-100">
                <div class="position-relative">
                    <img
                        :src="product.image"
                        :alt="product.name"
                        class="card-img-top product-image"
                    >
                    <span class="badge bg-warning position-absolute top-0 end-0 m-2"
                    >
                        #{{ index + 1 }}
                    </span>
                </div>
                <div class="card-body">
                    <h5 class="card-title">{{ product.name }}</h5>
                    <p class="card-text text-muted small">{{
                    truncateText(product.description, 80) }}</p>
                    <div class="d-flex justify-content-between align-items-center">
                        <span class="price-badge">${{ formatPrice(product.price) }}
                        </span>
                        <small class="text-muted">{{ getProductCategory(product) }}
                        </small>
                    </div>
                </div>
                <div class="card-footer bg-transparent">
                    <button class="btn btn-custom w-100" @click=
                    "openProductModal(product)">
                        <i class="fas fa-eye icon-spacing"></i>View Details
                    </button>
                </div>
            </div>
        </div>
    </div>
</section>
```

- **New Arrivals**: Shows the next 3 products if available.

```
<!-- New Arrivals Section -->
<section class="container mt-5" v-if="products.length > 3">
    <div class="row mb-4">
        <div class="col-md-8">
            <h2><i class="fas fa-sparkles text-info me-2"></i>New Arrivals</h2>
            <p class="text-muted">Fresh additions to our collection</p>
        </div>
    </div>

    <div class="row g-4 mb-5">
        <div
            v-for="product in products.slice(3, 6)"
            :key="product.id"
            class="col-lg-4 col-md-6"
        >
            <div class="card product-card h-100">
                <div class="position-relative">
                    <img
                        :src="product.image"
                        :alt="product.name"
                        class="card-img-top product-image"
                    >
                    <span class="badge bg-success position-absolute top-0 end-0 m-2"
                    >
                        New
                    </span>
                </div>
                <div class="card-body">
                    <h5 class="card-title">{{ product.name }}</h5>
                    <p class="card-text text-muted small">{{
                    truncateText(product.description, 80) }}</p>
                    <div class="d-flex justify-content-between align-items-center">
                        <span class="price-badge">${{ formatPrice(product.price) }}
                        </span>
                        <small class="text-muted">{{ getProductCategory(product) }}
                        </small>
                    </div>
                </div>
                <div class="card-footer bg-transparent">
                    <button class="btn btn-custom w-100" @click=
                    "openProductModal(product)">
                        <i class="fas fa-eye icon-spacing"></i>View Details
                    </button>
                </div>
            </div>
        </div>
    </div>
</section>
```

- **Call to Action**: Contains website description and button to products page.

```
<!-- Call to Action Grid Section -->
<section class="container mt-5">
    <div class="row g-4 mb-5">
        <div class="col-md-8">
            <div class="explore-products-container">
                <h2>Explore All Products</h2>
                <p>Discover our complete collection of handmade crochet items
                crafted with love and attention to detail.</p>
                <a href="products.php" class="btn">
                    <i class="fas fa-th me-2"></i>VIEW ALL PRODUCTS
                </a>
            </div>
        </div>
        <div class="col-md-4">
            <div class="made-with-love-container">
                <div class="heart-icon">♥</div>
                <h3>Made with Love</h3>
                <p>Every item crafted by hand with care and precision</p>
            </div>
        </div>
    </div>
</section>
```

- **Product Modal**: A Bootstrap modal showing product details and an "Add to Cart" button.

```
<!-- Enhanced Product Detail Modal -->
<div class="modal fade" id="productModal" tabindex="-1" aria-labelledby="productModalLabel" aria-hidden="true">
    <div v-if="message" class="alert alert-success position-fixed top-0 end-0 m-3" style="z-index: 9999;">
        {{ message }}
    </div>
    <div class="modal-dialog modal-lg modal-dialog-centered">
        <div class="modal-content custom-card" v-if="selectedProduct">
            <div class="modal-header">
                <h5 class="modal-title">{{ selectedProduct.name }}</h5>
                <button type="button" class="btn-close" data-bs-dismiss="modal"></button>
            </div>
            <div class="modal-body">
                <div class="row">
                    <div class="col-md-6">
                        <img :src="selectedProduct.image" class="img-fluid rounded" :alt="selectedProduct.name">
                    </div>
                    <div class="col-md-6">
                        <h4 class="text-success mb-3">${{ formatPrice(selectedProduct.price) }}</h4>
                        <p>{{ selectedProduct.description }}</p>
                        <hr>
                        <div class="row">
                            <div class="col-6">
                                <p><strong>Category:</strong><br>{{ getProductCategory(selectedProduct) }}</p>
                            </div>
                            <div class="col-6">
                                <p><strong>Status:</strong><br><span class="badge bg-success">In Stock</span></p>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn profile-btn cancel-btn" data-bs-dismiss="modal">Close</button>
                <button type="button" class="btn profile-btn delete-btn" @click="addToCart(selectedProduct)">
                    <i class="fas fa-cart-plus icon-spacing"></i>Add to Cart
                </button>
            </div>
        </div>
    </div>
</div>
```

index.js:

This file contains the **Vue app logic** responsible for dynamic rendering, product filtering, modals, and cart interaction.

1. Fetching Product Data

- Loads product data from products.json when the page is mounted.
- Stores it in the products array for use in rendering sections.

```
async fetchProducts() {
    try {
        const response = await fetch('products.json');
        if (!response.ok) {
            throw new Error(`HTTP error! status: ${response.status}`);
        }
        const data = await response.json();
        this.products = data;
        console.log('Successfully loaded products from products.json:', this.
        products.length);
    } catch (error) {
        this.error = 'Error loading products from products.json: ' + error.message;
        console.error('Error loading products:', error);
        console.error('Make sure products.json exists and is properly formatted');
    } finally {
        this.loading = false;
    }
},
```

2. Category Filtering

- Limits category grid display to 5 main categories (excluding "All").
- showCategoryInfo(category) navigates to products.php?category=....

```
data() {
    return {
        products: [],
        selectedProduct: null,
        loading: true,
        error: null,
        message: '',
        isSuccess: false,
        isError: false,
        modalInstance: null,

        // Category definitions for product categorization
        categories: [
            { id: 'all', name: 'All Products', icon: '' },
            { id: 'accessories', name: 'Accessories', icon: '' },
            { id: 'clothing', name: 'Clothing', icon: '' },
            { id: 'home', name: 'Home Decor', icon: '' },
            { id: 'baby', name: 'Baby Items', icon: '' },
            { id: 'seasonal', name: 'Seasonal', icon: '' }
        ]
    };
},

computed: {
    // Show only 5 categories for the grid display
    visibleCategories() {
        return this.categories.slice(1); // Skip 'all' and show 5 categories
    },

    // Group products by category for better organization
    productsByCategory() {
        const grouped = {};
        this.categories.forEach(cat => {
            grouped[cat.id] = this.products.filter(product =>
                this.getProductCategoryId(product) === cat.id
            );
        });
        return grouped;
    },
```

```
144         // UPDATED: Navigate to products page with category filter
145         showCategoryInfo(category) {
146             // Navigate to products page with the selected category filter
147             window.location.href = `products.php?category=${category.id}`;
148         },
149
150         // Get count of products in each category
151         getCategoryCount(categoryId) {
152             if (categoryId === 'all') {
153                 return this.products.length;
154             }
155             return this.products.filter(product =>
156                 this.getProductCategoryId(product) === categoryId
157             ).length;
158         },
```

3. Product Grouping

- Dynamically divides products into **featured** and **new arrivals** sections using .slice().

```
43
44          // Get featured products (first 3)
45          featuredProducts() {
46              return this.products.slice(0, 3);
47          },
48
49          // Get new arrivals (next 3 products)
50          newArrivals() {
51              return this.products.slice(3, 6);
52          }
53      },
```

4. Modal Management

- Opens a Bootstrap modal (#productModal) and fills it with the selected product's data.
- On modal close, resets state (selectedProduct, message, etc.).

```
55  methods: {
56      // Enhanced modal opening method
57      openProductModal(product) {
58          this.selectedProduct = product;
59          // Initialize Bootstrap modal if not already done
60          if (!this.modalInstance) {
61              const modalElement = document.getElementById('productModal');
62              this.modalInstance = new bootstrap.Modal(modalElement);
63
64              // Add event listener for modal close
65              modalElement.addEventListener('hidden.bs.modal', () => {
66                  this.selectedProduct = null;
67                  this.message = '';
68                  this.isSuccess = false;
69                  this.isError = false;
70              });
71          }
72          this.modalInstance.show();
73      },
74
75      // Keep your existing openModal method for compatibility
76      openModal(product) {
77          this.openProductModal(product);
78      },
```

5. Add to Cart Functionality

- Sends the selected product's details to cart_api.php using a POST request.
- Displays success or error messages using showMessage().

```
addToCart(product) {
    fetch('cart_api.php', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            id: product.id,
            name: product.name,
            price: product.price,
            image: product.image
        })
    })
    .then(response => response.json())
    .then(data => {
        if (data.success) {
            this.showMessage(data.message || 'Added to cart!', true);
        } else {
            this.showMessage(data.message || 'Failed to add to cart.', false);
        }
    })
    .catch(error => {
        console.error('Error adding to cart:', error);
        this.showMessage('Error adding to cart.', false);
    });
},

showMessage(text) {
    this.message = text;
    setTimeout(() => {
        this.message = '';
    }, 2000);
},
```

6. Product Categorization Logic

- Analyses keywords in product name and description to assign a category:
    - "booties" → Baby
    - "pillow" → Home Decor
    - "stocking" → Seasonal
    - "scarf" → Clothing
    - fallback → Accessories

This categorization supports filtering and display grouping.

```
// Enhanced product categorization helper methods
getProductCategoryId(product) {
    const name = product.name.toLowerCase();
    const desc = product.description.toLowerCase();

    // Baby Items
    if (name.includes('baby') || name.includes('booties') || desc.includes('nursery
    decor') || desc.includes('baby's feet')) {
        return 'baby';
    }
    // Seasonal Items (e.g., Christmas)
    else if (name.includes('christmas') || name.includes('holiday') || name.includes
    ('stocking')) {
        return 'seasonal';
    }
```

```
            // Home Decor
            else if (name.includes('pillow') || name.includes('plant') ||
                     name.includes('pot cover') || desc.includes('home decor')) {
                return 'home';
            }
            // Clothing
            else if (name.includes('hat') || name.includes('scarf') || name.includes(
            'headband') ||
                     name.includes('sweater')) { // Removed 'cozy' from clothing
                     for now
                return 'clothing';
            }
            // Accessories (This is the default/fallback if no other category matches)
            // Including "Soft Crochet Blanket" and "Crochet Coffee Cup Cozy" here
            else {
                return 'accessories';
            }
        },

        getProductCategory(product) {
            const categoryId = this.getProductCategoryId(product);
            const category = this.categories.find(cat => cat.id === categoryId);
            return category ? category.name : 'Accessories';
        },

        // Enhanced filtering method for future use
        filterProductsByCategory(categoryId) {
            if (categoryId === 'all') {
                return this.products;
            }
            return this.products.filter(product =>
                this.getProductCategoryId(product) === categoryId
            );
        },
```

7. Utility Methods

- formatPrice(price) → Ensures price is always shown with 2 decimals.

```
        // Format helpers
        formatPrice(price) {
            return parseFloat(price).toFixed(2);
        },
```

- truncateText(text, length) → Trims long descriptions.

```
        truncateText(text, maxLength) {
            if (!text) return '';
            if (text.length <= maxLength) return text;
            return text.substring(0, maxLength).trim() + '...';
        },
```

- getCategoryStats() → Builds a summary of product counts per category (for debugging or future UI use).

```
        // Get category statistics for display
        getCategoryStats() {
            const stats = {};
            this.categories.forEach(category => {
                if (category.id !== 'all') {
                    stats[category.id] = {
                        name: category.name,
                        count: this.getCategoryCount(category.id),
                        icon: category.icon
                    };
                }
            });
            return stats;
        }
    },
```

Workflow Overview:

[index.php] - HTML structure & Bootstrap modal

↓

[index.js] - Vue 3 logic for rendering, modal, filtering

↓

[products.json] - Supplies product data

↓

[cart_api.php] - Handles cart insertion via fetch() POST

How It All Comes Together

- **User opens the homepage (index.php)**
- **Vue app mounts on #app and fetches products.json**
- Products are grouped into **categories**, **best sellers**, and **new arrivals**
- When a user clicks **"View Details"**, the modal shows product info
- Clicking **"Add to Cart"** triggers a POST to cart_api.php via fetch in index.js

g. Use of products.php, products.js and products.json in the Project

This product catalog system uses:

- products.php – HTML and Vue mount point
- products.js – Vue logic for filtering, searching, sorting, and pagination
- products.json – A static data source listing all products

These files collectively power a dynamic, filterable, and user-friendly product browsing interface.

products.php:
This file defines the **page structure and layout**, using HTML, Bootstrap, and Vue directives.

1. HTML Head Section

- Loads Bootstrap, Font Awesome, and Google Fonts.
- Links to style.css for custom design.
- Loads Vue 3 and products.js at the bottom of the page.

2. Shared Components

- Reuses the navigation bar and footer across the site using modular PHP includes.

3. Vue App Mount Point

- The Vue app from products.js is mounted here and controls everything inside.

```
22      <div id="app" class="container my-5">
```

4. Category Tabs

- Renders product categories dynamically using a v-for loop.
- Clicking a tab update the selectedCategory in Vue and filters products.

```
27      <!-- Category Filter Tabs (Demonstrating Arrays & Selection Directives) -->
28      <div class="category-tabs">
29          <h5 class="mb-3 text-center">
30              <i class="fas fa-tags icon-spacing"></i>Browse by Category
31          </h5>
32          <div class="text-center">
33              <button
34                  v-for="category in categories"
35                  :key="category.id"
36                  :class="['btn', 'category-btn', { active: selectedCategory ===
                    category.id }]"
37                  @click="selectCategory(category.id)"
38              >
39                  {{ category.icon }} {{ category.name }}
40                  <span class="badge bg-light text-dark ms-1">{{ category.count }}</span>
41              </button>
42          </div>
43      </div>
```

5. Advanced Filters

- Includes (Loading state and no product message included):
  - **Search bar** (text input for live filtering).

```
<!-- Advanced Filters Section -->
<div class="filter-section">
    <div class="row align-items-end">
        <div class="col-md-4">
            <label class="form-label">
                <i class="fas fa-search icon-spacing"></i>Search Products
            </label>
            <div class="position-relative">
                <input
                    type="text"
                    class="form-control"
                    v-model.trim="searchQuery"
                    placeholder="Search names, descriptions..."
                    @input="resetPagination"
                >
```

```
                <!-- Clear button (X) positioned on the right side -->
                <button
                    v-show="searchQuery.trim() !== ''"
                    @click="searchQuery = ''; resetPagination()"
                    class="btn position-absolute"
                    style="right: 8px; top: 50%; transform: translateY(-50%);
                    border: none; background: transparent; color: #6c757d; padding:
                    0; width: 24px; height: 24px; display: flex; align-items:
                    center; justify-content: center; border-radius: 50%;
                    transition: all 0.2s ease;"
                    onmouseover="this.style.backgroundColor='#f8f9fa';
                    this.style.color='#495057';"
                    onmouseout="this.style.backgroundColor='transparent';
                    this.style.color='#6c757d';"
                    type="button"
                    title="Clear search"
                >
                    <i class="fas fa-times" style="font-size: 14px;"></i>
                </button>
            </div>
        </div>
```

  - **Price range** (dropdown).

```
            <div class="col-md-2">
                <label class="form-label">
                    <i class="fas fa-dollar-sign icon-spacing"></i>Price Range
                </label>
                <select class="form-select" v-model="priceRange" @change=
                "resetPagination">
                    <option v-for="range in priceRanges" :key="range.value" :value=
                    "range.value">
                        {{ range.label }}
                    </option>
                </select>
            </div>
```

  - **Sort by** (e.g., price ascending/descending).

```
            <div class="col-md-2">
                <label class="form-label">
                    <i class="fas fa-sort icon-spacing"></i>Sort By
                </label>
                <select class="form-select" v-model="sortBy">
                    <option v-for="option in sortOptions" :key="option.value" :value=
                    "option.value">
                        {{ option.label }}
                    </option>
                </select>
            </div>
```

- **Items per page.**

```
95          <div class="col-md-2">
96              <label class="form-label">Items Per Page</label>
97              <select class="form-select" v-model.number="itemsPerPage" @change=
                "resetPagination">
98                  <option v-for="size in pageSizes" :key="size" :value="size">{{ size
                    }}</option>
99              </select>
100         </div>
```

- **Active filters display, Pagination info summary** and **No products message.**

```
120         <!-- Active Filters Display -->
121         <div v-show="hasActiveFilters" class="mt-3 pt-3 border-top">
122             <small class="text-muted me-2">Active Filters:</small>
123             <span v-if="searchQuery" class="badge bg-primary me-2">
124                 Search: "{{ searchQuery }}" <i class="fas fa-times ms-1" @click=
                    "searchQuery = ''" style="cursor: pointer;"></i>
125             </span>
126             <span v-if="selectedCategory !== 'all'" class="badge bg-success me-2">
127                 Category: {{ getCategoryName(selectedCategory) }} <i class="fas
                    fa-times ms-1" @click="selectedCategory = 'all'" style="cursor:
                    pointer;"></i>
128             </span>
129             <span v-if="priceRange !== 'all'" class="badge bg-warning me-2">
130                 Price: {{ getPriceRangeLabel(priceRange) }} <i class="fas fa-times ms-1"
                     @click="priceRange = 'all'" style="cursor: pointer;"></i>
131             </span>
132             <button class="btn btn-sm btn-outline-danger" @click="clearAllFilters">
133                 <i class="fas fa-refresh icon-spacing"></i>Clear All
134             </button>
135         </div>
136     </div>
138     <!-- Results Summary with Pagination Info -->
139     <div class="d-flex justify-content-between align-items-center mb-4" v-if=
        "filteredProducts.length > 0">
140         <span class="info-bubble me-3">
141             <strong>{{ filteredProducts.length }}</strong> product<span v-if=
                "filteredProducts.length !== 1">s</span> found
142             <span v-if="hasActiveFilters" class="text-muted ms-2">
143                 (filtered from {{ products.length }} total)
144             </span>
145         </span>
146         <span class="info-bubble">
147             <span v-if="totalPages > 1">Page {{ currentPage }} of {{ totalPages }}
                </span>
148             ({{ startIndex }}-{{ endIndex }} of {{ filteredProducts.length }})
149         </span>
150     </div>
151
152     <!-- No Products Message (Conditional Rendering Directive) -->
153     <div v-if="filteredProducts.length === 0 && !loading" class="no-products">
154         <i class="fas fa-search fa-4x text-muted mb-4"></i>
155         <h3>No Products Found</h3>
156         <p class="text-muted mb-4">
157             <span v-if="hasActiveFilters">
158                 Try adjusting your filters or search terms
159             </span>
160             <span v-else>
161                 No products are currently available
162             </span>
163         </p>
164         <button v-if="hasActiveFilters" class="btn btn-brown" @click="clearAllFilters">
165             <i class="fas fa-refresh icon-spacing"></i>Show All Products
166         </button>
167     </div>
168
169     <!-- Loading State -->
170     <div v-if="loading" class="text-center py-5">
171         <div class="spinner-border text-primary" role="status" style="width: 3rem;
            height: 3rem;">
172             <span class="visually-hidden">Loading...</span>
173         </div>
174         <p class="mt-3 text-muted">Loading products...</p>
175     </div>
```

- **View toggle** (grid or list layout).

```
                    <div class="col-md-2">
                        <label class="form-label">View Mode</label>
                        <div class="view-mode-toggle d-flex">
                            <button
                                :class="['view-btn', 'flex-fill', { active: viewMode === 'grid' }]"
                                @click="viewMode = 'grid'"
                            >
                                <i class="fas fa-th"></i>
                            </button>
                            <button
                                :class="['view-btn', 'flex-fill', { active: viewMode === 'list' }]"
                                @click="viewMode = 'list'"
                            >
                                <i class="fas fa-list"></i>
                            </button>
                        </div>
                    </div>
                </div>
            <!-- Products Display (Repetition Directives with Dynamic Layout) -->
            <div v-if="!loading && filteredProducts.length > 0">
                <!-- Grid View -->
                <div v-if="viewMode === 'grid'" class="row g-4">
                    <div
                        v-for="(product, index) in paginatedProducts"
                        :key="product.id"
                        :class="gridColumnClass"
                    >
                        <div class="card product-card h-100">
                            <img
                                :src="product.image"
                                :alt="product.name"
                                class="card-img-top product-image"
                            >
                            <div class="card-body">
                                <h5 class="card-title">{{ product.name }}</h5>
                                <p class="card-text text-muted small">{{ truncateText(product.description, 80) }}</p>
                                <div class="d-flex justify-content-between align-items-center">
                                    <span class="price-badge">${{ formatPrice(product.price) }}</span>
                                    <small class="text-muted">{{ getProductCategory(product) }}</small>
                                </div>
                            </div>
                            <div class="card-footer bg-transparent">
                                <button class="btn btn-custom w-100" @click="openProductModal(product)">
                                    <i class="fas fa-eye icon-spacing"></i>View Details
                                </button>
                            </div>
                        </div>
                    </div>
                </div>
```

```
<!-- List View -->
<div v-else class="list-view">
    <div
        v-for="(product, index) in paginatedProducts"
        :key="product.id"
        class="card product-card mb-3"
    >
        <div class="row g-0">
            <div class="col-md-3">
                <img
                    :src="product.image"
                    :alt="product.name"
                    class="img-fluid h-100"
                    style="object-fit: cover; border-radius: 15px 0 0 15px;"
                >
            </div>
            <div class="col-md-9">
                <div class="card-body d-flex justify-content-between">
                    <div>
                        <h5 class="card-title">{{ product.name }}</h5>
                        <p class="card-text">{{ product.description }}</p>
                        <small class="text-muted">Category: {{
                        getProductCategory(product) }}</small>
                    </div>
                    <div class="text-end">
                        <div class="price-badge mb-2">${{
                        formatPrice(product.price) }}</div>
                        <button class="btn btn-custom" @click=
                        "openProductModal(product)">
                            <i class="fas fa-eye icon-spacing"></i>View
                        </button>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

- These controls are reactive and modify the product list in real time.

6. Pagination

- Uses the vuejs-paginate-next library to paginate results.

```
<!-- Enhanced Pagination Controls -->
<div v-if="totalPages > 1" class="mt-5">
    <div class="d-flex justify-content-center align-items-center">

        <paginate
            v-model="currentPage"
            :page-count="totalPages"
            :click-handler="changePage"
            :prev-text="'Prev'"
            :next-text="'Next'"
            :container-class="'pagination justify-content-center mb-0'"
            :page-class="'page-item'"
            :page-link-class="'page-link btn-custom'"
            :prev-class="'page-item'"
            :prev-link-class="'page-link btn-custom'"
            :next-class="'page-item'"
            :next-link-class="'page-link btn-custom'"
            :active-class="'active'"
            :disabled-class="'disabled'"
            :page-range="5"
        />

        <button
            class="btn btn-outline-primary ms-2"
            :disabled="currentPage === totalPages"
            @click="changePage(totalPages)"
        >
            <i class="fas fa-angle-double-right"></i>
        </button>
    </div>
```

- Supports jump-to-page, prev/next navigation, and input for direct jumps.

```
<!-- Jump to Page -->
<div class="text-center mt-3">
    <small class="text-muted me-2">Jump to page:</small>
    <input
        type="number"
        class="form-control d-inline-block"
        style="width: 80px;"
        :min="1"
        :max="totalPages"
        v-model.number="jumpToPage"
        @keyup.enter="goToPage"
    >
    <button class="btn-brown ms-2" @click="goToPage">Go</button>
</div>
</div>
```

7. Product Modal

- Dynamically shows full details of the selected product.
- Includes an “Add to Cart” button (which connects to cart_api.php).

```
<!-- Enhanced Product Detail Modal -->
<div class="modal fade" id="productModal" tabindex="-1" aria-labelledby="productModalLabel" aria-hidden="true">
    <div v-if="message" class="alert alert-success position-fixed top-0 end-0 m-3" style="z-index: 9999;">
        {{ message }}
    </div>
    <div class="modal-dialog modal-lg modal-dialog-centered">
        <div class="modal-content custom-card" v-if="selectedProduct">
            <div class="modal-header">
                <h5 class="modal-title">{{ selectedProduct.name }}</h5>
                <button type="button" class="btn-close" data-bs-dismiss="modal"></button>
            </div>
            <div class="modal-body">
                <div class="row">
                    <div class="col-md-6">
                        <img :src="selectedProduct.image" class="img-fluid rounded" :alt="selectedProduct.name">
                    </div>
                    <div class="col-md-6">
                        <h4 class="text-success mb-3">${{ formatPrice(selectedProduct.price) }}</h4>
                        <p>{{ selectedProduct.description }}</p>
                        <hr>
                        <div class="row">
                            <div class="col-6">
                                <p><strong>Category:</strong><br>{{ getProductCategory(selectedProduct) }}</p>
                            </div>
                            <div class="col-6">
                                <p><strong>Status:</strong><br><span class="badge bg-success">In Stock</span></p>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn profile-btn cancel-btn" data-bs-dismiss="modal">Close</button>
                <button type="button" class="btn profile-btn delete-btn" @click="addToCart(selectedProduct)">
                    <i class="fas fa-cart-plus icon-spacing"></i>Add to Cart
                </button>
            </div>
        </div>
    </div>
</div>
</div>
```

products.js:

This file contains the Vue 3 logic that powers the products.php product catalog. It supports category filtering, advanced search, pagination, sorting, responsive layout switching, and integration with the cart system.

1. Data Initialization

- Holds all reactive data used for filtering, pagination, sorting, and view mode.

style.css products.js

```
7     data() {
8         return {
9             // Core product data (ARRAY DEMONSTRATION)
10            products: [],
11            selectedProduct: null,
12            loading: false,
13            message: '',
14            modalInstance: null,
15
16            // Filter and search arrays (ARRAY DEMONSTRATION)
17            categories: [
18                { id: 'all', name: 'All Products', icon: '', count: 0 },
19                { id: 'accessories', name: 'Accessories', icon: '', count: 0 },
20                { id: 'clothing', name: 'Clothing', icon: '', count: 0 },
21                { id: 'home', name: 'Home Decor', icon: '', count: 0 },
22                { id: 'baby', name: 'Baby Items', icon: '', count: 0 },
23                { id: 'seasonal', name: 'Seasonal', icon: '', count: 0 }
24            ],
25
26            priceRanges: [
27                { value: 'all', label: 'All Prices' },
28                { value: '0-10', label: 'Under $10' },
29                { value: '10-20', label: '$10 - $20' },
30                { value: '20-30', label: '$20 - $30' },
31                { value: '30-50', label: '$30 - $50' },
32                { value: '50+', label: '$50+' }
33            ],
34
35            sortOptions: [
36                { value: 'name_asc', label: 'Name (A-Z)' },
37                { value: 'name_desc', label: 'Name (Z-A)' },
38                { value: 'price_low', label: 'Price: Low to High' },
39                { value: 'price_high', label: 'Price: High to Low' },
40                { value: 'id_asc', label: 'Newest First' },
41                { value: 'id_desc', label: 'Oldest First' }
42            ],
43
44            pageSizes: [6, 9, 12, 18, 24],
45
46            // Filter states (SELECTION DIRECTIVES) - Initialize from URL or defaults
47            selectedCategory: this.getInitialUrlParam('category', 'all'),
48            searchQuery: this.getInitialUrlParam('q', ''),
49            priceRange: this.getInitialUrlParam('price', 'all'),
50            sortBy: this.getInitialUrlParam('sort', 'name_asc'),
51            viewMode: this.getInitialUrlParam('view', 'grid'),
52
53            // Pagination states (PAGINATION DEMONSTRATION) - Initialize from URL or
              defaults
54            currentPage: parseInt(this.getInitialUrlParam('page', 1)),
55            itemsPerPage: parseInt(this.getInitialUrlParam('limit', 9)),
56            jumpToPage: parseInt(this.getInitialUrlParam('page', 1)) // Keep jumpToPage in
              sync
57        };
58    },
```

2. Fetching Product Data

- Loads product data from the static products.json file once the app is mounted.

```
439         // DATA LOADING METHODS
440         async loadProducts() {
441             this.loading = true;
442             try {
443                 // Load products from products.json
444                 const response = await fetch('products.json');
445                 if (!response.ok) {
446                     // Corrected template literal syntax
447                     throw new Error(`HTTP error! status: ${response.status}`);
448                 }
449                 const data = await response.json();
450                 this.products = data;
451                 this.updateCategoryCounts();
452             } catch (error) {
453                 console.error('Error loading products:', error);
454                 this.showMessage('Error loading products. Please check if products.json
                    exists.');
455                 // Fallback to empty array
456                 this.products = [];
457             } finally {
458                 this.loading = false;
459             }
460         }
461     },
```

3. Filtering Logic

- By Category
  - Uses keywords in the product's name or description to assign a category (e.g., "baby", "clothing", "home").
  - Each product is assigned a category dynamically.
  - Filters update filteredProducts, which powers the visible list.

```
268         // FILTER HELPER METHODS (FORMAT & FILTER DEMONSTRATIONS)
269         getProductCategoryId(product) {
270             const name = product.name.toLowerCase();
271             const desc = product.description.toLowerCase();
272
273             // Baby Items
274             if (name.includes('baby') || name.includes('booties') || desc.includes('nursery
                decor') || desc.includes('baby's feet')) {
275                 return 'baby';
276             }
277             // Seasonal Items (e.g., Christmas)
278             else if (name.includes('christmas') || name.includes('holiday') || name.includes
                ('stocking')) {
279                 return 'seasonal';
280             }
281             // Home Decor
282             else if (name.includes('pillow') || name.includes('plant') ||
283                      name.includes('pot cover') || desc.includes('home decor')) {
284                 return 'home';
285             }
286             // Clothing
287             else if (name.includes('hat') || name.includes('scarf') || name.includes(
                'headband') ||
288                      name.includes('sweater')) { // Removed 'cozy' from clothing
                        for now
289                 return 'clothing';
290             }
291             // Accessories (This is the default/fallback if no other category matches)
292             // Including "Soft Crochet Blanket" and "Crochet Coffee Cup Cozy" here
293             else {
294                 return 'accessories';
295             }
296         },
```

- By Search, Sort and Price
    - Allow users to filter products with multi-condition logic.
    - Includes search (v-model="searchQuery"), price range, and dropdown sorting (e.g., price, name, newest)

```
computed: {
    // FILTER DEMONSTRATION - Complex filtering with multiple conditions
    filteredProducts() {
        let filtered = [...this.products];

        // Category filter
        if (this.selectedCategory !== 'all') {
            filtered = filtered.filter(product => {
                return this.getProductCategoryId(product) === this.selectedCategory;
            });
        }

        // Search filter (name and description)
        if (this.searchQuery.trim()) {
            const query = this.searchQuery.toLowerCase().trim();
            filtered = filtered.filter(product =>
                product.name.toLowerCase().includes(query) ||
                product.description.toLowerCase().includes(query)
            );
        }

        // Price range filter
        if (this.priceRange !== 'all') {
            filtered = filtered.filter(product => {
                const price = product.price;
                switch (this.priceRange) {
                    case '0-10': return price < 10;
                    case '10-20': return price >= 10 && price < 20;
                    case '20-30': return price >= 20 && price < 30;
                    case '30-50': return price >= 30 && price < 50;
                    case '50+': return price >= 50;
                    default: return true;
                }
            });
        }

        // Sorting
        return filtered.sort((a, b) => {
            switch (this.sortBy) {
                case 'name_desc':
                    return b.name.localeCompare(a.name);
                case 'price_low':
                    return a.price - b.price;
                case 'price_high':
                    return b.price - a.price;
                case 'id_desc': // Newest First - higher ID is newer
                    return b.id - a.id;
                case 'id_asc': // Oldest First - lower ID is older
                    return a.id - b.id;
                default: // name_asc
                    return a.name.localeCompare(b.name);
            }
        });
    },
```

4. Pagination Logic

- Display a limited set of products per page.
- Controlled by VuePaginate component.
- Supports jump-to-page, items per page, and total product range display.

```
115         // PAGINATION COMPUTATIONS
116         totalProductsCount() { // New computed property for total count of filtered products
117             return this.filteredProducts.length;
118         },
119
120         totalPages() {
121             return Math.ceil(this.totalProductsCount / this.itemsPerPage);
122         },
123
124         paginatedProducts() {
125             const start = (this.currentPage - 1) * this.itemsPerPage;
126             const end = start + this.itemsPerPage;
127             return this.filteredProducts.slice(start, end);
128         },
129
130         startIndex() {
131             return (this.currentPage - 1) * this.itemsPerPage + 1;
132         },
133
134         endIndex() {
135             return Math.min(this.currentPage * this.itemsPerPage, this.filteredProducts.
                length);
136         },
```

5. Grid Sizing based on Items Per Page

- The products grid is dynamically arranged and resized based on the number of items shown per page using bootstrap.

```
161         gridColumnClass() {
162             // Dynamic grid sizing based on items per page
163             switch (this.itemsPerPage) {
164                 case 6: return 'col-lg-4 col-md-6';
165                 case 9: return 'col-lg-4 col-md-6';
166                 case 12: return 'col-xl-3 col-lg-4 col-md-6';
167                 case 18: return 'col-xl-3 col-lg-4 col-md-6 col-sm-6';
168                 case 24: return 'col-xl-2 col-lg-3 col-md-4 col-sm-6';
169                 default: return 'col-lg-4 col-md-6';
170             }
171         }
172     },
```

6. Modal Management

- Show selected product in a Bootstrap modal.
- The modal displays product image, description, category, and price.
- Resets when closed.

```
375         // MODAL METHODS
376         openProductModal(product) {
377             this.selectedProduct = product;
378             // Initialize Bootstrap modal if not already done
379             if (!this.modalInstance) {
380                 const modalElement = document.getElementById('productModal');
381                 // Ensure Bootstrap is loaded before trying to instantiate Modal
382                 if (typeof bootstrap !== 'undefined' && bootstrap.Modal) {
383                     this.modalInstance = new bootstrap.Modal(modalElement);
384                 } else {
```

```
385                 console.warn('Bootstrap Modal not found. Make sure Bootstrap JS is
                    loaded.');
386                 // Fallback if Bootstrap isn't loaded, e.g., show a simple alert
387                 // Corrected template literal syntax
388                 alert(`Product: ${product.name}\nPrice: $${product.price}\nDescription:
                    ${product.description}`);
389                 return;
390             }
391         }
392         this.modalInstance.show();
393     },
394
395     closeProductModal() {
396         if (this.modalInstance) {
397             this.modalInstance.hide();
398         }
399         this.selectedProduct = null;
400     },
```

7. Add to Cart Functionality

- Send product info to the backend for cart storage.
- Sends a POST request to cart_api.php.
- Triggers a success message after adding.

```
402     addToCart(product) {
403         fetch('cart_api.php', {
404             method: 'POST',
405             headers: {
406                 'Content-Type': 'application/json'
407             },
408             body: JSON.stringify({
409                 id: product.id,
410                 name: product.name,
411                 price: product.price,
412                 image: product.image
413             })
414         })
415         .then(response => response.json())
416         .then(data => {
417             if (data.success) {
418                 this.showMessage(data.message || 'Added to cart!', true);
419             } else {
420                 this.showMessage(data.message || 'Failed to add to cart.', false);
421             }
422         })
423         .catch(error => {
424             console.error('Error adding to cart:', error);
425             this.showMessage('Error adding to cart.', false);
426         });
427     },
428
429     showMessage(text, isSuccess = true) { // Added isSuccess parameter for styling
430         this.message = text;
431         // You'll need to pass 'isSuccess' / 'isError' state to the template for styling
432         // For simplicity, using one 'message' here. If you have different styles,
433         // you might need additional ref properties like isMessageSuccess,
            isMessageError.
434         setTimeout(() => {
435             this.message = '';
436         }, 1000);
437     },
```

8. Utility and UI Helpers

- formatPrice(price) – Converts price to 2 decimal points.

```
314         // FORMAT FILTERS
315         formatPrice(price) {
316             return parseFloat(price).toFixed(2);
317         },
```

- truncateText(text, maxLength) – Limits long descriptions.

```
319         truncateText(text, maxLength) {
320             if (text.length <= maxLength) return text;
321             return text.substring(0, maxLength).trim() + '...';
322         },
```

- updateUrl() – Syncs filters to URL query parameters.

```
205 methods: {
206     // Helper to get initial URL parameter
207     getInitialUrlParam(paramName, defaultValue) {
208             const urlParams = new URLSearchParams(window.location.search);
209             return urlParams.get(paramName) || defaultValue;
210     },
211
212     // Method to update URL query parameters
213     updateUrl() {
214             const urlParams = new URLSearchParams();
215
216             if (this.selectedCategory !== 'all') {
217                 urlParams.set('category', this.selectedCategory);
218             }
219             if (this.searchQuery.trim() !== '') {
220                 urlParams.set('q', this.searchQuery.trim());
221             }
222             if (this.priceRange !== 'all') {
223                 urlParams.set('price', this.priceRange);
224             }
225             if (this.sortBy !== 'name_asc') { // Only add if not default
226                 urlParams.set('sort', this.sortBy);
227             }
228             if (this.currentPage !== 1) { // Only add if not default
229                 urlParams.set('page', this.currentPage);
230             }
231             if (this.itemsPerPage !== 9) { // Only add if not default
232                 urlParams.set('limit', this.itemsPerPage);
233             }
234             if (this.viewMode !== 'grid') { // Only add if not default
235                 urlParams.set('view', this.viewMode);
236             }
237
238             // Corrected template literal syntax
239             const newUrl = `${window.location.pathname}?${urlParams.toString()}`;
240             // Use replaceState to avoid cluttering history with every key press for
                search,
241             // but pushState is also an option if you want every filter change in
                history.
242             // For search, replaceState is generally better for UX.
243             window.history.replaceState({}, '', newUrl);
244     },
```

- clearAllFilters() – Resets all filters and pagination.

```
365         // FILTER MANAGEMENT
366         clearAllFilters() {
367             this.selectedCategory = 'all';
368             this.searchQuery = '';
369             this.priceRange = 'all';
370             this.sortBy = 'name_asc';
371             this.itemsPerPage = 9; // Reset items per page too
372             this.resetPagination(); // This will trigger updateUrl via watcher
373         },
```

9. State Watchers

- Reactively track user input and maintain sync.
- Triggers updateUrl() to keep the URL in sync with UI state.

```
    watch: {
        // Watch for changes in filter and pagination states to update URL
        selectedCategory: 'updateUrl',
        searchQuery: 'updateUrl',
        priceRange: 'updateUrl',
        sortBy: 'updateUrl',
        currentPage: 'updateUrl',
        itemsPerPage: 'updateUrl',
        viewMode: 'updateUrl',
```

```
        // Auto-reset pagination when filters change (still necessary)
        filteredProducts() {
            // Only reset if the current page is out of bounds for the new filtered results
            if (this.currentPage > this.totalPages && this.totalPages > 0) {
                this.currentPage = 1;
                this.jumpToPage = 1;
            } else if (this.totalPages === 0) { // If no products match filters
                    this.currentPage = 1;
                    this.jumpToPage = 1;
            }
        },

        // Update category counts when products change
        products: {
            handler() {
                this.updateCategoryCounts();
            },
            deep: true
        }
    },
```

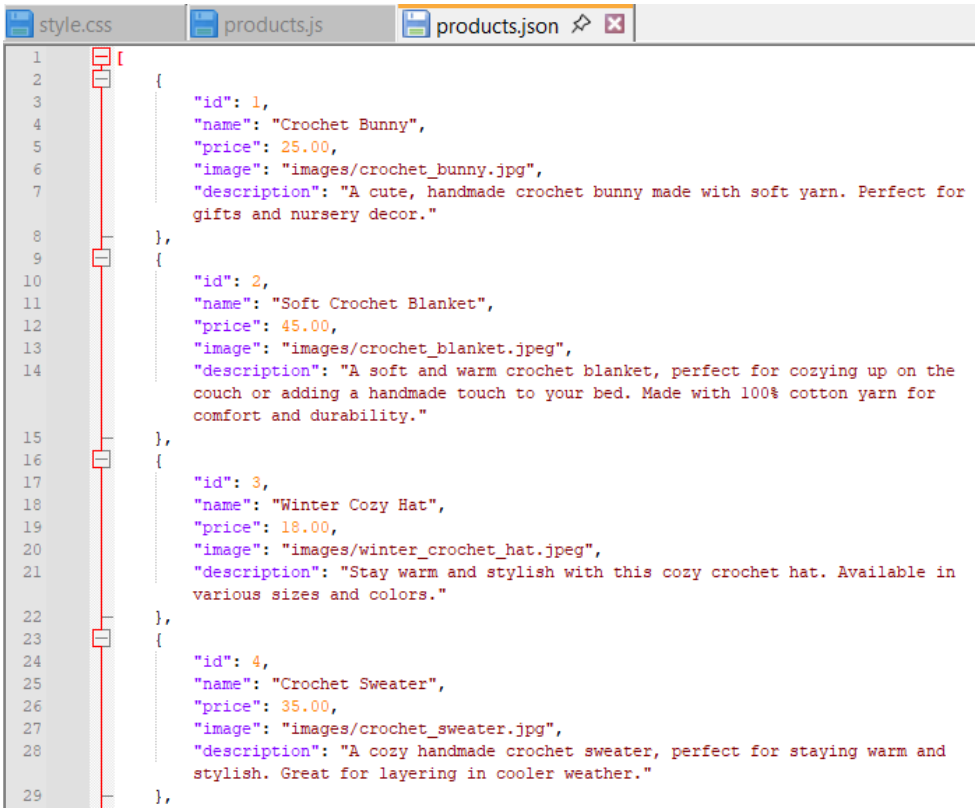
products.json:

This file contains an array of product objects:

Each object includes:

- id: unique identifier
- name, price, image, description: used for display, filtering, and categorization

This data is loaded directly into the Vue app for rendering and interaction.



```
    {
        "id": 6,
        "name": "Crochet Coffee Cup Cozy",
        "price": 8.00,
        "image": "images/crochet_coffee_cup_cozy.jpg",
        "description": "Keep your hands cool with this eco-friendly crochet coffee cup cozy. Fits most standard cup sizes."
    },
    {
        "id": 7,
        "name": "Crochet Baby Booties",
        "price": 15.00,
        "image": "images/crochet_baby_booties.jpg",
        "description": "Cute and comfy crochet baby booties, perfect for your little one. Soft, stretchy, and gentle on baby's feet."
    },
    {
        "id": 8,
        "name": "Crochet Keychain",
        "price": 6.00,
        "image": "images/crochet_keychain.jpg",
        "description": "Handmade crochet keychain. Add a touch of personality to your keys, bags, or as a gift."
    },
    {
        "id": 9,
        "name": "Crochet Plant Pot Cover",
        "price": 18.00,
        "image": "images/crochet_plant_pot_cover.jpg",
        "description": "A stylish crochet cover for your plant pots. Adds charm and protection to your plants while decorating your space."
    },
    {
        "id": 10,
        "name": "Crochet Handbag",
        "price": 22.00,
        "image": "images/crochet_handbag.jpg",
        "description": "Stylish and eco-friendly crochet handbag, perfect for carrying your essentials with a touch of handmade charm."
    },
    {
        "id": 11,
        "name": "Crochet Scarf",
        "price": 20.00,
        "image": "images/crochet_scarf.jpg",
        "description": "Warm and cozy crochet scarf, perfect for colder weather. (Wooden house shown in image not included.)"
    },
    {
        "id": 12,
        "name": "Crochet Tote Bag",
        "price": 28.00,
        "image": "images/crochet_tote_bag.jpg",
        "description": "Handmade crochet tote bag. A stylish and functional accessory for everyday use, shopping, or travel."
    },
    {
        "id": 13,
        "name": "Crochet Christmas Stocking",
        "price": 12.00,
        "image": "images/crochet_christmas_stocking.jpg",
        "description": "Add a handmade touch to your holiday decorations with this crochet Christmas stocking. Perfect for filling with gifts."
    },
    {
        "id": 14,
        "name": "Crochet Baby Blanket",
        "price": 35.00,
        "image": "images/crochet_baby_blanket.jpeg",
        "description": "A soft and warm crochet baby blanket, perfect for newborns. Made with hypoallergenic materials."
    },
    {
        "id": 15,
        "name": "Crochet Phone Case",
        "price": 14.00,
        "image": "images/crochet_phone_case.jpg",
        "description": "Protect your phone with this cute crochet phone case. Handmade with care and available in various sizes."
    }
]
```

Workflow Overview

[products.php] ← Loads the UI, header/footer, modal, and mounts Vue app

↓

[products.js] ← Vue 3 app handles data loading, filtering, UI logic, and fetch()

↓

[products.json] ← Supplies static product data to be rendered

↓

[cart_api.php] ← Receives product info when user adds to cart

How It All Comes Together

1. **User opens products.php.** The Vue app (products.js) mounts on #app.
2. products.json is fetched and stored in products.
3. Products are dynamically filtered, searched, sorted, and paginated.
4. When a user clicks “View Details,” a modal shows product info.
5. Clicking “Add to Cart” sends a POST to cart_api.php.
6. The page URL updates dynamically as users apply filters (great for shareable links).

h. Use of cart.php, cart.js and cart_api.php in the Project

The **shopping cart system** in this PHP-based web application is built using a modular architecture. It separates presentation (cart.php), client-side logic (cart.js), and backend operations (cart_api.php). Below is an in-depth explanation of each component and how they work together.

cart.php:

cart.php is the **main HTML/PHP view file** that presents the shopping cart to the user. It typically includes:

1. Shared Components

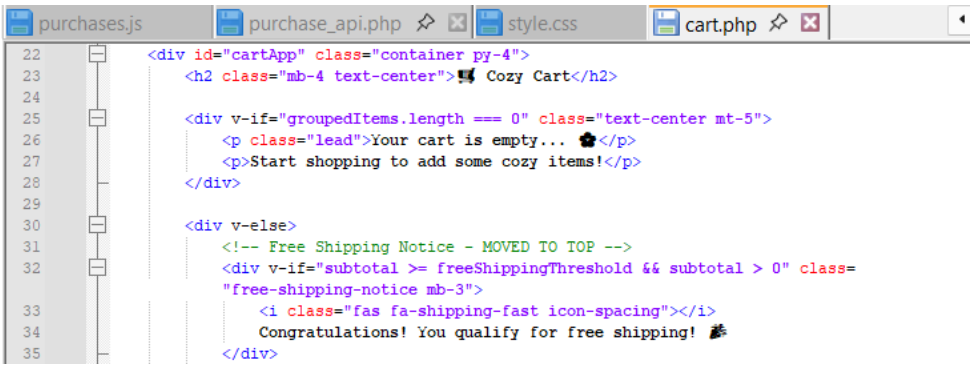
```
<!-- Header -->
<?php include 'commons/header.php'; ?>
```

```
<!-- Footer -->
<?php include 'commons/footer.php'; ?>
```

- These include the navigation bar and footer, keeping the layout consistent across the site.
- The header.php also enforces session-based authentication, ensuring only logged-in users can access their cart.

2. Vue App Mount Point

- The entire cart functions are driven by the Vue app defined in cart.js, which mounts on #cartApp.
- The interface is also created in this file.



```
<!-- Shipping Progress - MOVED TO TOP -->
<div v-else-if="subtotal > 0 && subtotal < freeShippingThreshold" class="shipping-progress mb-3">
    <i class="fas fa-truck icon-spacing"></i>
    Add ${{ (freeShippingThreshold - subtotal).toFixed(2) }} more to qualify for free shipping!
    <div class="progress mt-2" style="height: 8px;">
        <div
            class="progress-bar bg-warning"
            :style="{ width: (subtotal / freeShippingThreshold * 100) + '%' }"
        ></div>
    </div>
</div>

<!-- Cart Controls -->
<div class="cart-container">
    <div class="row align-items-center">
        <div class="col-1 d-flex justify-content-center">
            <input
                type="checkbox"
                id="selectAll"
                @change="toggleSelectAll()"
                :checked="groupedItems.length > 0 && groupedItems.every(item => item.selected)"
            >
        </div>
        <div class="col-7">
            <label for="selectAll">
                Select All Items ({{ selectedItemsCount }} selected)
            </label>
        </div>
        <div class="col-4 text-end">
            <button class="remove-all-btn" @click="clearCart()">
                <i class="fas fa-trash icon-spacing"></i>Clear Cart
            </button>
        </div>
    </div>
</div>

<!-- Cart Items Container -->
<div class="cart-container">
    <div class="cart-item row align-items-center" v-for="item in groupedItems" :key="item.id">
        <div class="col-1 d-flex justify-content-center">
            <input type="checkbox" v-model="item.selected" />
        </div>

        <div class="col-2">
            <img :src="item.image" class="img-fluid" :alt="item.name" />
        </div>

        <div class="col-5">
            <h5>{{ item.name }}</h5>
            <p class="mb-1">Price: ${{ item.price.toFixed(2) }}</p>
            <p class="mb-1">
                Quantity:
                <div class="btn-group btn-group-sm" role="group" aria-label="Quantity controls">
                    <button
                        class="btn btn-outline-secondary"
                        @click="removeOne(item)"
                        :disabled="item.quantity === 1"
                    >-</button>
                    <span class="btn btn-outline-primary disabled quantity-number">
                        {{ item.quantity }}
                    </span>
                    <button class="btn btn-outline-secondary" @click="addOne(item)">+</button>
                </div>
            </p>
            <p>Total: ${{ (item.price * item.quantity).toFixed(2) }}</p>
        </div>

        <div class="col-4 text-end">
            <button class="remove-all-btn" @click="removeAll(item)">
                <i class="fas fa-trash"></i>
            </button>
        </div>
    </div>
</div>

```

```
            <!-- Order Summary -->
            <div class="mt-4 total-price-section">
                <h5 class="mb-3">Order Summary</h5>

                <div class="d-flex justify-content-between mb-2">
                    <span>Subtotal ({{ selectedItemsCount }} items):</span>
                    <span>${{ subtotal.toFixed(2) }}</span>
                </div>

                <div class="d-flex justify-content-between mb-2">
                    <span>Shipping:</span>
                    <span v-if="postage === 0 && subtotal > 0" class="text-success">
                        <i class="fas fa-check icon-spacing"></i>FREE
                    </span>
                    <span v-else-if="postage === 0">$0.00</span>
                    <span v-else>${{ postage.toFixed(2) }}</span>
                </div>

                <hr>

                <div class="d-flex justify-content-between">
                    <h5>Total:</h5>
                    <h5 class="text-primary">${{ totalPrice.toFixed(2) }}</h5>
                </div>

                <div class="mt-3 text-center">
                    <button
                        class="btn btn-brown btn-lg"
                        :disabled="selectedItemsCount === 0"
                        @click="proceedToCheckout"
                    >
                        <i class="fas fa-credit-card icon-spacing"></i>Proceed to Checkout
                    </button>
                </div>
            </div>
        </div>

        <!-- Confirmation Dialog (Custom Modal) - MOVED INSIDE #cartApp -->
        <div class="modal fade" id="confirmationModal" tabindex="-1" aria-labelledby=
        "confirmationModalLabel" aria-hidden="true">
            <div class="modal-dialog modal-dialog-centered">
                <div class="modal-content custom-card">
                    <div class="modal-header">
                        <h5 class="modal-title section-title" id="confirmationModalLabel">
                        {{ confirmTitle }}</h5>
                        <button type="button" class="btn-close" data-bs-dismiss="modal"
                        aria-label="Close" @click="closeConfirm"></button>
                    </div>
                    <div class="modal-body text-muted">
                        {{ confirmMessage }}
                    </div>
                    <div class="modal-footer">
                        <button type="button" class="btn profile-btn cancel-btn"
                        data-bs-dismiss="modal" @click="closeConfirm">Cancel</button>
                        <button type="button" class="btn profile-btn delete-btn" @click=
                        "executeConfirmAction">Confirm</button>
                    </div>
                </div>
            </div>
        </div>
    </div>
```

3. Bootstrap Modal for Confirmations

- Used to confirm actions like removing items or proceeding to checkout.
- Controlled by Vue via the showConfirm() and closeConfirm() methods.

```
        <!-- Confirmation Dialog (Custom Modal) - MOVED INSIDE #cartApp -->
        <div class="modal fade" id="confirmationModal" tabindex="-1" aria-labelledby=
        "confirmationModalLabel" aria-hidden="true">
```

4. JavaScript Integration

- Loads Vue 3 from a CDN.
- Loads and runs cart.js, which renders the cart, updates items, and manages checkout.

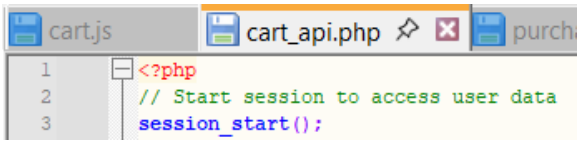
```
    <!-- Load Vue and custom cart script -->
    <script src="https://unpkg.com/vue@3/dist/vue.global.prod.js"></script>
    <script src="cart.js"></script>
```

cart_api.php:

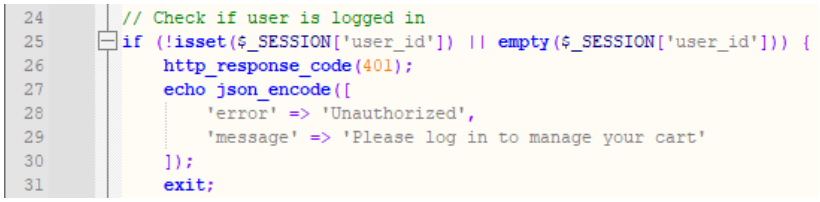
The cart_api.php file is the **backend API** that allows authenticated users to interact with their shopping cart through secure, RESTful endpoints. It handles all cart operations such as loading items, adding products, updating quantities, and deleting items.

1. Session Management and Security

- Starts a PHP session to access the logged-in user's ID.

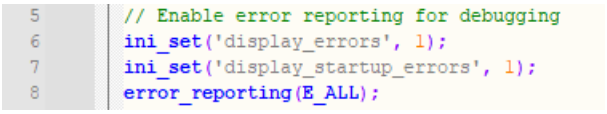


- Verifies that a user is logged in before allowing access to cart data.
- If not authenticated, the script returns an HTTP 401 Unauthorized response.



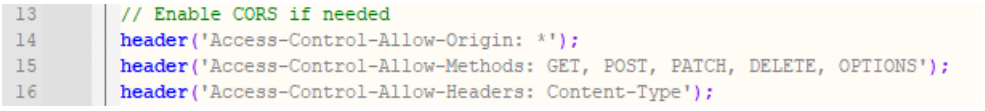
2. Debugging and Development Helpers

- Enables full error reporting during development for easier debugging.



3. Response Format and CORS Setup

- Specifies that the API returns JSON data.
- Enables Cross-Origin Resource Sharing (CORS), allowing requests from the frontend (e.g., cart.js).



- Handles **preflight OPTIONS** requests for CORS, which browsers send before making actual API calls with non-simple methods like PATCH and DELETE.

4. Database Connection

- Connects to the project MySQL database using root credentials.
- Validates connection and returns error if it fails.

```php
// Connect to MySQL
$conn = mysqli_connect('localhost', 'root', '', 'project');
if (!$conn) {
    http_response_code(500);
    echo json_encode([
        'error' => 'Failed to connect to database',
        'details' => mysqli_connect_error()
    ]);
    exit;
}
```

5. Routing Based on HTTP Method

The file uses $_SERVER['REQUEST_METHOD'] to determine what operation to perform:

POST – Add Item to Cart

- Adds a product to the user's cart.
- Expects id, name, price, and image in the request body.
- Uses prepared statements to avoid SQL injection.

```php
if ($method === 'POST') {
    // Add item to cart
    $input = json_decode(file_get_contents('php://input'), true);

    if (!isset($input['id'], $input['name'], $input['price'], $input['image'])) {
        http_response_code(400);
        echo json_encode(['error' => 'Missing required fields: id, name, price, image']);
        exit;
    }

    // Validate input
    if (!is_numeric($input['price']) || $input['price'] < 0) {
        http_response_code(400);
        echo json_encode(['error' => 'Invalid price']);
        exit;
    }

    $stmt = mysqli_prepare($conn, "INSERT INTO cart (user_id, product_id, name, price, image) VALUES (?, ?, ?, ?, ?)");
    if (!$stmt) {
        throw new Exception('Failed to prepare statement: ' . mysqli_error($conn));
    }

    mysqli_stmt_bind_param($stmt, "iisds", $user_id, $input['id'], $input['name'], $input['price'], $input['image']);

    if (mysqli_stmt_execute($stmt)) {
        echo json_encode([
            'success' => true,
            'message' => 'Item added to cart',
            'item_id' => mysqli_insert_id($conn)
        ]);
    } else {
        throw new Exception('Failed to execute statement: ' . mysqli_stmt_error($stmt));
    }

    mysqli_stmt_close($stmt);
```

GET – Load Cart Items

- Retrieves all cart items for the current user.
- Returns the raw list, which cart.js then groups and renders.

```
89      } elseif ($method === 'GET') {
90          // Get all cart items for the current user
91          $stmt = mysqli_prepare($conn, "SELECT * FROM cart WHERE user_id = ? ORDER BY id ASC"
            );
92          if (!$stmt) {
93              throw new Exception('Failed to prepare statement: ' . mysqli_error($conn));
94          }
95
96          mysqli_stmt_bind_param($stmt, "i", $user_id);
97          mysqli_stmt_execute($stmt);
98          $result = mysqli_stmt_get_result($stmt);
99
100         if (!$result) {
101             throw new Exception('Failed to fetch cart items: ' . mysqli_error($conn));
102         }
103
104         $items = [];
105         while ($row = mysqli_fetch_assoc($result)) {
106             $items[] = [
107                 'id' => $row['id'],
108                 'product_id' => $row['product_id'],
109                 'name' => $row['name'],
110                 'price' => floatval($row['price']),
111                 'image' => $row['image']
112             ];
113         }
114
115         mysqli_stmt_close($stmt);
116         echo json_encode($items);
117
```

PATCH – Modify Quantity

- Adds or removes a single instance of an item based on the action field.

```
168     } elseif ($method === 'PATCH') {
169         // Update cart items
170         $input = json_decode(file_get_contents('php://input'), true);
171
172         if (!isset($input['id'], $input['action'])) {
173             http_response_code(400);
174             echo json_encode(['error' => 'Missing required fields: id, action']);
175             exit;
176         }
177
178         $productId = $input['id'];
179         $action = $input['action'];
180
181         if ($action === 'remove_one') {
182             // Remove one instance of the product for current user
183             $stmt = mysqli_prepare($conn, "DELETE FROM cart WHERE user_id = ? AND
                product_id = ? LIMIT 1");
184             if (!$stmt) {
185                 throw new Exception('Failed to prepare statement: ' . mysqli_error($conn));
186             }
187
188             mysqli_stmt_bind_param($stmt, "ii", $user_id, $productId);
189
190             if (mysqli_stmt_execute($stmt)) {
191                 $affected_rows = mysqli_stmt_affected_rows($stmt);
192                 echo json_encode([
193                     'success' => true,
194                     'message' => 'One item removed',
195                     'removed' => $affected_rows > 0
196                 ]);
197             } else {
198                 throw new Exception('Failed to execute statement: ' . mysqli_stmt_error(
                    $stmt));
199             }
200
201             mysqli_stmt_close($stmt);
```

```php
        } elseif ($action === 'add_one') {
            // Add one instance of the product for current user
            if (!isset($input['name'], $input['price'], $input['image'])) {
                http_response_code(400);
                echo json_encode(['error' => 'Missing item details for add_one action']);
                exit;
            }

            // Validate input
            if (!is_numeric($input['price']) || $input['price'] < 0) {
                http_response_code(400);
                echo json_encode(['error' => 'Invalid price']);
                exit;
            }

            $stmt = mysqli_prepare($conn, "INSERT INTO cart (user_id, product_id, name,
            price, image) VALUES (?, ?, ?, ?, ?)");
            if (!$stmt) {
                throw new Exception('Failed to prepare statement: ' . mysqli_error($conn));
            }

            mysqli_stmt_bind_param($stmt, "iisds", $user_id, $productId, $input['name'],
            $input['price'], $input['image']);

            if (mysqli_stmt_execute($stmt)) {
                echo json_encode([
                    'success' => true,
                    'message' => 'One item added',
                    'item_id' => mysqli_insert_id($conn)
                ]);
            } else {
                throw new Exception('Failed to execute statement: ' . mysqli_stmt_error(
                $stmt));
            }

            mysqli_stmt_close($stmt);

        } else {
            http_response_code(400);
            echo json_encode(['error' => 'Invalid action. Use "remove_one" or "add_one"']);
        }

    } else {
        http_response_code(405);
        echo json_encode(['error' => 'Method not allowed']);
    }
```

DELETE – Remove Item(s)

- Handles both:
    - Removing all cart items (clearCart)
    - Removing a single product completely (removeAll(item))

```php
    } elseif ($method === 'DELETE') {
        // Remove items from cart
        $input = json_decode(file_get_contents('php://input'), true);

        if (isset($input['clear_all']) && $input['clear_all'] === true) {
            // Clear entire cart for current user
            $stmt = mysqli_prepare($conn, "DELETE FROM cart WHERE user_id = ?");
            if (!$stmt) {
                throw new Exception('Failed to prepare statement: ' . mysqli_error($conn));
            }

            mysqli_stmt_bind_param($stmt, "i", $user_id);

            if (mysqli_stmt_execute($stmt)) {
                echo json_encode([
                    'success' => true,
                    'message' => 'Cart cleared successfully'
                ]);
            } else {
                throw new Exception('Failed to clear cart: ' . mysqli_stmt_error($stmt));
            }

            mysqli_stmt_close($stmt);
```

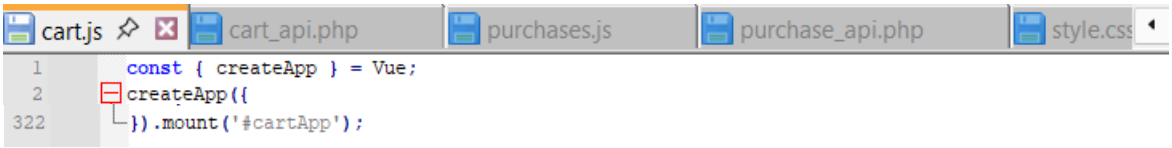
```php
        } elseif (isset($input['id'])) {
            // Remove all instances of a specific product for current user
            $stmt = mysqli_prepare($conn, "DELETE FROM cart WHERE user_id = ? AND
            product_id = ?");
            if (!$stmt) {
                throw new Exception('Failed to prepare statement: ' . mysqli_error($conn));
            }

            mysqli_stmt_bind_param($stmt, "ii", $user_id, $input['id']);

            if (mysqli_stmt_execute($stmt)) {
                $affected_rows = mysqli_stmt_affected_rows($stmt);
                echo json_encode([
                    'success' => true,
                    'message' => 'Items removed from cart',
                    'removed_count' => $affected_rows
                ]);
            } else {
                throw new Exception('Failed to execute statement: ' . mysqli_stmt_error(
                $stmt));
            }

            mysqli_stmt_close($stmt);
        } else {
            http_response_code(400);
            echo json_encode(['error' => 'Missing product ID or clear_all flag']);
        }
```

cart.js:

The cart.js file is a Vue 3-based frontend script that powers the **interactive cart system** of the website. It is mounted on the #cartApp DOM element, typically found in cart.php. This script handles displaying cart items, modifying quantities, calculating totals, and performing the checkout operation.

1. Vue App Initialization

- Initializes a Vue 3 app and binds it to an HTML element with id="cartApp".
- This Vue instance drives the UI logic for the shopping cart on cart.php.



2. Data Properties

- rawItems: Raw list of cart items from the backend (via cart_api.php).
- groupedItems: Aggregated list combining quantities of the same product.
- postageRate & freeShippingThreshold: Used to compute shipping costs.
- confirmDialog, confirmTitle, etc.: Used for custom confirmation modals.

```vue
    data() {
        return {
            rawItems: [],
            groupedItems: [],
            postageRate: 5.00, // Fixed postage rate
            freeShippingThreshold: 50.00, // Free shipping over $50
            confirmDialog: false, // Controls visibility of the custom confirmation modal
            confirmTitle: '',     // Title for the confirmation modal
            confirmMessage: '',   // Message for the confirmation modal
            confirmAction: null,  // Callback function to execute on confirmation
        };
    },
```

3. Computed Properties

These properties automatically update when dependencies change:

- subtotal: Sum of selected items (price × quantity).
- postage: $0 if subtotal exceeds threshold or cart is empty.
- totalPrice: Final amount (subtotal + postage).
- selectedItemsCount: Number of selected items for checkout.

```
computed: {
    subtotal() {
        return this.groupedItems
            .filter(item => item.selected)
            .reduce((sum, item) => sum + (item.price * item.quantity), 0);
    },
    postage() {
        // Free shipping if subtotal is over threshold
        if (this.subtotal >= this.freeShippingThreshold) {
            return 0;
        }
        // No postage if cart is empty
        if (this.subtotal === 0) {
            return 0;
        }
        return this.postageRate;
    },
    totalPrice() {
        return this.subtotal + this.postage;
    },
    selectedItemsCount() {
        return this.groupedItems.filter(item => item.selected).length;
    }
},
```

4. Key Methods

a. groupItems()

- Aggregates rawItems into groupedItems by product ID and counts quantity.

```
methods: {
    groupItems() {
        // Store current selection state before grouping
        const currentSelections = {};
        this.groupedItems.forEach(item => {
            currentSelections[item.id] = item.selected;
        });

        const grouped = {};
        this.rawItems.forEach(item => {
            if (!grouped[item.product_id]) {
                grouped[item.product_id] = {
                    id: item.product_id,
                    name: item.name,
                    price: parseFloat(item.price),
                    image: item.image,
                    quantity: 1,
                    // Preserve previous selection state, default to true for new items
                    selected: currentSelections[item.product_id] !== undefined ?
                    currentSelections[item.product_id] : true
                };
            } else {
                grouped[item.product_id].quantity += 1;
            }
        });
        this.groupedItems = Object.values(grouped);
    },
```

b. reloadCart()

- Fetches the cart data from cart_api.php (GET), then groups items.

```
65      reloadCart() {
66          fetch('cart_api.php')
67              .then(res => {
68                  if (!res.ok) {
69                      throw new Error(`HTTP error! status: ${res.status}`);
70                  }
71                  return res.json();
72              })
73              .then(data => {
74                  this.rawItems = data;
75                  this.groupItems();
76              })
77              .catch(err => {
78                  console.error('Failed to load cart items:', err);
79                  this.showInfoMessage('Error', 'Failed to load cart items. Please
                    refresh the page.'); // Using custom message
80              });
81      },
```

c. removeAll(item)

- Deletes all quantities of a product using a DELETE request to cart_api.php.

```
82      removeAll(item) {
83          this.showConfirm(
84              'Remove Item',
85              `Are you sure you want to remove all ${item.name} from cart?`,
86              () => {
87                  fetch('cart_api.php', {
88                      method: 'DELETE',
89                      headers: { 'Content-Type': 'application/json' },
90                      body: JSON.stringify({ id: item.id })
91                  })
92                      .then(res => {
93                          if (!res.ok) {
94                              throw new Error(`HTTP error! status: ${res.status}`);
95                          }
96                          return res.json();
97                      })
98                      .then(() => this.reloadCart())
99                      .catch(err => {
100                         console.error('Failed to remove item:', err);
101                         this.showInfoMessage('Error', 'Failed to remove item. Please
                            try again.'); // Using custom message
102                     });
103             }
104         );
105     },
```

d. removeOne(item) & addOne(item)

- Updates quantity via PATCH requests to cart_api.php, using action: 'remove_one' or 'add_one'.

```
106         removeOne(item) {
107             fetch('cart_api.php', {
108                 method: 'PATCH',
109                 headers: { 'Content-Type': 'application/json' },
110                 body: JSON.stringify({ id: item.id, action: 'remove_one' })
111             })
112                 .then(res => {
113                     if (!res.ok) {
114                         throw new Error(`HTTP error! status: ${res.status}`);
115                     }
116                     return res.json();
117                 })
118                 .then(() => this.reloadCart())
119                 .catch(err => {
120                     console.error('Failed to remove item:', err);
121                     this.showInfoMessage('Error', 'Failed to update quantity. Please try
                        again.'); // Using custom message
122                 });
123         },
124         addOne(item) {
125             fetch('cart_api.php', {
126                 method: 'PATCH',
127                 headers: { 'Content-Type': 'application/json' },
128                 body: JSON.stringify({
129                     id: item.id,
130                     action: 'add_one',
131                     name: item.name,
132                     price: item.price,
133                     image: item.image
134                 })
135             })
136                 .then(res => {
137                     if (!res.ok) {
138                         throw new Error(`HTTP error! status: ${res.status}`);
139                     }
140                     return res.json();
141                 })
142                 .then(() => this.reloadCart())
143                 .catch(err => {
144                     console.error('Failed to add item:', err);
145                     this.showInfoMessage('Error', 'Failed to update quantity. Please try
                        again.'); // Using custom message
146                 });
147         },
```

e. toggleSelectAll()

- Selects all the items in the cart

```
148         toggleSelectAll() {
149             const allSelected = this.groupedItems.every(item => item.selected);
150             this.groupedItems.forEach(item => {
151                 item.selected = !allSelected;
152             });
153         },
```

f. clearCart()

- Clears the entire cart using a DELETE request with { clear_all: true }.

```
154         clearCart() {
155             this.showConfirm(
156                 'Clear Cart',
157                 'Are you sure you want to clear the entire cart?',
158                 () => {
159                     fetch('cart_api.php', {
160                         method: 'DELETE',
161                         headers: { 'Content-Type': 'application/json' },
162                         body: JSON.stringify({ clear_all: true })
163                     })
164                         .then(res => {
165                             if (!res.ok) {
166                                 throw new Error(`HTTP error! status: ${res.status}`);
167                             }
168                             return res.json();
169                         })
170                         .then(() => this.reloadCart())
171                         .catch(err => {
172                             console.error('Failed to clear cart:', err);
173                             this.showInfoMessage('Error', 'Failed to clear cart. Please try
                                again.'); // Using custom message
174                         });
175                 }
176             );
177         },
```

5. Checkout Functionality

- Validates selection.
- Sends a POST request to purchase_api.php with selected items, subtotal, shipping, and total.
- If successful, it shows a success message and removes only the selected items from the cart.

cart.js | cart_api.php | purchases.js | purchase_api.php | style.css

```
178         async proceedToCheckout() {
179             if (this.selectedItemsCount === 0) {
180                 this.showInfoMessage('Info', 'Please select items to checkout'); // Using
                    custom message
181                 return;
182             }
183
184             // Get only selected items with their quantities
185             const selectedItems = this.groupedItems
186                 .filter(item => item.selected)
187                 .map(item => ({
188                     product_id: item.id,
189                     name: item.name,
190                     price: item.price,
191                     image: item.image,
192                     quantity: item.quantity
193                 }));
194
195             this.showConfirm(
196                 'Proceed to Checkout',
197                 `Proceed to checkout with ${this.selectedItemsCount} items for ${this.
                    totalPrice.toFixed(2)}?`,
198                 async () => {
199                     try {
200                         // Show loading state
201                         const checkoutBtn = document.querySelector('button[data-checkout]');
202                         const originalContent = checkoutBtn ? checkoutBtn.innerHTML : '<i
                            class="fas fa-credit-card"></i> Proceed to Checkout';
```

```
203                 if (checkoutBtn) {
204                     checkoutBtn.disabled = true;
205                     checkoutBtn.innerHTML = '<i class="fas fa-spinner fa-spin"></i>
                        Processing...';
206                 }
207
208                 const response = await fetch('purchase_api.php', {
209                     method: 'POST',
210                     headers: {
211                         'Content-Type': 'application/json'
212                     },
213                     body: JSON.stringify({
214                         action: 'create_from_selected_cart',
215                         selected_items: selectedItems,
216                         subtotal: this.subtotal,
217                         shipping: this.postage,
218                         total: this.totalPrice
219                     })
220                 });
221
222                 if (!response.ok) {
223                     throw new Error(`HTTP error! status: ${response.status}`);
224                 }
225
226                 const result = await response.json();
227
228                 if (result.success) {
229                     this.showInfoMessage('Success', `Order created successfully!
                        Order #${result.order_number}`); // Using custom message
230
231                     // Remove only the selected items from cart
232                     await this.removeSelectedItemsFromCart();
233
234                     // Reload cart to reflect changes
235                     this.reloadCart();
236
237                     // Optionally redirect to purchase history with confirmation
                        modal
238                     this.showConfirm(
239                         'View Purchases',
240                         'Would you like to view your purchase history?',
241                         () => {
242                             window.location.href = 'purchases.php';
243                         }
244                     );
245                 } else {
246                     throw new Error(result.message || 'Failed to create order');
247                 }
248             } catch (error) {
249                 console.error('Checkout failed:', error);
250                 this.showInfoMessage('Error', 'Failed to process checkout. Please
                    try again.'); // Using custom message
251             } finally {
252                 // Reset button state
253                 const checkoutBtn = document.querySelector('button[data-checkout]');
254                 if (checkoutBtn) {
255                     const cartData = { selectedItemsCount: this.selectedItemsCount
                        }; // Simplified for scope
256                     checkoutBtn.disabled = cartData.selectedItemsCount === 0;
257                     checkoutBtn.innerHTML = originalContent;
258                 }
259             }
260         }
261     );
262 },
263
```

```
264         // Helper method to remove selected items from cart after successful checkout
265         async removeSelectedItemsFromCart() {
266             const selectedItems = this.groupedItems.filter(item => item.selected);
267
268             for (const item of selectedItems) {
269                 try {
270                     await fetch('cart_api.php', {
271                         method: 'DELETE',
272                         headers: { 'Content-Type': 'application/json' },
273                         body: JSON.stringify({ id: item.id })
274                     });
275                 } catch (error) {
276                     console.error(`Failed to remove item ${item.name} from cart:`, error);
277                 }
278             }
279         },
```

6. Custom Confirmation Modal

- Triggers a Bootstrap modal dialog with a custom title, message, and callback action.
- Methods like executeConfirmAction() and closeConfirm() help manage modal behaviour.

```
281         // Custom confirmation modal functions
282         showConfirm(title, message, actionCallback) {
283             this.confirmTitle = title;
284             this.confirmMessage = message;
285             this.confirmAction = actionCallback;
286             // Use Bootstrap's JavaScript API to show the modal
287             const modalElement = new bootstrap.Modal(document.getElementById(
                'confirmationModal'));
288             modalElement.show();
289         },
290
291         closeConfirm() {
292             this.confirmDialog = false; // Reset Vue state
293             this.confirmTitle = '';
294             this.confirmMessage = '';
295             this.confirmAction = null;
296             // Use Bootstrap's JavaScript API to hide the modal
297             const modalElement = bootstrap.Modal.getInstance(document.getElementById(
                'confirmationModal'));
298             if (modalElement) {
299                 modalElement.hide();
300             }
301         },
302
303         executeConfirmAction() {
304             if (this.confirmAction) {
305                 this.confirmAction();
306             }
307             this.closeConfirm(); // Close the modal after action
308         },
309
```

7. Helper Function

- A placeholder for custom alerts or toast messages, currently using console.log().

```
310         // Helper for displaying general info messages instead of native alerts
311         showInfoMessage(title, message) {
312             // This is a placeholder. For a real application, you'd integrate a custom
                toast/snackbar
313             // or a more sophisticated modal for general info messages.
314             console.log(`${title}: ${message}`);
315             // For now, if you want a visible (but non-native) alert, you could temporarily
                use:
316             // alert(`${title}: ${message}`);
317         }
```

8. Mounted Lifecycle Hook

- Loads the cart items when the component is first rendered.

```
319     mounted() {
320         this.reloadCart();
321     }
```

How It Connects to Other Files:

cart.php:

- The HTML page where the cart is rendered.
- Includes where this Vue app is mounted.
- Also includes references to this script:

```
175     <!-- Load Vue and custom cart script -->
176     <script src="https://unpkg.com/vue@3/dist/vue.global.prod.js"></script>
177     <script src="cart.js"></script>
```

cart_api.php

- Acts as the backend API for all cart operations.
- Handles GET (fetch cart), PATCH (add/remove quantity), DELETE (remove item/clear cart).
- Used by cart.js via fetch() for real-time updates without page reloads.

The cart.js file is the core of the **interactive cart experience** in the application. It:

- Dynamically renders and updates cart items.
- Calculates prices and shipping in real time.
- Interfaces with cart_api.php for all backend operations.
- Works in tandem with cart.php, which serves as the page shell.

This modular architecture improves user experience and maintains clean separation between frontend logic (cart.js), layout (cart.php), and data operations (cart_api.php).

How It All Connects

[cart.php] - renders HTML + includes Vue + modal

↓

[cart.js] - Vue app controls UI logic and sends requests

↓

[cart_api.php] - securely handles all backend cart actions

- **cart.php**: HTML template and mount point for Vue
- **cart.js**: Vue logic for cart interactivity
- **cart_api.php**: Handles data fetch, modification, and removal securely

i. Use of purchases.php, purchases.js and purchases_api.php in the Project

The **purchase history** system is built using a modular structure similar to the cart system. It separates the **view layer** (purchases.php), the **frontend logic** (purchases.js), and later connects with a backend API (purchases_api.php) that manages data operations securely.

purchases.php

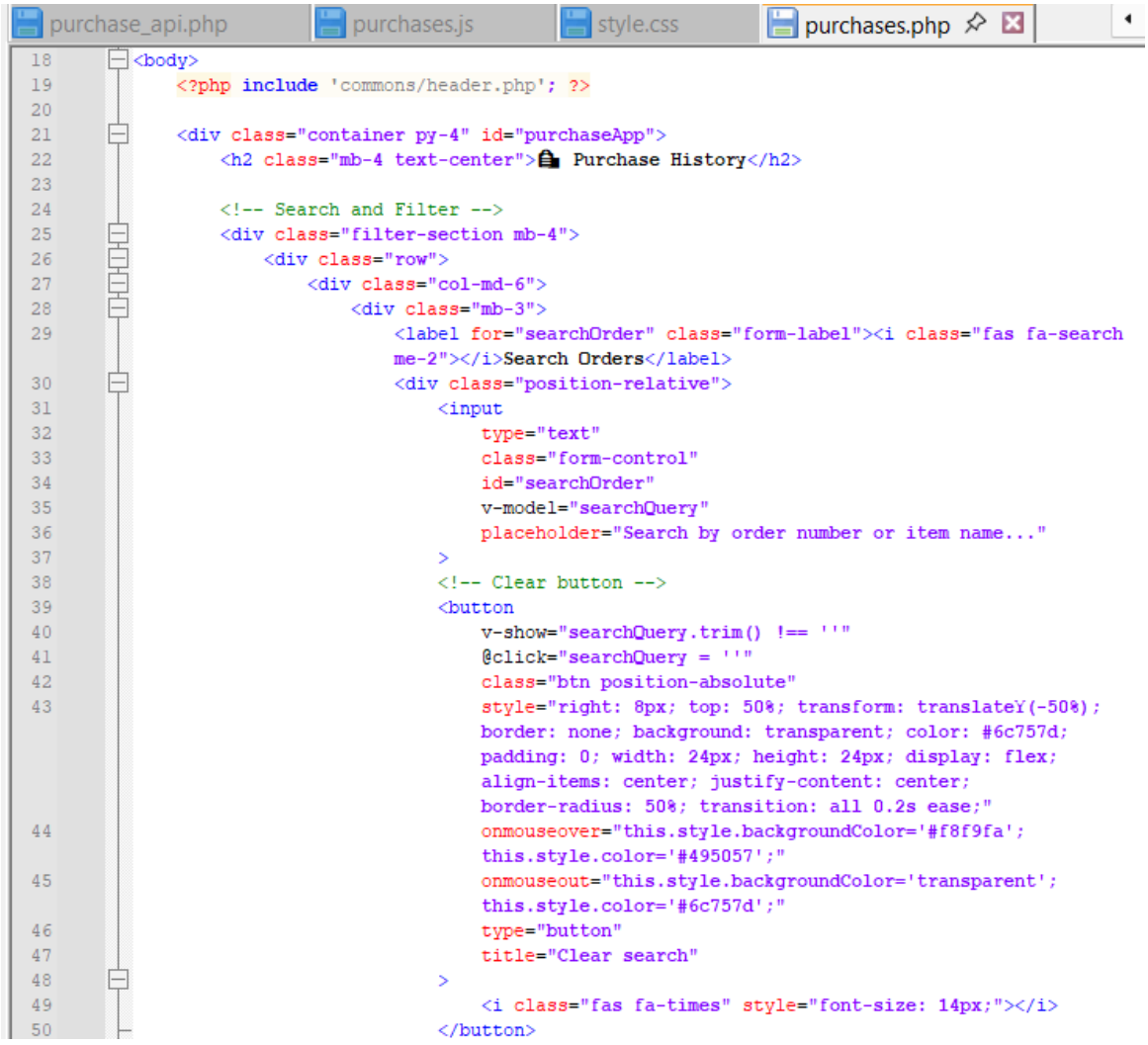
This file is responsible for rendering the purchase history page interface and mounting the Vue app.

1. Shared Components

- Includes common layout files like header.php and footer.php to maintain consistency.
- Protects the route using session checks (inside header.php), ensuring only logged-in users can access it.

2. Vue App Mount Point

- Defines a container where the purchases.js Vue app is mounted.
- Renders interface of all user purchases dynamically within this section.




```
<body>
    <?php include 'commons/header.php'; ?>

    <div class="container py-4" id="purchaseApp">
        <h2 class="mb-4 text-center">Purchase History</h2>

        <!-- Search and Filter -->
        <div class="filter-section mb-4">
            <div class="row">
                <div class="col-md-6">
                    <div class="mb-3">
                        <label for="searchOrder" class="form-label"><i class="fas fa-search
                        me-2"></i>Search Orders</label>
                        <div class="position-relative">
                            <input
                                type="text"
                                class="form-control"
                                id="searchOrder"
                                v-model="searchQuery"
                                placeholder="Search by order number or item name..."
                            >
                            <!-- Clear button -->
                            <button
                                v-show="searchQuery.trim() !== ''"
                                @click="searchQuery = ''"
                                class="btn position-absolute"
                                style="right: 8px; top: 50%; transform: translateY(-50%);
                                border: none; background: transparent; color: #6c757d;
                                padding: 0; width: 24px; height: 24px; display: flex;
                                align-items: center; justify-content: center;
                                border-radius: 50%; transition: all 0.2s ease;"
                                onmouseover="this.style.backgroundColor='#f8f9fa';
                                this.style.color='#495057';"
                                onmouseout="this.style.backgroundColor='transparent';
                                this.style.color='#6c757d';"
                                type="button"
                                title="Clear search"
                            >
                                <i class="fas fa-times" style="font-size: 14px;"></i>
                            </button>
```

```
                    </div>
                </div>
            </div>
            <div class="col-md-3">
                <div class="mb-3">
                    <label for="statusFilter" class="form-label"><i class="fas
                    fa-filter me-2"></i>Filter by Status</label>
                    <select class="form-select" id="statusFilter" v-model="statusFilter"
                    >
                        <option value="">All Orders</option>
                        <option value="pending">Pending</option>
                        <option value="cancelled">Cancelled</option>
                    </select>
                </div>
            </div>
            <div class="col-md-3">
                <div class="mb-3">
                    <label for="sortBy" class="form-label"><i class="fas fa-sort me-2"
                    ></i>Sort by</label>
                    <select class="form-select" id="sortBy" v-model="sortBy">
                        <option value="newest">Newest First</option>
                        <option value="oldest">Oldest First</option>
                        <option value="amount_high">Highest Amount</option>
                        <option value="amount_low">Lowest Amount</option>
                    </select>
                </div>
            </div>
        </div>
    </div>

    <!-- No purchases message -->
    <div v-if="filteredPurchases.length === 0 && !loading" class="text-center mt-5">
        <p class="lead">No purchases found... 🌸</p>
        <p>Your purchase history will appear here once you make some orders!</p>
    </div>

    <!-- Loading state -->
    <div v-if="loading" class="text-center mt-5">
        <div class="spinner-border text-primary" role="status">
            <span class="visually-hidden">Loading...</span>
        </div>
    </div>

    <!-- Purchase Items -->
    <div v-for="purchase in filteredPurchases" :key="purchase.id" class=
    "purchase-container">
        <div class="purchase-header">
            <div class="row align-items-center">
                <div class="col-md-6">
                    <h5 class="mb-1">Order #{{ purchase.order_number }}</h5>
                    <small class="text-muted">{{ formatDate(purchase.order_date) }}
                    </small>
                </div>
                <div class="col-md-3">
                    <span :class="'status-badge status-' + purchase.order_status">
                        {{ purchase.order_status }}
                    </span>
                </div>
                <div class="col-md-3 text-end">
                    <div class="btn-group btn-group-sm">
                        <button
                            class="btn btn-outline-cozy"
                            @click="toggleEdit(purchase.id)"
                            v-if="purchase.order_status === 'pending'"
                        >
                            <i class="fas fa-edit"></i> Edit
                        </button>
```

```
                        <button
                            class="btn btn-danger-soft"
                            @click="cancelOrder(purchase.id)"
                            v-if="purchase.order_status === 'pending' ||
                            purchase.order_status === 'processing'"
                        >
                            <i class="fas fa-times"></i> Cancel
                        </button>
                    </div>
                </div>
            </div>
        </div>

        <!-- Edit Form -->
        <div v-if="editingOrder === purchase.id" class="edit-form">
            <h6>Edit Order Items</h6>
            <div v-for="item in purchase.items" :key="item.id" class="row
            align-items-center mb-2">
                <div class="col-md-6">
                    <strong>{{ item.name }}</strong>
                </div>
                <div class="col-md-2">
                    <input
                        type="number"
                        class="form-control quantity-input"
                        v-model.number="item.newQuantity"
                        min="1"
                    >
                </div>
                <div class="col-md-2">
                    ${{ (item.price * item.newQuantity).toFixed(2) }}
                </div>
                <div class="col-md-2">
                    <button
                        class="btn btn-danger-soft btn-sm"
                        @click="removeItemFromOrder(purchase.id, item.id)"
                    >
                        <i class="fas fa-trash"></i>
                    </button>
                </div>
            </div>
            <div class="mt-3">
                <button class="btn profile-btn delete-btn me-2" @click=
                "saveOrderChanges(purchase.id)">
                    <i class="fas fa-save icon-spacing"></i>Save Changes
                </button>
                <button class="btn profile-btn cancel-btn" @click="cancelEdit()">
                    Cancel
                </button>
            </div>
        </div>
        <!-- Order Items Display -->
        <div v-else>
            <div v-if="purchase.items && purchase.items.length === 0" class=
            "text-center text-muted py-4">
                <p>This order currently has no items.</p>
            </div>
            <div v-for="item in purchase.items" :key="item.id" class="purchase-item">
                <div class="row align-items-center">
                    <div class="col-2">
                        <img :src="item.image" class="img-fluid" :alt="item.name" style=
                        "max-height: 60px;">
                    </div>
                    <div class="col-6">
                        <h6 class="mb-1">{{ item.name }}</h6>
                        <small class="text-muted">Quantity: {{ item.quantity }}</small>
                    </div>
                    <div class="col-2">
                        <span class="fw-bold">${{ item.price.toFixed(2) }}</span>
                    </div>
```

```
                        <div class="col-2 text-end">
                            <span class="fw-bold">${{ (item.price *
                            item.quantity).toFixed(2) }}</span>
                        </div>
                    </div>
                </div>
            </div>

            <!-- Order Summary -->
            <div class="row mt-3 pt-3" style="border-top: 1px solid #e9ecef;">
                <div class="col-md-8"></div>
                <div class="col-md-4">
                    <div class="d-flex justify-content-between mb-1">
                        <span>Subtotal:</span>
                        <span>${{ (purchase.total_amount -
                        purchase.shipping_cost).toFixed(2) }}</span>
                    </div>
                    <div class="d-flex justify-content-between mb-1">
                        <span>Shipping:</span>
                        <span v-if="purchase.shipping_cost === 0" class="text-success">FREE
                        </span>
                        <span v-else>${{ purchase.shipping_cost.toFixed(2) }}</span>
                    </div>
                    <div class="d-flex justify-content-between fw-bold">
                        <span>Total:</span>
                        <span>${{ purchase.total_amount.toFixed(2) }}</span>
                    </div>
                </div>
            </div>
        </div>

        <!-- Confirmation Dialog -->
        <div class="modal fade" id="confirmationModal" tabindex="-1" aria-labelledby=
        "confirmationModalLabel" aria-hidden="true">
            <div class="modal-dialog modal-dialog-centered">
                <div class="modal-content custom-card">
                    <div class="modal-header">
                        <h5 class="modal-title section-title" id="confirmationModalLabel">
                        {{ confirmTitle }}</h5>
                        <button type="button" class="btn-close" data-bs-dismiss="modal"
                        aria-label="Close" @click="closeConfirm"></button>
                    </div>
                    <div class="modal-body text-muted">
                        {{ confirmMessage }}
                    </div>
                    <div class="modal-footer">
                        <button type="button" class="btn profile-btn cancel-btn"
                        data-bs-dismiss="modal" @click="closeConfirm">Cancel</button>
                        <button type="button" class="btn profile-btn delete-btn" @click=
                        "executeConfirmAction">Confirm</button>
                    </div>
                </div>
            </div>
        </div>
```

3. Filters and Sorting Controls

- Includes a search bar to find purchases by **order number** or **item name**.
- Provides dropdowns to filter purchases by **order status** (pending, cancelled) and sort them by **date or amount**.

```
        <!-- Search and Filter -->
        <div class="filter-section mb-4">
            <div class="row">
                <div class="col-md-6">
                    <div class="mb-3">
                        <label for="searchOrder" class="form-label"><i class="fas fa-search
                        me-2"></i>Search Orders</label>
                        <div class="position-relative">
```

```
                        <input
                            type="text"
                            class="form-control"
                            id="searchOrder"
                            v-model="searchQuery"
                            placeholder="Search by order number or item name..."
                        >
                        <!-- Clear button -->
                        <button
                            v-show="searchQuery.trim() !== ''"
                            @click="searchQuery = ''"
                            class="btn position-absolute"
                            style="right: 8px; top: 50%; transform: translateY(-50%);
                            border: none; background: transparent; color: #6c757d;
                            padding: 0; width: 24px; height: 24px; display: flex;
                            align-items: center; justify-content: center;
                            border-radius: 50%; transition: all 0.2s ease;"
                            onmouseover="this.style.backgroundColor='#f8f9fa';
                            this.style.color='#495057';"
                            onmouseout="this.style.backgroundColor='transparent';
                            this.style.color='#6c757d';"
                            type="button"
                            title="Clear search"
                        >
                            <i class="fas fa-times" style="font-size: 14px;"></i>
                        </button>
                    </div>
                </div>
            </div>
            <div class="col-md-3">
                <div class="mb-3">
                    <label for="statusFilter" class="form-label"><i class="fas
                    fa-filter me-2"></i>Filter by Status</label>
                    <select class="form-select" id="statusFilter" v-model="statusFilter"
                    >
                        <option value="">All Orders</option>
                        <option value="pending">Pending</option>
                        <option value="cancelled">Cancelled</option>
                    </select>
                </div>
            </div>
            <div class="col-md-3">
                <div class="mb-3">
                    <label for="sortBy" class="form-label"><i class="fas fa-sort me-2"
                    ></i>Sort by</label>
                    <select class="form-select" id="sortBy" v-model="sortBy">
                        <option value="newest">Newest First</option>
                        <option value="oldest">Oldest First</option>
                        <option value="amount_high">Highest Amount</option>
                        <option value="amount_low">Lowest Amount</option>
                    </select>
                </div>
            </div>
        </div>
    </div>
```

4. Purchase Items Display

- For each order:
    - Shows **order number**, **date**, **status**, and **total breakdown** (subtotal, shipping, total).
    - Lists purchased items with their **image**, **name**, **price**, and **quantity**.

```
    <!-- Purchase Items -->
    <div v-for="purchase in filteredPurchases" :key="purchase.id" class=
    "purchase-container">
        <div class="purchase-header">
            <div class="row align-items-center">
                <div class="col-md-6">
                    <h5 class="mb-1">Order #{{ purchase.order_number }}</h5>
                    <small class="text-muted">{{ formatDate(purchase.order_date) }}
                    </small>
                </div>
```

```
                    <div class="col-md-3">
                        <span :class="'status-badge status-' + purchase.order_status">
                            {{ purchase.order_status }}
                        </span>
                    </div>
                    <div class="col-md-3 text-end">
                        <div class="btn-group btn-group-sm">
                            <button
                                class="btn btn-outline-cozy"
                                @click="toggleEdit(purchase.id)"
                                v-if="purchase.order_status === 'pending'"
                            >
                                <i class="fas fa-edit"></i> Edit
                            </button>
                            <button
                                class="btn btn-danger-soft"
                                @click="cancelOrder(purchase.id)"
                                v-if="purchase.order_status === 'pending' ||
                                purchase.order_status === 'processing'"
                            >
                                <i class="fas fa-times"></i> Cancel
                            </button>
                        </div>
                    </div>
                </div>
            </div>
            <!-- Order Items Display -->
            <div v-else>
                <div v-if="purchase.items && purchase.items.length === 0" class=
                "text-center text-muted py-4">
                    <p>This order currently has no items.</p>
                </div>
                <div v-for="item in purchase.items" :key="item.id" class="purchase-item">
                    <div class="row align-items-center">
                        <div class="col-2">
                            <img :src="item.image" class="img-fluid" :alt="item.name" style=
                            "max-height: 60px;">
                        </div>
                        <div class="col-6">
                            <h6 class="mb-1">{{ item.name }}</h6>
                            <small class="text-muted">Quantity: {{ item.quantity }}</small>
                        </div>
                        <div class="col-2">
                            <span class="fw-bold">${{ item.price.toFixed(2) }}</span>
                        </div>
                        <div class="col-2 text-end">
                            <span class="fw-bold">${{ (item.price *
                            item.quantity).toFixed(2) }}</span>
                        </div>
                    </div>
                </div>
            </div>

            <!-- Order Summary -->
            <div class="row mt-3 pt-3" style="border-top: 1px solid #e9ecef;">
                <div class="col-md-8"></div>
                <div class="col-md-4">
                    <div class="d-flex justify-content-between mb-1">
                        <span>Subtotal:</span>
                        <span>${{ (purchase.total_amount -
                        purchase.shipping_cost).toFixed(2) }}</span>
                    </div>
                    <div class="d-flex justify-content-between mb-1">
                        <span>Shipping:</span>
                        <span v-if="purchase.shipping_cost === 0" class="text-success">FREE
                        </span>
                        <span v-else>${{ purchase.shipping_cost.toFixed(2) }}</span>
                    </div>
                    <div class="d-flex justify-content-between fw-bold">
                        <span>Total:</span>
                        <span>${{ purchase.total_amount.toFixed(2) }}</span>
                    </div>
                </div>
            </div>
        </div>
```

5. Edit and Cancel Actions

- Users can:
  - Click "Edit" to modify quantities of items (only for pending orders).
  - Click "Cancel" to cancel pending or processing orders.
- Editing opens an inline form showing current items, quantities, and total.

```
                <div class="col-md-3 text-end">
                    <div class="btn-group btn-group-sm">
                        <button
                            class="btn btn-outline-cozy"
                            @click="toggleEdit(purchase.id)"
                            v-if="purchase.order_status === 'pending'"
                        >
                            <i class="fas fa-edit"></i> Edit
                        </button>
                        <button
                            class="btn btn-danger-soft"
                            @click="cancelOrder(purchase.id)"
                            v-if="purchase.order_status === 'pending' ||
                            purchase.order_status === 'processing'"
                        >
                            <i class="fas fa-times"></i> Cancel
                        </button>
                    </div>
                </div>
            <!-- Edit Form -->
            <div v-if="editingOrder === purchase.id" class="edit-form">
                <h6>Edit Order Items</h6>
                <div v-for="item in purchase.items" :key="item.id" class="row
                align-items-center mb-2">
                    <div class="col-md-6">
                        <strong>{{ item.name }}</strong>
                    </div>
                    <div class="col-md-2">
                        <input
                            type="number"
                            class="form-control quantity-input"
                            v-model.number="item.newQuantity"
                            min="1"
                        >
                    </div>
                    <div class="col-md-2">
                        ${{ (item.price * item.newQuantity).toFixed(2) }}
                    </div>
                    <div class="col-md-2">
                        <button
                            class="btn btn-danger-soft btn-sm"
                            @click="removeItemFromOrder(purchase.id, item.id)"
                        >
                            <i class="fas fa-trash"></i>
                        </button>
                    </div>
                </div>
                <div class="mt-3">
                    <button class="btn profile-btn delete-btn me-2" @click=
                    "saveOrderChanges(purchase.id)">
                        <i class="fas fa-save icon-spacing"></i>Save Changes
                    </button>
                    <button class="btn profile-btn cancel-btn" @click="cancelEdit()">
                        Cancel
                    </button>
                </div>
            </div>
```

6. Confirmation Modal

- Controlled via Vue and Bootstrap.
- Used to confirm critical actions like cancelling an order or removing items.

```
207         <!-- Confirmation Dialog -->
208         <div class="modal fade" id="confirmationModal" tabindex="-1" aria-labelledby=
            "confirmationModalLabel" aria-hidden="true">
209             <div class="modal-dialog modal-dialog-centered">
210                 <div class="modal-content custom-card">
211                     <div class="modal-header">
212                         <h5 class="modal-title section-title" id="confirmationModalLabel">
                            {{ confirmTitle }}</h5>
213                         <button type="button" class="btn-close" data-bs-dismiss="modal"
                            aria-label="Close" @click="closeConfirm"></button>
214                     </div>
215                     <div class="modal-body text-muted">
216                         {{ confirmMessage }}
217                     </div>
218                     <div class="modal-footer">
219                         <button type="button" class="btn profile-btn cancel-btn"
                            data-bs-dismiss="modal" @click="closeConfirm">Cancel</button>
220                         <button type="button" class="btn profile-btn delete-btn" @click=
                            "executeConfirmAction">Confirm</button>
221                     </div>
222                 </div>
223             </div>
224         </div>
225     </div>
```

7. Script Integration

- Loads Vue 3 from CDN.
- Loads and mounts the logic from purchases.js.

```
232     <!-- Load Vue and custom cart script -->
233     <script src="https://unpkg.com/vue@3/dist/vue.global.prod.js"></script>
234     <script src="purchases.js"></script>
```

purchases_api.php

The **purchases_api.php** file acts as a RESTful backend controller for the purchase history and checkout systems. It handles loading orders, creating purchases (from cart items or selected items), updating item quantities, cancelling orders, and removing specific purchase items.

1. Session Management and Authentication

- Starts the session to access $_SESSION['user_id'].

purchase_api.php | purchases.js

```
1   <?php
2   // Start session to access user data
3   session_start();
```

- Ensures only logged-in users can proceed.
- Returns a 401 Unauthorized response if access is denied.

```
24  // Check if user is logged in
25  if (!isset($_SESSION['user_id']) || empty($_SESSION['user_id'])) {
26      http_response_code(401);
27      echo json_encode([
28          'error' => 'Unauthorized',
29          'message' => 'Please log in to access your orders'
30      ]);
31      exit;
32  }
```

2. Debugging and Development Helpers

- Enables full error reporting during development for easier debugging.

```php
// Enable error reporting for debugging
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);
```

3. CORS and Response Headers

- Enables cross-origin requests and handles CORS preflight (OPTIONS) requests.
- Ensures correct headers for frontend access.

```php
// Enable CORS if needed
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: GET, POST, PATCH, DELETE, OPTIONS');
header('Access-Control-Allow-Headers: Content-Type');

// Handle preflight requests
if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
    http_response_code(200);
    exit;
}
```

4. Database Connection

- Connects to the MySQL project database.
- Sets UTF-8 encoding for multilingual support.
- Errors during connection return HTTP 500.

```php
// Connect to MySQL
$conn = mysqli_connect('localhost', 'root', '', 'project');
if (!$conn) {
    http_response_code(500);
    echo json_encode([
        'error' => 'Failed to connect to database',
        'details' => mysqli_connect_error()
    ]);
    exit;
}
```

5. Routing Based on HTTP Method

GET – Load Purchase History

- Retrieves all purchase records for the authenticated user.
- For each purchase, it also loads the associated items.
- Returns JSON with enriched purchase data:

```php
if ($method === 'GET') {
    // Get all purchases for the current user with their items
    $purchasesQuery = "
        SELECT p.*,
               COUNT(pi.id) as item_count,
               SUM(pi.quantity) as total_items
        FROM purchases p
        LEFT JOIN purchase_items pi ON p.id = pi.purchase_id
        WHERE p.user_id = ?
        GROUP BY p.id
        ORDER BY p.order_date DESC
    ";

    $purchasesStmt = mysqli_prepare($conn, $purchasesQuery);
    if (!$purchasesStmt) {
        throw new Exception('Failed to prepare statement: ' . mysqli_error($conn));
    }

    mysqli_stmt_bind_param($purchasesStmt, "i", $user_id);
    mysqli_stmt_execute($purchasesStmt);
    $purchasesResult = mysqli_stmt_get_result($purchasesStmt);

    if (!$purchasesResult) {
        throw new Exception('Failed to fetch purchases: ' . mysqli_error($conn));
    }

    $purchases = [];
    while ($row = mysqli_fetch_assoc($purchasesResult)) {
        // Get items for this purchase
        $itemsQuery = "SELECT * FROM purchase_items WHERE purchase_id = ? ORDER BY id
        ASC";
        $itemsStmt = mysqli_prepare($conn, $itemsQuery);
        mysqli_stmt_bind_param($itemsStmt, "i", $row['id']);
        mysqli_stmt_execute($itemsStmt);
        $itemsResult = mysqli_stmt_get_result($itemsStmt);

        $items = [];
        while ($itemRow = mysqli_fetch_assoc($itemsResult)) {
            $items[] = [
                'id' => (int)$itemRow['id'],
                'product_id' => (int)$itemRow['product_id'],
                'name' => $itemRow['name'],
                'price' => (float)$itemRow['price'],
                'quantity' => (int)$itemRow['quantity'],
                'image' => $itemRow['image']
            ];
        }
        mysqli_stmt_close($itemsStmt);

        $purchases[] = [
            'id' => (int)$row['id'],
            'order_number' => $row['order_number'],
            'total_amount' => (float)$row['total_amount'],
            'shipping_cost' => (float)$row['shipping_cost'],
            'order_status' => $row['order_status'],
            'order_date' => $row['order_date'],
            'updated_at' => $row['updated_at'],
            'item_count' => (int)$row['item_count'],
            'total_items' => (int)$row['total_items'],
            'items' => $items
        ];
    }

    mysqli_stmt_close($purchasesStmt);
    echo json_encode($purchases);
```

POST – Create a New Purchase

```
118     } elseif ($method === 'POST') {
119         // Create new purchase from selected cart items
120         $input = json_decode(file_get_contents('php://input'), true);
121
122         // Start transaction
123         mysqli_begin_transaction($conn);
124
125         try {
126             // Generate order number
127             $orderNumber = 'ORD-' . date('Y') . '-' . str_pad(rand(1, 9999), 4, '0',
                STR_PAD_LEFT);
128
129             // Check if order number exists and regenerate if needed
130             $checkStmt = mysqli_prepare($conn, "SELECT id FROM purchases WHERE order_number
                = ?");
131             mysqli_stmt_bind_param($checkStmt, "s", $orderNumber);
132             mysqli_stmt_execute($checkStmt);
133             $checkResult = mysqli_stmt_get_result($checkStmt);
134
135             while (mysqli_num_rows($checkResult) > 0) {
136                 $orderNumber = 'ORD-' . date('Y') . '-' . str_pad(rand(1, 9999), 4, '0',
                    STR_PAD_LEFT);
137                 mysqli_stmt_bind_param($checkStmt, "s", $orderNumber);
138                 mysqli_stmt_execute($checkStmt);
139                 $checkResult = mysqli_stmt_get_result($checkStmt);
140             }
141             mysqli_stmt_close($checkStmt);
```

Supports two formats:

a. Selected Items Checkout

- Verifies subtotal and shipping calculations for integrity.
- Begins a transaction and inserts the purchase and purchase items.

```
143             // Handle different request formats
144             if (isset($input['action']) && $input['action'] === 'create_from_selected_cart')
                {
145                 // New format: selected items passed directly
146                 if (!isset($input['selected_items']) || empty($input['selected_items'])) {
147                     throw new Exception('No items selected for checkout');
148                 }
149
150                 $selectedItems = $input['selected_items'];
151                 $subtotal = isset($input['subtotal']) ? (float)$input['subtotal'] : 0;
152                 $shippingCost = isset($input['shipping']) ? (float)$input['shipping'] : 0;
153                 $totalAmount = isset($input['total']) ? (float)$input['total'] : ($subtotal
                    + $shippingCost);
154
155                 // Validate the calculations
156                 $calculatedSubtotal = 0;
157                 foreach ($selectedItems as $item) {
158                     $calculatedSubtotal += (float)$item['price'] * (int)$item['quantity'];
159                 }
160
161                 // Allow small floating point differences
162                 if (abs($calculatedSubtotal - $subtotal) > 0.01) {
163                     throw new Exception('Subtotal mismatch detected');
164                 }
165
166                 $calculatedShipping = $calculatedSubtotal >= 50.00 ? 0.00 : 5.00;
167                 if (abs($calculatedShipping - $shippingCost) > 0.01) {
168                     throw new Exception('Shipping cost mismatch detected');
169                 }
```

b. Legacy Cart Checkout

- Loads all items from the user's cart.
- Groups them by product_id.
- Calculates subtotal, shipping (free if subtotal ≥ 50.00), and total.
- Clears the cart after successful purchase.

```
171             } else {
172                 // Legacy format: get all cart items for user
173                 $cartStmt = mysqli_prepare($conn, "SELECT * FROM cart WHERE user_id = ?");
174                 if (!$cartStmt) {
175                     throw new Exception('Failed to prepare cart statement: ' . mysqli_error(
                        $conn));
176                 }
177
178                 mysqli_stmt_bind_param($cartStmt, "i", $user_id);
179                 mysqli_stmt_execute($cartStmt);
180                 $cartResult = mysqli_stmt_get_result($cartStmt);
181
182                 $cartItems = [];
183                 while ($row = mysqli_fetch_assoc($cartResult)) {
184                     $cartItems[] = $row;
185                 }
186                 mysqli_stmt_close($cartStmt);
187
188                 if (empty($cartItems)) {
189                     throw new Exception('Cart is empty');
190                 }
191
192                 // Group cart items by product_id and calculate totals
193                 $groupedItems = [];
194                 $subtotal = 0;
195
196                 foreach ($cartItems as $item) {
197                     $productId = $item['product_id'];
198                     if (!isset($groupedItems[$productId])) {
199                         $groupedItems[$productId] = [
200                             'product_id' => $productId,
201                             'name' => $item['name'],
202                             'price' => (float)$item['price'],
203                             'image' => $item['image'],
204                             'quantity' => 0
205                         ];
206                     }
207                     $groupedItems[$productId]['quantity']++;
208                     $subtotal += (float)$item['price'];
209                 }
210
211                 $selectedItems = array_values($groupedItems);
212                 $shippingCost = $subtotal >= 50.00 ? 0.00 : 5.00;
213                 $totalAmount = $subtotal + $shippingCost;
214             }
215
216             // Insert purchase record with user_id
217             $purchaseStmt = mysqli_prepare($conn, "
218                 INSERT INTO purchases (user_id, order_number, total_amount, shipping_cost,
                    order_status)
219                 VALUES (?, ?, ?, ?, 'pending')
220             ");
221             mysqli_stmt_bind_param($purchaseStmt, "isdd", $user_id, $orderNumber,
                $totalAmount, $shippingCost);
222
223             if (!mysqli_stmt_execute($purchaseStmt)) {
224                 throw new Exception('Failed to create purchase: ' . mysqli_stmt_error(
                    $purchaseStmt));
225             }
226
227             $purchaseId = mysqli_insert_id($conn);
228             mysqli_stmt_close($purchaseStmt);
```

```php
            // Insert purchase items
            $itemStmt = mysqli_prepare($conn, "
                INSERT INTO purchase_items (purchase_id, product_id, name, price, quantity, image)
                VALUES (?, ?, ?, ?, ?, ?)
            ");

            foreach ($selectedItems as $item) {
                mysqli_stmt_bind_param($itemStmt, "iisdis",
                    $purchaseId,
                    $item['product_id'],
                    $item['name'],
                    $item['price'],
                    $item['quantity'],
                    $item['image']
                );

                if (!mysqli_stmt_execute($itemStmt)) {
                    throw new Exception('Failed to add purchase item: ' . mysqli_stmt_error($itemStmt));
                }
            }
            mysqli_stmt_close($itemStmt);

            // Clear cart for current user only if using legacy format
            if (!isset($input['action']) || $input['action'] !== 'create_from_selected_cart') {
                $clearCartStmt = mysqli_prepare($conn, "DELETE FROM cart WHERE user_id = ?");
                mysqli_stmt_bind_param($clearCartStmt, "i", $user_id);
                if (!mysqli_stmt_execute($clearCartStmt)) {
                    throw new Exception('Failed to clear cart: ' . mysqli_stmt_error($clearCartStmt));
                }
                mysqli_stmt_close($clearCartStmt);
            }

            // Commit transaction
            mysqli_commit($conn);

            echo json_encode([
                'success' => true,
                'message' => 'Purchase created successfully',
                'order_number' => $orderNumber,
                'purchase_id' => $purchaseId,
                'total_amount' => $totalAmount
            ]);

        } catch (Exception $e) {
            mysqli_rollback($conn);
            throw $e;
        }
```

PATCH – Update Orders

```php
    } elseif ($method === 'PATCH') {
        // Update purchase or purchase items (only for current user)
        $input = json_decode(file_get_contents('php://input'), true);

        if (!isset($input['action'])) {
            http_response_code(400);
            echo json_encode(['error' => 'Missing action parameter']);
            exit;
        }
```

a. Update Quantities

- Only works for pending orders.
- Updates each item's quantity and recalculates total and shipping.
- Automatically applies free shipping if applicable.

```php
$action = $input['action'];

if ($action === 'update_items') {
    // Update quantities of items in a purchase
    if (!isset($input['purchase_id'], $input['items'])) {
        http_response_code(400);
        echo json_encode(['error' => 'Missing purchase_id or items']);
        exit;
    }

    // Check if purchase belongs to current user and can be edited
    $statusStmt = mysqli_prepare($conn, "SELECT order_status FROM purchases WHERE id = ? AND user_id = ?");
    mysqli_stmt_bind_param($statusStmt, "ii", $input['purchase_id'], $user_id);
    mysqli_stmt_execute($statusStmt);
    $statusResult = mysqli_stmt_get_result($statusStmt);
    $statusRow = mysqli_fetch_assoc($statusResult);
    mysqli_stmt_close($statusStmt);

    if (!$statusRow) {
        http_response_code(404);
        echo json_encode(['error' => 'Order not found or access denied']);
        exit;
    }

    if ($statusRow['order_status'] !== 'pending') {
        http_response_code(400);
        echo json_encode(['error' => 'Order cannot be edited']);
        exit;
    }

    mysqli_begin_transaction($conn);

    try {
        $updateStmt = mysqli_prepare($conn, "UPDATE purchase_items SET quantity = ? WHERE id = ?");

        foreach ($input['items'] as $item) {
            if ($item['quantity'] > 0) {
                mysqli_stmt_bind_param($updateStmt, "ii", $item['quantity'], $item['id']);
                if (!mysqli_stmt_execute($updateStmt)) {
                    throw new Exception('Failed to update item quantity: ' . mysqli_stmt_error($updateStmt));
                }
            }
        }
        mysqli_stmt_close($updateStmt);

        // Recalculate total amount
        $totalQuery = "
            SELECT SUM(price * quantity) as subtotal
            FROM purchase_items
            WHERE purchase_id = ?
        ";
        $totalStmt = mysqli_prepare($conn, $totalQuery);
        mysqli_stmt_bind_param($totalStmt, "i", $input['purchase_id']);
        mysqli_stmt_execute($totalStmt);
        $totalResult = mysqli_stmt_get_result($totalStmt);
        $totalRow = mysqli_fetch_assoc($totalResult);
        mysqli_stmt_close($totalStmt);

        $subtotal = (float)$totalRow['subtotal'];
        $shippingCost = $subtotal >= 50.00 ? 0.00 : 5.00;
        $newTotalAmount = $subtotal + $shippingCost;

        // Update purchase total (verify user ownership again)
        $updatePurchaseStmt = mysqli_prepare($conn, "
            UPDATE purchases
            SET total_amount = ?, shipping_cost = ?, updated_at = CURRENT_TIMESTAMP
            WHERE id = ? AND user_id = ?
        ");
```

```
                mysqli_stmt_bind_param($updatePurchaseStmt, "ddii", $newTotalAmount,
                $shippingCost, $input['purchase_id'], $user_id);

                if (!mysqli_stmt_execute($updatePurchaseStmt)) {
                    throw new Exception('Failed to update purchase total: ' .
                    mysqli_stmt_error($updatePurchaseStmt));
                }
                mysqli_stmt_close($updatePurchaseStmt);

                mysqli_commit($conn);

                echo json_encode([
                    'success' => true,
                    'message' => 'Order updated successfully',
                    'new_total' => $newTotalAmount
                ]);

            } catch (Exception $e) {
                mysqli_rollback($conn);
                throw $e;
            }
```

b. Cancel Order

- Marks a pending or processing order as cancelled.
- Updates updated_at timestamp.

```
        } elseif ($action === 'cancel_order') {
            // Cancel an order (only user's own orders)
            if (!isset($input['purchase_id'])) {
                http_response_code(400);
                echo json_encode(['error' => 'Missing purchase_id']);
                exit;
            }

            // Check if order belongs to user and can be cancelled
            $statusStmt = mysqli_prepare($conn, "SELECT order_status FROM purchases WHERE
            id = ? AND user_id = ?");
            mysqli_stmt_bind_param($statusStmt, "ii", $input['purchase_id'], $user_id);
            mysqli_stmt_execute($statusStmt);
            $statusResult = mysqli_stmt_get_result($statusStmt);
            $statusRow = mysqli_fetch_assoc($statusResult);
            mysqli_stmt_close($statusStmt);

            if (!$statusRow) {
                http_response_code(404);
                echo json_encode(['error' => 'Order not found or access denied']);
                exit;
            }

            if (!in_array($statusRow['order_status'], ['pending', 'processing'])) {
                http_response_code(400);
                echo json_encode(['error' => 'Order cannot be cancelled']);
                exit;
            }

            $cancelStmt = mysqli_prepare($conn, "
                UPDATE purchases
                SET order_status = 'cancelled', updated_at = CURRENT_TIMESTAMP
                WHERE id = ? AND user_id = ?
            ");
            mysqli_stmt_bind_param($cancelStmt, "ii", $input['purchase_id'], $user_id);

            if (mysqli_stmt_execute($cancelStmt)) {
                echo json_encode([
                    'success' => true,
                    'message' => 'Order cancelled successfully'
                ]);
            } else {
                throw new Exception('Failed to cancel order: ' . mysqli_stmt_error(
                $cancelStmt));
            }
            mysqli_stmt_close($cancelStmt);
```

DELETE – Remove Items

- Deletes the specified item.
- If no items remain, cancels the whole order.
- Otherwise, recalculates total and shipping.

```
    } elseif ($method === 'DELETE') {
        // Delete purchase items or entire purchase (only user's own)
        $input = json_decode(file_get_contents('php://input'), true);

        if (!isset($input['action'])) {
            http_response_code(400);
            echo json_encode(['error' => 'Missing action parameter']);
            exit;
        }

        $action = $input['action'];

        if ($action === 'remove_item') {
            // Remove specific item from purchase
            if (!isset($input['item_id'])) {
                http_response_code(400);
                echo json_encode(['error' => 'Missing item_id']);
                exit;
            }

            // Get purchase_id and check if order belongs to user and can be edited
            $itemQuery = "
                SELECT pi.purchase_id, p.order_status
                FROM purchase_items pi
                JOIN purchases p ON pi.purchase_id = p.id
                WHERE pi.id = ? AND p.user_id = ?
            ";
            $itemStmt = mysqli_prepare($conn, $itemQuery);
            mysqli_stmt_bind_param($itemStmt, "ii", $input['item_id'], $user_id);
            mysqli_stmt_execute($itemStmt);
            $itemResult = mysqli_stmt_get_result($itemStmt);
            $itemRow = mysqli_fetch_assoc($itemResult);
            mysqli_stmt_close($itemStmt);

            if (!$itemRow) {
                http_response_code(404);
                echo json_encode(['error' => 'Item not found or access denied']);
                exit;
            }

            if ($itemRow['order_status'] !== 'pending') {
                http_response_code(400);
                echo json_encode(['error' => 'Order cannot be edited']);
                exit;
            }

            mysqli_begin_transaction($conn);
            try {
                // Delete the item
                $deleteStmt = mysqli_prepare($conn, "DELETE FROM purchase_items WHERE id =
                ?");
                mysqli_stmt_bind_param($deleteStmt, "i", $input['item_id']);

                if (!mysqli_stmt_execute($deleteStmt)) {
                    throw new Exception('Failed to delete item: ' . mysqli_stmt_error(
                    $deleteStmt));
                }
                mysqli_stmt_close($deleteStmt);

                // Check if any items remain
                $countStmt = mysqli_prepare($conn, "SELECT COUNT(*) as item_count FROM
                purchase_items WHERE purchase_id = ?");
                mysqli_stmt_bind_param($countStmt, "i", $itemRow['purchase_id']);
                mysqli_stmt_execute($countStmt);
                $countResult = mysqli_stmt_get_result($countStmt);
                $countRow = mysqli_fetch_assoc($countResult);
                mysqli_stmt_close($countStmt);
```

```
498                 if ($countRow['item_count'] == 0) {
499                     // No items left, cancel the order
500                     $cancelStmt = mysqli_prepare($conn, "
501                         UPDATE purchases
502                         SET order_status = 'cancelled', updated_at = CURRENT_TIMESTAMP
503                         WHERE id = ? AND user_id = ?
504                     ");
505                     mysqli_stmt_bind_param($cancelStmt, "ii", $itemRow['purchase_id'],
                        $user_id);
506
507                     if (!mysqli_stmt_execute($cancelStmt)) {
508                         throw new Exception('Failed to cancel empty order: ' .
                            mysqli_stmt_error($cancelStmt));
509                     }
510                     mysqli_stmt_close($cancelStmt);
511                 } else {
512                     // Recalculate total amount
513                     $totalQuery = "
514                         SELECT SUM(price * quantity) as subtotal
515                         FROM purchase_items
516                         WHERE purchase_id = ?
517                     ";
518                     $totalStmt = mysqli_prepare($conn, $totalQuery);
519                     mysqli_stmt_bind_param($totalStmt, "i", $itemRow['purchase_id']);
520                     mysqli_stmt_execute($totalStmt);
521                     $totalResult = mysqli_stmt_get_result($totalStmt);
522                     $totalRow = mysqli_fetch_assoc($totalResult);
523                     mysqli_stmt_close($totalStmt);
524
525                     $subtotal = (float)$totalRow['subtotal'];
526                     $shippingCost = $subtotal >= 50.00 ? 0.00 : 5.00;
527                     $newTotalAmount = $subtotal + $shippingCost;
528
529                     // Update purchase total (verify user ownership)
530                     $updatePurchaseStmt = mysqli_prepare($conn, "
531                         UPDATE purchases
532                         SET total_amount = ?, shipping_cost = ?, updated_at =
                            CURRENT_TIMESTAMP
533                         WHERE id = ? AND user_id = ?
534                     ");
535                     mysqli_stmt_bind_param($updatePurchaseStmt, "ddii", $newTotalAmount,
                        $shippingCost, $itemRow['purchase_id'], $user_id);
536
537                     if (!mysqli_stmt_execute($updatePurchaseStmt)) {
538                         throw new Exception('Failed to update purchase total: ' .
                            mysqli_stmt_error($updatePurchaseStmt));
539                     }
540                     mysqli_stmt_close($updatePurchaseStmt);
541                 }
542
543                 mysqli_commit($conn);
545                 echo json_encode([
546                     'success' => true,
547                     'message' => 'Item removed successfully'
548                 ]);
549
550             } catch (Exception $e) {
551                 mysqli_rollback($conn);
552                 throw $e;
553             }
```

purchases.js

This is the main Vue 3 script that controls the interactive behavior of the purchase history interface.

1. Reactive Data Properties

- purchases: All fetched purchase records.
- editingOrder: Tracks which order (if any) is being edited.
- searchQuery, statusFilter, sortBy: Used for dynamic filtering and sorting.
- confirmDialog, confirmTitle, confirmMessage, confirmAction: Used to trigger and manage confirmation dialogs.

```
data() {
    return {
        purchases: [],
        loading: false,
        editingOrder: null,
        searchQuery: '',
        statusFilter: '',
        sortBy: 'newest',
        confirmDialog: false, // Controls visibility of the custom confirmation modal
        confirmTitle: '',     // Title for the confirmation modal
        confirmMessage: '',   // Message for the confirmation modal
        confirmAction: null,  // Callback function to execute on confirmation
    };
},
```

2. Computed Properties

- filteredPurchases dynamically filters and sorts:
    - Searches by **order number** or **item name**.
    - Filters by **status**.
    - Sorts by **newest**, **oldest**, **highest amount**, or **lowest amount**.

```
computed: {
    filteredPurchases() {
        let filtered = this.purchases;

        // Filter by search query
        if (this.searchQuery) {
            const query = this.searchQuery.toLowerCase();
            filtered = filtered.filter(purchase =>
                purchase.order_number.toLowerCase().includes(query) ||
                purchase.items.some(item => item.name.toLowerCase().includes(query))
            );
        }

        // Filter by status
        if (this.statusFilter) {
            filtered = filtered.filter(purchase => purchase.order_status === this.
            statusFilter);
        }

        // Sort results
        return filtered.sort((a, b) => {
            switch (this.sortBy) {
                case 'oldest':
                    return new Date(a.order_date) - new Date(b.order_date);
                case 'amount_high':
                    return b.total_amount - a.total_amount;
                case 'amount_low':
                    return a.total_amount - b.total_amount;
                default: // newest
                    return new Date(b.order_date) - new Date(a.order_date);
            }
        });
    }
},
```

3. Purchase Loading and State Management

- loadPurchases() fetches data from purchases_api.php and populates the UI.
- Uses newQuantity to hold editable values separate from original quantities.

```
107         loadPurchases() {
108             this.loading = true;
109             // Return the promise from the fetch chain to allow chaining in
                removeItemFromOrder
110             return fetch('purchase_api.php')
111                 .then(res => {
112                     if (!res.ok) {
113                         throw new Error(`HTTP error! status: ${res.status}`);
114                     }
115                     return res.json();
116                 })
117                 .then(data => {
118                     this.purchases = data.map(purchase => ({
119                         ...purchase,
120                         items: purchase.items.map(item => ({
121                             ...item,
122                             newQuantity: item.quantity
123                         }))
124                     }));
125                     this.loading = false;
126                     return data; // Important: return data for subsequent .then() calls
127                 })
128                 .catch(err => {
129                     console.error('Failed to load purchases:', err);
130                     this.showInfoMessage('Error', 'Failed to load purchase history. Please
                        refresh the page.');
131                     this.loading = false;
132                     throw err; // Re-throw to propagate the error
133                 });
134         },
```

4. Editing Purchases

- toggleEdit(orderId) allows a user to open or close edit mode on a given order.
- saveOrderChanges(orderId) sends a PATCH request to update item quantities.
- cancelEdit() resets any unsaved changes and exits edit mode.

```
136         toggleEdit(orderId) {
137             this.editingOrder = this.editingOrder === orderId ? null : orderId;
138         },
139
140         cancelEdit() {
141             this.editingOrder = null;
142             // Reset quantities to original values
143             this.purchases.forEach(purchase => {
144                 purchase.items.forEach(item => {
145                     item.newQuantity = item.quantity;
146                 });
147             });
148         },
149
150         saveOrderChanges(orderId) {
151             const purchase = this.purchases.find(p => p.id === orderId);
152             if (!purchase) return;
153
154             const updates = purchase.items.map(item => ({
155                 id: item.id,
156                 quantity: item.newQuantity
157             }));
```

```
159             fetch('purchase_api.php', {
160                 method: 'PATCH',
161                 headers: { 'Content-Type': 'application/json' },
162                 body: JSON.stringify({
163                     action: 'update_items',
164                     purchase_id: orderId,
165                     items: updates
166                 })
167             })
168                 .then(res => {
169                     if (!res.ok) {
170                         throw new Error(`HTTP error! status: ${res.status}`);
171                     }
172                     return res.json();
173                 })
174                 .then(() => {
175                     this.editingOrder = null;
176                     this.loadPurchases();
177                     this.showInfoMessage('Success', 'Order updated successfully!'); //
                        Using custom message
178                 })
179                 .catch(err => {
180                     console.error('Failed to update order:', err);
181                     this.showInfoMessage('Error', 'Failed to update order. Please try
                        again.'); // Using custom message
182                 });
183         },
```

5. Removing Items from Orders

- removeItemFromOrder(purchaseId, itemId) removes a specific item via a DELETE request.
- Automatically exits edit mode if the order becomes empty.

```
185         removeItemFromOrder(purchaseId, itemId) {
186             this.showConfirm(
187                 'Remove Item',
188                 'Are you sure you want to remove this item from the order?',
189                 () => {
190                     fetch('purchase_api.php', {
191                         method: 'DELETE',
192                         headers: { 'Content-Type': 'application/json' },
193                         body: JSON.stringify({
194                             action: 'remove_item',
195                             item_id: itemId
196                         })
197                     })
198                         .then(res => {
199                             if (!res.ok) {
200                                 throw new Error(`HTTP error! status: ${res.status}`);
201                             }
202                             return res.json();
203                         })
204                         .then(() => {
205                             // After successful item removal, reload purchases.
206                             // Then, check the state of the purchase and exit edit mode if
                                it's empty.
207                             this.loadPurchases().then(() => {
208                                 const updatedPurchase = this.purchases.find(p => p.id ===
                                    purchaseId);
209                                 // If the purchase no longer has items (and implicitly
                                    cancelled by backend)
210                                 if (updatedPurchase && updatedPurchase.items.length === 0) {
211                                     this.editingOrder = null; // Exit edit mode
212                                 }
213                             });
214                         })
215                         .catch(err => {
216                             console.error('Failed to remove item:', err);
217                             this.showInfoMessage('Error', 'Failed to remove item. Please
                                try again.'); // Using custom message
218                         });
219                 }
220             );
221         },
```

6. Canceling an Order

- cancelOrder(orderId) sends a PATCH request to mark an order as canceled.
- Only works on pending or processing orders.

```
223         cancelOrder(orderId) {
224             this.showConfirm(
225                 'Cancel Order',
226                 'Are you sure you want to cancel this order?',
227                 () => {
228                     fetch('purchase_api.php', {
229                         method: 'PATCH',
230                         headers: { 'Content-Type': 'application/json' },
231                         body: JSON.stringify({
232                             action: 'cancel_order',
233                             purchase_id: orderId
234                         })
235                     })
236                         .then(res => {
237                             if (!res.ok) {
238                                 throw new Error(`HTTP error! status: ${res.status}`);
239                             }
240                             return res.json();
241                         })
242                         .then(() => {
243                             this.loadPurchases(); // Reload purchases to reflect status
                                change
244                             this.showInfoMessage('Success', 'Order cancelled successfully.'
                                ); // Using custom message
245                         })
246                         .catch(err => {
247                             console.error('Failed to cancel order:', err);
248                             this.showInfoMessage('Error', 'Failed to cancel order. Please
                                try again.'); // Using custom message
249                         });
250                 }
251             );
252         },
```

7. Custom Confirmation Modal

- showConfirm(), closeConfirm(), and executeConfirmAction() manage user confirmations for destructive actions like cancel or remove.
- Uses Bootstrap modals with Vue state for control.

```
264         // Custom confirmation modal functions
265         showConfirm(title, message, actionCallback) {
266             this.confirmTitle = title;
267             this.confirmMessage = message;
268             this.confirmAction = actionCallback;
269             // Use Bootstrap's JavaScript API to show the modal
270             const modalElement = new bootstrap.Modal(document.getElementById(
                'confirmationModal'));
271             modalElement.show();
272         },
273
274         closeConfirm() {
275             this.confirmDialog = false; // Reset Vue state
276             this.confirmTitle = '';
277             this.confirmMessage = '';
278             this.confirmAction = null;
279             // Use Bootstrap's JavaScript API to hide the modal
280             const modalElement = bootstrap.Modal.getInstance(document.getElementById(
                'confirmationModal'));
281             if (modalElement) {
282                 modalElement.hide();
283             }
284         },
285
286         executeConfirmAction() {
287             if (this.confirmAction) {
288                 this.confirmAction();
289             }
290             this.closeConfirm(); // Close the modal after action
291         },
```

8. Filter Persistence

- saveFilterState() stores filter settings (search, status, sort) in sessionStorage.
- loadFilterState() restores these on page reload.
- clearFilterState() allows clearing the saved state if needed.

```
66      methods: {
67          // Save current filter state to sessionStorage
68          saveFilterState() {
69              const filterState = {
70                  searchQuery: this.searchQuery,
71                  statusFilter: this.statusFilter,
72                  sortBy: this.sortBy
73              };
74              try {
75                  sessionStorage.setItem('purchaseFilters', JSON.stringify(filterState));
76              } catch (e) {
77                  // Handle cases where sessionStorage might not be available
78                  console.warn('Could not save filter state:', e);
79              }
80          },
81
82          // Load filter state from sessionStorage
83          loadFilterState() {
84              try {
85                  const savedState = sessionStorage.getItem('purchaseFilters');
86                  if (savedState) {
87                      const filterState = JSON.parse(savedState);
88                      this.searchQuery = filterState.searchQuery || '';
89                      this.statusFilter = filterState.statusFilter || '';
90                      this.sortBy = filterState.sortBy || 'newest';
91                  }
92              } catch (e) {
93                  // Handle cases where sessionStorage might not be available or data is
                    corrupted
94                  console.warn('Could not load filter state:', e);
95              }
96          },
97
98          // Clear saved filter state
99          clearFilterState() {
100             try {
101                 sessionStorage.removeItem('purchaseFilters');
102             } catch (e) {
103                 console.warn('Could not clear filter state:', e);
104             }
105         },
```

9. Helper Methods

- formatDate(dateString) converts raw timestamps to readable format.

```
254         formatDate(dateString) {
255             return new Date(dateString).toLocaleDateString('en-US', {
256                 year: 'numeric',
257                 month: 'long',
258                 day: 'numeric',
259                 hour: '2-digit',
260                 minute: '2-digit'
261             });
262         },
```

- showInfoMessage() can be customized to display toasts/snackbars for status feedback.

```
293         // Helper for displaying general info messages instead of native alerts
294         showInfoMessage(title, message) {
295             // You can implement a custom toast/snackbar here, or simply log to console for
                now
296             console.log(`${title}: ${message}`);
297             // For a basic visual cue without native alerts:
298             // alert(`${title}: ${message}`); // Revert to custom UI later if needed
299         }
300     },
```

10. Lifecycle Hook

- mounted() runs on page load:
    - Loads any saved filters.
    - Fetches the user's purchase history from the API.

```
302     mounted() {
303         // Load saved filter state first, then load purchases
304         this.loadFilterState();
305         this.loadPurchases();
306     },
307
308     // Clean up when component is destroyed (optional)
309     beforeUnmount() {
310         // Optionally clear filter state when leaving the page
311         // this.clearFilterState();
312     }
313 }).mount('#purchaseApp');
```

Workflow Overview:

[purchases.php] – HTML layout and Vue mount point

↓

[purchases.js] – Vue logic for loading, displaying, editing, and canceling purchases

↓

[purchases_api.php] – (to be added) RESTful backend handling data persistence

How It All Comes Together

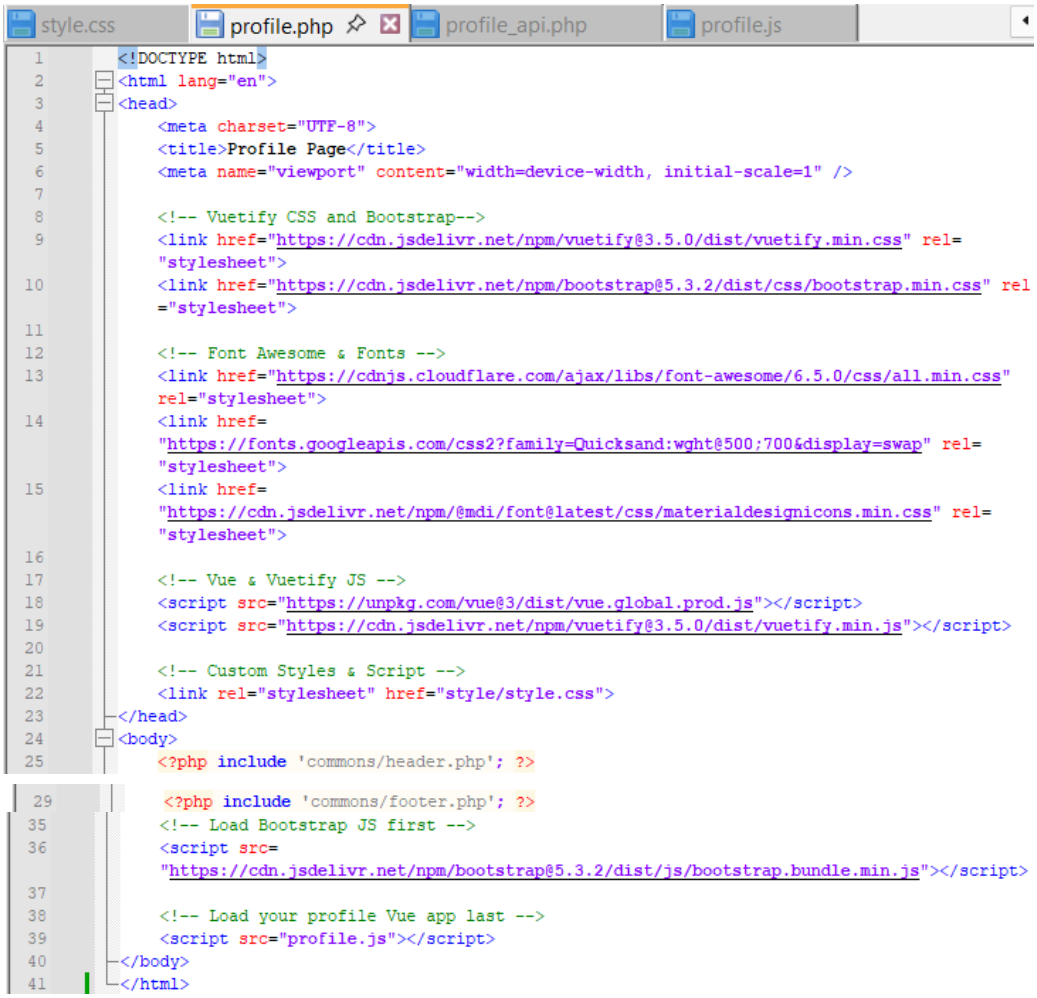
- A user opens the Purchase History page (purchases.php).
- Vue app (purchases.js) mounts on #purchaseApp and loads filter state from sessionStorage.
- The app calls purchases_api.php to load the list of purchases and renders each as a section.
- Users can:
    - Search orders by ID or item name.
    - Filter by status.
    - Sort by date or amount.
    - Edit quantities (if pending).
    - Cancel or remove items with confirmation.
- All actions like PATCH, DELETE, and GET interact asynchronously with purchases_api.php.

j. Use of profile.php, profile.js and profile_api.php in the Project

profile.php

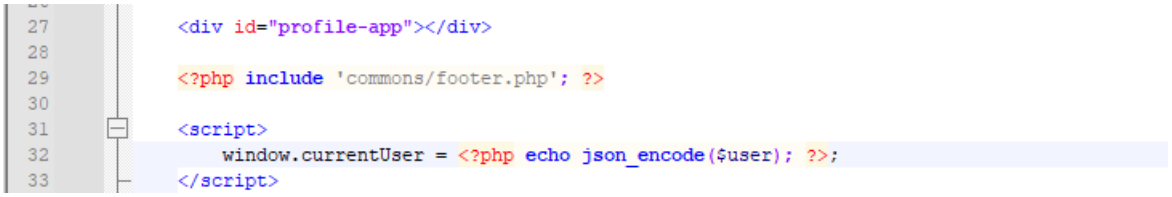
1. Page Structure and Imports

- Loads Bootstrap, Vuetify, Font Awesome, Google Fonts, and style.css.
- Imports Vue 3 and Vuetify JS for SPA behaviour.
- Includes the shared header and footer using <?php include 'commons/header.php'; ?>.



2. Mount Point and Data Injection

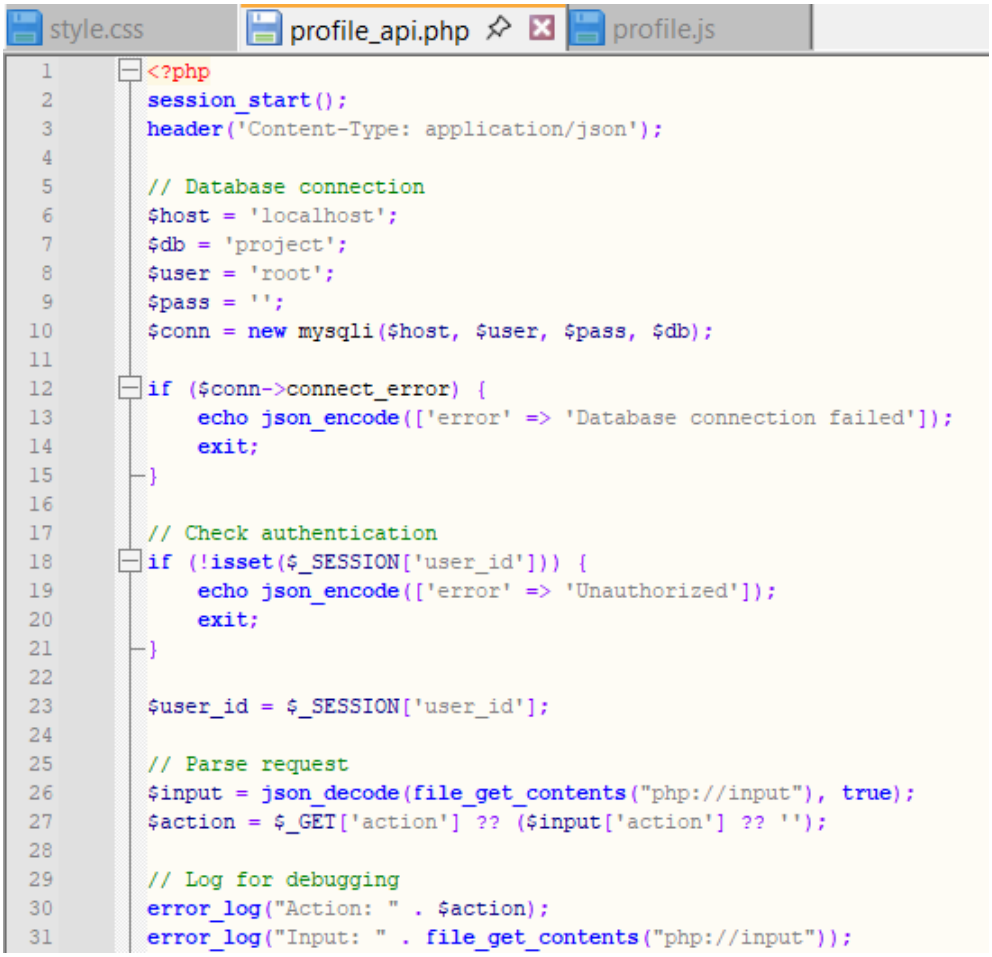
- Injects the current logged-in user's data from the $_SESSION via $user into the frontend (profile.js).
- Vue will mount to #profile-app and use window.currentUser to prefill the user's information.



profile_api.php

1. General Setup

- Establishes a MySQL connection.
- Checks that the user is logged in ($_SESSION['user_id']).
- Parses the action from $_GET['action'] or JSON $_POST.




```php
<?php
session_start();
header('Content-Type: application/json');

// Database connection
$host = 'localhost';
$db = 'project';
$user = 'root';
$pass = '';
$conn = new mysqli($host, $user, $pass, $db);

if ($conn->connect_error) {
    echo json_encode(['error' => 'Database connection failed']);
    exit;
}

// Check authentication
if (!isset($_SESSION['user_id'])) {
    echo json_encode(['error' => 'Unauthorized']);
    exit;
}

$user_id = $_SESSION['user_id'];

// Parse request
$input = json_decode(file_get_contents("php://input"), true);
$action = $_GET['action'] ?? ($input['action'] ?? '');

// Log for debugging
error_log("Action: " . $action);
error_log("Input: " . file_get_contents("php://input"));
```

2. User Profile Update (action=update_user)

- Performs validation on name/email/password.
- Optional password update: only updates password if it passes format checks and confirmation match.
- Returns JSON success/error message.

```php
// ============ USER PROFILE OPERATIONS ============

if ($action === 'update_user') {
    $user = $input['user'] ?? [];
    $name = trim($user['name'] ?? '');
    $email = trim($user['email'] ?? '');
    $password = $user['password'] ?? ''; // Keep as string, can be empty
    $password_confirm = $user['password_confirm'] ?? ''; // New field

    error_log("User update data: " . json_encode($user));

    if (!$name || !$email) {
        echo json_encode(['error' => 'Name and email are required', 'fields' => ['name' =>
        $name, 'email' => $email]]);
        exit;
    }
```

```php
    // Check if email is valid
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        echo json_encode(['error' => 'Invalid email format']);
        exit;
    }

    // Check if email is already taken by another user
    $stmt = $conn->prepare("SELECT id FROM users WHERE email = ? AND id != ?");
    $stmt->bind_param("si", $email, $user_id);
    $stmt->execute();
    $result = $stmt->get_result();

    if ($result->num_rows > 0) {
        echo json_encode(['error' => 'Email is already taken by another user']);
        exit;
    }

    // If password is provided, validate it and its confirmation
    if (!empty($password)) {
        if ($password !== $password_confirm) {
            echo json_encode(['error' => 'Password and confirmation do not match']);
            exit;
        }

        // Validate password strength
        if (strlen($password) < 8) {
            echo json_encode(['error' => 'Password must be at least 8 characters']);
            exit;
        }

        if (!preg_match('/[A-Z]/', $password)) {
            echo json_encode(['error' => 'Password must contain at least one uppercase letter']);
            exit;
        }
        if (!preg_match('/[0-9]/', $password)) {
            echo json_encode(['error' => 'Password must contain at least one number']);
            exit;
        }

        if (!preg_match('/[!@#$%^&*()_+\-=\[\]{};\':"\\|,.<>\/?]/', $password)) {
            echo json_encode(['error' => 'Password must contain at least one special character']);
            exit;
        }

        $hashed = password_hash($password, PASSWORD_DEFAULT);
        $stmt = $conn->prepare("UPDATE users SET name = ?, email = ?, password = ? WHERE id = ?");
        $stmt->bind_param("sssi", $name, $email, $hashed, $user_id);
    } else {
        // Update without changing password
        $stmt = $conn->prepare("UPDATE users SET name = ?, email = ? WHERE id = ?");
        $stmt->bind_param("ssi", $name, $email, $user_id);
    }

    if ($stmt->execute()) {
        // Update session data
        $_SESSION['user_name'] = $name;
        $_SESSION['user_email'] = $email;

        echo json_encode(['success' => true, 'message' => 'Profile updated successfully']);
    } else {
        echo json_encode(['error' => 'Failed to update profile: ' . $conn->error]);
    }
    exit;
}
```

3. Get User Addresses (action=get_addresses)

- Returns all addresses owned by the current user.
- Sorted by creation date (latest first).
- Used to populate address management UI in Vue.

```php
// ============ ADDRESS OPERATIONS ============

// Get all addresses for the current user
if ($action === 'get_addresses') {
    $stmt = $conn->prepare("SELECT * FROM addresses WHERE user_id = ? ORDER BY created_at DESC");
    $stmt->bind_param("i", $user_id);
    $stmt->execute();
    $result = $stmt->get_result();

    $addresses = [];
    while ($row = $result->fetch_assoc()) {
        $addresses[] = $row;
    }

    echo json_encode(['success' => true, 'addresses' => $addresses]);
    exit;
}

// Create a new address
if ($action === 'create_address') {
    $address = $input['address'] ?? [];

    $name = trim($address['name'] ?? '');
    $phone = trim($address['phone'] ?? '');
    $unit_number = trim($address['unit_number'] ?? '');
    $street = trim($address['street'] ?? '');
    $city = trim($address['city'] ?? '');
    $state = trim($address['state'] ?? '');
    $postcode = trim($address['postcode'] ?? '');
    $country = trim($address['country'] ?? 'Malaysia');

    // Validate required fields
    if (!$name || !$phone || !$street || !$city || !$state || !$postcode || !$country) {
        echo json_encode(['error' => 'All required fields must be filled (name, phone, street, city, state, postcode, country)']);
        exit;
    }

    // Validate phone number - allow international formats with country codes
    if (!preg_match('/^[\+]?[\d\s\-\(\)]+$/', $phone) || strlen(preg_replace('/[\s\-\(\)\+]/', '', $phone)) < 7) {
        echo json_encode(['error' => 'Please enter a valid phone number']);
        exit;
    }

    // Validate postcode based on country
    if ($country === 'Malaysia') {
        if (!preg_match('/^\d{5}$/', $postcode)) {
            echo json_encode(['error' => 'Malaysian postcode must be 5 digits']);
            exit;
        }
    } else {
        // Generic postcode validation for other countries
        if (!preg_match('/^[A-Za-z0-9\s\-]{3,10}$/', $postcode)) {
            echo json_encode(['error' => 'Please enter a valid postcode']);
            exit;
        }
    }
```

4. Create Address (action=create_address)

- Inserts new address with user_id linked to session user.
- Applies country-specific postcode validation (e.g., 5-digit for Malaysia).
- Returns success or validation error in JSON.

```
172         // Insert new address
173         $stmt = $conn->prepare("INSERT INTO addresses (user_id, name, phone, unit_number,
            street, city, state, postcode, country) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)");
174         $stmt->bind_param("issssssss", $user_id, $name, $phone, $unit_number, $street, $city,
            $state, $postcode, $country);
175
176         if ($stmt->execute()) {
177             echo json_encode(['success' => true, 'message' => 'Address created successfully',
                'address_id' => $conn->insert_id]);
178         } else {
179             echo json_encode(['error' => 'Failed to create address: ' . $conn->error]);
180         }
181         exit;
182     }
```

5. Update Address (action=update_address)

- Validates field contents and confirms the address belongs to the user.
- Updates record if valid.

```
184     // Update an existing address
185     if ($action === 'update_address') {
186         $address = $input['address'] ?? [];
187
188         $address_id = intval($address['id'] ?? 0);
189         $name = trim($address['name'] ?? '');
190         $phone = trim($address['phone'] ?? '');
191         $unit_number = trim($address['unit_number'] ?? '');
192         $street = trim($address['street'] ?? '');
193         $city = trim($address['city'] ?? '');
194         $state = trim($address['state'] ?? '');
195         $postcode = trim($address['postcode'] ?? '');
196         $country = trim($address['country'] ?? 'Malaysia');
198         if (!$address_id) {
199             echo json_encode(['error' => 'Address ID is required']);
200             exit;
201         }
202
203         // Validate required fields
204         if (!$name || !$phone || !$street || !$city || !$state || !$postcode || !$country) {
205             echo json_encode(['error' => 'All required fields must be filled (name, phone,
                street, city, state, postcode, country)']);
206             exit;
207         }
208
209         // Validate phone number - allow international formats with country codes
210         if (!preg_match('/^[\+]?[\d\s\-\(\)]+$/', $phone) || strlen(preg_replace(
            '/[\s\-\(\)\+]/', '', $phone)) < 7) {
211             echo json_encode(['error' => 'Please enter a valid phone number']);
212             exit;
213         }
214
215         // Validate postcode based on country
216         if ($country === 'Malaysia') {
217             if (!preg_match('/^\d{5}$/', $postcode)) {
218                 echo json_encode(['error' => 'Malaysian postcode must be 5 digits']);
219                 exit;
220             }
221         } else {
222             // Generic postcode validation for other countries
223             if (!preg_match('/^[A-Za-z0-9\s\-]{3,10}$/', $postcode)) {
224                 echo json_encode(['error' => 'Please enter a valid postcode']);
225                 exit;
226             }
227         }
```

```php
    // Check if address belongs to current user
    $stmt = $conn->prepare("SELECT id FROM addresses WHERE id = ? AND user_id = ?");
    $stmt->bind_param("ii", $address_id, $user_id);
    $stmt->execute();
    $result = $stmt->get_result();

    if ($result->num_rows === 0) {
        echo json_encode(['error' => 'Address not found or unauthorized']);
        exit;
    }
    // Update address
    $stmt = $conn->prepare("UPDATE addresses SET name = ?, phone = ?, unit_number = ?,
    street = ?, city = ?, state = ?, postcode = ?, country = ? WHERE id = ? AND user_id = ?"
    );
    $stmt->bind_param("ssssssssii", $name, $phone, $unit_number, $street, $city, $state,
    $postcode, $country, $address_id, $user_id);

    if ($stmt->execute()) {
        echo json_encode(['success' => true, 'message' => 'Address updated successfully']);
    } else {
        echo json_encode(['error' => 'Failed to update address: ' . $conn->error]);
    }
    exit;
}
```

6. Delete Address (action=delete_address)

- Removes the selected address from the database if it belongs to the user.
- Returns JSON confirmation.

```php
// Delete an address
if ($action === 'delete_address') {
    $address_id = intval($input['address_id'] ?? 0);

    if (!$address_id) {
        echo json_encode(['error' => 'Address ID is required']);
        exit;
    }

    // Check if address belongs to current user
    $stmt = $conn->prepare("SELECT id FROM addresses WHERE id = ? AND user_id = ?");
    $stmt->bind_param("ii", $address_id, $user_id);
    $stmt->execute();
    $result = $stmt->get_result();

    if ($result->num_rows === 0) {
        echo json_encode(['error' => 'Address not found or unauthorized']);
        exit;
    }

    // Delete address
    $stmt = $conn->prepare("DELETE FROM addresses WHERE id = ? AND user_id = ?");
    $stmt->bind_param("ii", $address_id, $user_id);

    if ($stmt->execute()) {
        echo json_encode(['success' => true, 'message' => 'Address deleted successfully']);
    } else {
        echo json_encode(['error' => 'Failed to delete address: ' . $conn->error]);
    }
    exit;
}
```

7. Security & Session Handling

- All operations are gated by $_SESSION['user_id'].
- Uses prepared statements to prevent SQL injection.
- Returns detailed JSON error messages for debugging and frontend feedback.

profile.js

This file powers a full-featured profile management interface using Vue 3 and Vuetify. It supports real-time form validation, user info updates, and full CRUD functionality for shipping addresses.

1. Vue App Initialization

- Mounts a Vue app on the #profile-app div inside profile.php.
- Uses Vuetify to provide styled components like v-form, v-btn, and v-text-field.
- Loads window.currentUser (injected from PHP) as the starting data for the user.

```
createApp({
    setup() {
        const user = reactive({ ...window.currentUser });
        const userForm = reactive({
            ...user,
            password: '', // Ensure password is empty initially for security
            password_confirm: '' // New field for password confirmation
        });

        // Form refs for Vuetify validation
        const userFormRef = ref(null);
        const addressFormRef = ref(null);

        // Controls dropdown visibility for profile editing
        const showEditUser = ref(false);

        // New refs for password visibility toggles
        const showPassword = ref(false);
        const showConfirmPassword = ref(false);

        // Address management state
        const addresses = ref([]);
        const showAddressForm = ref(false);
        const editingAddress = ref(null);
        const addressForm = reactive({
            name: '',
            phone: '',
            country_code: '+60', // Default to Malaysia
            unit_number: '',
            street: '',
            city: '',
            state: '',
            postcode: '',
            country: 'Malaysia' // Default to Malaysia
        });

        // Confirmation modal state
        const confirmTitle = ref('');
        const confirmMessage = ref('');
        const confirmAction = ref(null);
```

```
        // Country codes data
        const countryCodes = [
            { code: '+60', country: 'Malaysia', flag: '' },
            { code: '+65', country: 'Singapore', flag: '' },
            { code: '+62', country: 'Indonesia', flag: '' },
            { code: '+66', country: 'Thailand', flag: '' },
            { code: '+84', country: 'Vietnam', flag: '' },
            { code: '+63', country: 'Philippines', flag: '' },
            { code: '+673', country: 'Brunei', flag: '' },
            { code: '+95', country: 'Myanmar', flag: '' },
            { code: '+855', country: 'Cambodia', flag: '' },
            { code: '+856', country: 'Laos', flag: '' },
            { code: '+1', country: 'USA/Canada', flag: '' },
            { code: '+44', country: 'United Kingdom', flag: '' },
            { code: '+61', country: 'Australia', flag: '' },
            { code: '+86', country: 'China', flag: '' },
            { code: '+91', country: 'India', flag: '' },
            { code: '+81', country: 'Japan', flag: '' },
            { code: '+82', country: 'South Korea', flag: '' }
        ];
        // Countries list
        const countries = [
            'Malaysia', 'Singapore', 'Indonesia', 'Thailand', 'Vietnam', 'Philippines',
            'Brunei', 'Myanmar', 'Cambodia', 'Laos', 'United States', 'Canada',
            'United Kingdom', 'Australia', 'China', 'India', 'Japan', 'South Korea',
            'Germany', 'France', 'Italy', 'Spain', 'Netherlands', 'Belgium', 'Switzerland',
            'Sweden', 'Norway', 'Denmark', 'Finland', 'New Zealand', 'Brazil', 'Argentina',
            'Chile', 'Mexico', 'South Africa', 'Egypt', 'UAE', 'Saudi Arabia', 'Turkey'
        ];

        // Malaysian states
        const malaysianStates = [
            'Johor', 'Kedah', 'Kelantan', 'Malacca', 'Negeri Sembilan', 'Pahang',
            'Penang', 'Perak', 'Perlis', 'Sabah', 'Sarawak', 'Selangor', 'Terengganu',
            'Federal Territory of Kuala Lumpur', 'Federal Territory of Labuan', 'Federal
            Territory of Putrajaya'
        ];
        // Other states/provinces by country
        const statesByCountry = {
            'Malaysia': malaysianStates,
            'Singapore': ['Central Region', 'East Region', 'North Region', 'North-East
            Region', 'West Region'],
            'Indonesia': ['Jakarta', 'West Java', 'Central Java', 'East Java', 'Bali',
            'Sumatra', 'Kalimantan', 'Sulawesi'],
            'Thailand': ['Bangkok', 'Chiang Mai', 'Phuket', 'Pattaya', 'Krabi', 'Samui',
            'Hua Hin'],
            'Philippines': ['Metro Manila', 'Cebu', 'Davao', 'Iloilo', 'Cagayan de Oro',
            'Zamboanga'],
            'United States': ['Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California',
            'Colorado', 'Connecticut', 'Delaware', 'Florida', 'Georgia'],
            'Canada': ['Ontario', 'Quebec', 'British Columbia', 'Alberta', 'Manitoba',
            'Saskatchewan', 'Nova Scotia', 'New Brunswick'],
            'Australia': ['New South Wales', 'Victoria', 'Queensland', 'Western Australia',
            'South Australia', 'Tasmania', 'Northern Territory', 'Australian Capital
            Territory']
        };

        // Get states for selected country
        const getStatesForCountry = (country) => {
            return statesByCountry[country] || [];
        };

        // Watch for country changes to update state options
        const onCountryChange = () => {
            const states = getStatesForCountry(addressForm.country);
            if (states.length > 0 && !states.includes(addressForm.state)) {
                addressForm.state = '';
            }
        };
```

2. User Profile Section

- **Displays current user information** using data from the window.currentUser object.
- **Editing functionality**:
  - Bound to a reactive userForm object containing name, email, and optional password fields.
  - Validation rules (userRules) enforce input quality (e.g., valid email, strong password).
  - If a new password is set, a confirmation field is required and must match.
- **Update Process**:
  - saveUser() triggers a confirmation modal.
  - performSaveUser() sends a POST request to profile_api.php with action: "update_user".
  - Server checks for email uniqueness and password requirements.
  - On success, updates the session and displays a success message.

```
219         // ============ USER PROFILE FUNCTIONS ============
220
221         // Actual save user function (called after confirmation)
222         const performSaveUser = async () => {
223             try {
224                 // Only include password and password_confirm if password field is not empt
225                 const payloadUser = {
226                     name: userForm.name,
227                     email: userForm.email
228                 };
229
230                 if (userForm.password) {
231                     payloadUser.password = userForm.password;
232                     payloadUser.password_confirm = userForm.password_confirm;
233                 }
235                 const payload = {
236                     action: 'update_user',
237                     user: payloadUser
238                 };
239
240                 const res = await fetch('profile_api.php', {
241                     method: 'POST',
242                     headers: { 'Content-Type': 'application/json' },
243                     body: JSON.stringify(payload)
244                 });
245
246                 const data = await res.json();
247
248                 if (data.success) {
249                     Object.assign(user, {
250                         name: userForm.name,
251                         email: userForm.email
252                     });
253
254                     userForm.password = '';
255                     userForm.password_confirm = ''; // Clear confirmation field
256                     showEditUser.value = false;
257
258                     nextTick(() => {
259                         userFormRef.value?.resetValidation();
260                     });
261
262                     showConfirm('Success!', 'Your profile has been updated successfully!',
                        null);
263                 } else {
264                     showConfirm('Error', data.error || "Failed to update profile. Please
                        try again.", null);
265                 }
266             } catch (err) {
267                 console.error("Error saving user info:", err);
268                 showConfirm('Error', 'An error occurred while saving your profile. Please
                    try again.', null);
269             }
```

```
    };

    // Save user with validation and confirmation
    const saveUser = async () => {
        const { valid } = await userFormRef.value.validate();
        if (!valid) return;

        showConfirm(
            'Confirm Changes',
            'Are you sure you want to save these changes to your profile?',
            performSaveUser
        );
    };

    // Cancel editing and reset form
    const cancelEdit = () => {
        Object.assign(userForm, user);
        userForm.password = '';
        userForm.password_confirm = ''; // Clear confirmation field
        showEditUser.value = false;

        nextTick(() => {
            userFormRef.value?.resetValidation();
        });
    };
```

3. Address Management

- **Addresses are loaded** via loadAddresses() from the API (action: get_addresses).

```
    // ============ ADDRESS MANAGEMENT FUNCTIONS ============

    // Load addresses from server
    const loadAddresses = async () => {
        try {
            const res = await fetch('profile_api.php', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ action: 'get_addresses' })
            });

            const data = await res.json();
            if (data.success) {
                addresses.value = data.addresses || [];
            } else {
                console.error('Failed to load addresses:', data.error);
            }
        } catch (err) {
            console.error('Error loading addresses:', err);
        }
    };
```

- **Add/Edit Mode**:
    - Form controlled by addressForm, pre-filled in edit mode.
    - Country/state options dynamically update based on selected country (onCountryChange).
    - Phone numbers are split and recombined with a country_code.

```
    // Reset address form
    const resetAddressForm = () => {
        Object.assign(addressForm, {
            name: '',
            phone: '',
            country_code: '+60',
            unit_number: '',
            street: '',
            city: '',
            state: '',
            postcode: '',
            country: 'Malaysia'
        });
        editingAddress.value = null;
    };
```

```
// Show add address form
const showAddAddress = () => {
    resetAddressForm();
    showAddressForm.value = true;
};

// Show edit address form
const showEditAddress = (address) => {
    // Parse the phone number to separate country code and number
    let countryCode = '+60';
    let phoneNumber = address.phone;

    // Try to extract country code from phone number
    for (const cc of countryCodes) {
        if (address.phone.startsWith(cc.code)) {
            countryCode = cc.code;
            phoneNumber = address.phone.substring(cc.code.length).trim();
            break;
        }
    }
        Object.assign(addressForm, {
            ...address,
            phone: phoneNumber,
            country_code: countryCode,
            state: address.state || '',
            country: address.country || 'Malaysia'
        });
        editingAddress.value = address.id;
        showAddressForm.value = true;
    };

    // Cancel address form
    const cancelAddressForm = () => {
        resetAddressForm();
        showAddressForm.value = false;

        nextTick(() => {
            addressFormRef.value?.resetValidation();
        });
    };
```

- **Save Address**:
    - Calls saveAddress() which validates form, and then POSTs to the API with action: create_address or update_address.

```
// Save address (create or update)
const saveAddress = async () => {
    const { valid } = await addressFormRef.value.validate();
    if (!valid) return;

    try {
        // Combine country code and phone number
        const fullPhoneNumber = addressForm.country_code + addressForm.phone;

        const payload = {
            action: editingAddress.value ? 'update_address' : 'create_address',
            address: {
                ...addressForm,
                phone: fullPhoneNumber // Send combined phone number
            }
        };

        if (editingAddress.value) {
            payload.address.id = editingAddress.value;
        }
```

```
            const res = await fetch('profile_api.php', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(payload)
            });

            const data = await res.json();

            if (data.success) {
                await loadAddresses(); // Reload addresses
                cancelAddressForm();
                showConfirm(
                    'Success!',
                    editingAddress.value ? 'Address updated successfully!' : 'Address added successfully!',
                    null
                );
            } else {
                showConfirm('Error', data.error || 'Failed to save address. Please try again.', null);
            }
        } catch (err) {
            console.error('Error saving address:', err);
            showConfirm('Error', 'An error occurred while saving the address. Please try again.', null);
        }
    };
```

- **Delete Address**:
  - Uses a confirmation modal before calling deleteAddress(addressId) with action: delete_address.

```
    // Delete address with confirmation
    const confirmDeleteAddress = (address) => {
        showConfirm(
            'Delete Address',
            `Are you sure you want to delete the address for ${address.name}?`,
            () => deleteAddress(address.id)
        );
    };

    // Actually delete the address
    const deleteAddress = async (addressId) => {
        try {
            const res = await fetch('profile_api.php', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({
                    action: 'delete_address',
                    address_id: addressId
                })
            });

            const data = await res.json();

            if (data.success) {
                await loadAddresses(); // Reload addresses
                showConfirm('Success!', 'Address deleted successfully!', null);
            } else {
                showConfirm('Error', data.error || 'Failed to delete address. Please try again.', null);
            }
        } catch (err) {
            console.error('Error deleting address:', err);
            showConfirm('Error', 'An error occurred while deleting the address. Please try again.', null);
        }
    };
```

4. Validation Systems

- Utilizes Vuetify's built-in form validation (v-form and rules arrays).
- Includes:
    - User Profile Details Validation

```
// ============ USER PROFILE VALIDATION ============

// Rules for Vuetify's validation system
const userRules = {
    name: [
        v => !!v.trim() || 'Name is required',
        v => v.trim().length >= 2 || 'Name must be at least 2 characters',
        v => /^[a-zA-Z\s]+$/.test(v.trim()) || 'Name must contain only alphabetic characters'
    ],
    email: [
        v => !!v.trim() || 'Email is required',
        v => /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/.test(v.trim()) || 'Please enter a valid email address'
    ],
    password: [
        // Password is optional on profile update, but if entered, must meet criteria
        v => !v || v.length >= 8 || 'Password must be at least 8 characters',
        v => !v || /[A-Z]/.test(v) || 'Password must contain at least one uppercase letter',
        v => !v || /[0-9]/.test(v) || 'Password must contain at least one number',
        v => !v || /[!@#$%^&*()_+\-=\[\]{};':"\\|,.<>\/?]/.test(v) || 'Password must contain at least one special character'
    ],
    password_confirm: [
        v => {
            // Only require confirmation if a new password is being set
            if (userForm.password) {
                return !!v || 'Password confirmation is required';
            }
            return true; // Not required if password field is empty
        },
        v => {
            // Only validate match if a new password is being set
            if (userForm.password) {
                return v === userForm.password || 'Passwords do not match';
            }
            return true; // No need to match if password field is empty
        }
    ]
};
```

    - Address Details Validation

```
// ============ ADDRESS VALIDATION ============

const addressRules = {
    name: [
        v => !!v.trim() || 'Name is required',
        v => v.trim().length >= 2 || 'Name must be at least 2 characters'
    ],
    phone: [
        v => !!v.trim() || 'Phone number is required',
        v => /^[\d\s\-\+\(\)]+$/.test(v.trim()) || 'Please enter a valid phone number',
        v => v.trim().length >= 7 || 'Phone number must be at least 7 digits',
        v => v.trim().length <= 15 || 'Phone number must not exceed 15 digits'
    ],
    street: [
        v => !!v.trim() || 'Street address is required',
        v => v.trim().length >= 5 || 'Please enter a complete street address'
    ],
```

```
        city: [
            v => !!v.trim() || 'City is required',
            v => v.trim().length >= 2 || 'City must be at least 2 characters'
        ],
        state: [
            v => !!v.trim() || 'State/Province is required'
        ],
        postcode: [
            v => !!v.trim() || 'Postcode is required',
            v => {
                const code = v.trim();
                // Malaysian postcode validation
                if (addressForm.country === 'Malaysia') {
                    return /^\d{5}$/.test(code) || 'Malaysian postcode must be 5 digits'
                    ;
                }
                // Generic postcode validation for other countries
                return /^[A-Za-z0-9\s\-]{3,10}$/.test(code) || 'Please enter a valid
                postcode';
            }
        ],
        country: [
            v => !!v.trim() || 'Country is required'
        ]
    };
```

5. Modal Management

- A reusable Bootstrap modal (#confirmationModal) is used to:
  - Confirm profile and address changes
  - Show errors and success messages
- Controlled via confirmTitle, confirmMessage, confirmAction, and executeConfirmAction()

```
    // ============ CONFIRMATION MODAL ============

    // Show confirmation dialog
    const showConfirm = (title, message, action) => {
        confirmTitle.value = title;
        confirmMessage.value = message;
        confirmAction.value = action;

        // Use Bootstrap modal
        const modal = new bootstrap.Modal(document.getElementById('confirmationModal'));
        modal.show();
    };

    // Execute the confirmed action
    const executeConfirmAction = () => {
        if (confirmAction.value) {
            confirmAction.value();
        }
        closeConfirm();
    };

    // Close confirmation modal
    const closeConfirm = () => {
        const modal = bootstrap.Modal.getInstance(document.getElementById(
        'confirmationModal'));
        if (modal) {
            modal.hide();
        }
        confirmAction.value = null;
    };
```

6. Lifecycle: Initialization

- onMounted() fetches address list immediately after the component mounts and then returns it:

```
457         onMounted(() => {
458             console.log("Component mounted, current user:", user);
459             loadAddresses();
460         });
461
462         return {
463             user,
464             userForm,
465             userFormRef,
466             userRules,
467             saveUser,
468             cancelEdit,
469             showEditUser,
470             showPassword, // Expose new ref
471             showConfirmPassword, // Expose new ref
472             // Address management
473             addresses,
474             addressForm,
475             addressFormRef,
476             addressRules,
477             showAddressForm,
478             editingAddress,
479             showAddAddress,
480             showEditAddress,
481             saveAddress,
482             cancelAddressForm,
483             confirmDeleteAddress,
484             // New data
485             countryCodes,
486             countries,
487             getStatesForCountry,
488             onCountryChange,
489             // Confirmation modal
490             confirmTitle,
491             confirmMessage,
492             executeConfirmAction,
493             closeConfirm
494         };
495     },
```

7. Template for Profile Page

- The forms on the profile page are coded using Vuetify.

```
497     template: `
498         <v-app>
499             <v-main class="profile-main-content">
500                 <v-container class="mt-6" style="max-width: 1000px;">
501                     <!-- User Profile Section -->
502                     <v-card class="pa-6 custom-card mb-6">
503                         <div class="d-flex justify-space-between align-center mb-4">
504                             <h2 class="section-title">User Profile</h2>
505                             <v-btn
506                                 v-if="!showEditUser"
507                                 class="profile-btn btn-brown"
508                                 prepend-icon="mdi-square-edit-outline"
509                                 @click="showEditUser = true"
510                             >
511                                 Edit Info
512                             </v-btn>
513                         </div>
514
515                         <div v-if="!showEditUser" class="user-info-display">
516                             <p><strong>Name:</strong> {{ user.name }}</p>
517                             <p><strong>Email:</strong> {{ user.email }}</p>
518                         </div>
519
520                         <div v-if="showEditUser" class="mt-4 profile-form">
521                             <h4 class="mb-4">Edit Profile Information</h4>
522                             <v-form ref="userFormRef" lazy-validation>
523                                 <v-text-field
524                                     label="Name"
525                                     v-model="userForm.name"
526                                     :rules="userRules.name"
527                                     variant="outlined"
528                                     class="mb-3"
529                                 ></v-text-field>
530                                 <v-text-field
531                                     label="Email"
532                                     v-model="userForm.email"
533                                     :rules="userRules.email"
534                                     variant="outlined"
535                                     class="mb-3"
536                                 ></v-text-field>
537                                 <v-text-field
538                                     label="New Password (leave empty to keep current)"
539                                     v-model="userForm.password"
540                                     :rules="userRules.password"
541                                     :type="showPassword ? 'text' : 'password'"
542                                     variant="outlined"
543                                     class="mb-3"
544                                     hint="Leave empty if you don't want to change your
                                    password"
545                                     persistent-hint
546                                     :append-inner-icon="showPassword ? 'mdi-eye' :
                                    'mdi-eye-off'"
547                                     @click:append-inner="showPassword = !showPassword"
548                                 ></v-text-field>
```

```
                            <v-text-field
                                label="Confirm New Password"
                                v-model="userForm.password_confirm"
                                :rules="userRules.password_confirm"
                                :type="showConfirmPassword ? 'text' : 'password'"
                                variant="outlined"
                                class="mb-4"
                                :disabled="!userForm.password"
                                :append-inner-icon="showConfirmPassword ? 'mdi-eye' :
                                'mdi-eye-off'"
                                @click:append-inner="showConfirmPassword =
                                !showConfirmPassword"
                            ></v-text-field>
                            <div class="d-flex gap-2">
                                <v-btn class="profile-btn success-btn"
                                prepend-icon="mdi-content-save"
                                @click="saveUser">
                                    Save Changes
                                </v-btn>
                                <v-btn class="profile-btn cancel-btn"
                                @click="cancelEdit">
                                    Cancel
                                </v-btn>
                            </div>
                        </v-form>
                    </div>
                </v-card>

                <!-- Shipping Addresses Section -->
                <v-card class="pa-6 custom-card">
                    <div class="d-flex justify-space-between align-center mb-4">
                        <h2 class="section-title">Shipping Addresses</h2>
                        <v-btn
                            v-if="!showAddressForm"
                            class="profile-btn btn-brown"
                            @click="showAddAddress"
                            prepend-icon="mdi-plus"
                        >
                            Add Address
                        </v-btn>
                    </div>
                    <!-- Address Form -->
                    <div v-if="showAddressForm" class="mb-6">
                        <h4 class="mb-4">{{ editingAddress ? 'Edit Address' : 'Add New
                        Address' }}</h4>
                        <v-form ref="addressFormRef" lazy-validation>
                            <v-row>
                                <v-col cols="12" md="6">
                                    <v-text-field
                                        label="Full Name"
                                        v-model="addressForm.name"
                                        :rules="addressRules.name"
                                        variant="outlined"
                                    ></v-text-field>
                                </v-col>
                                <v-col cols="12" md="6">
                                    <div class="d-flex gap-2">
                                        <v-select
                                            label="Country Code"
                                            v-model="addressForm.country_code"
                                            :items="countryCodes"
                                            item-title="country"
                                            item-value="code"
                                            variant="outlined"
                                            style="flex: 0 0 140px; max-height: 10px;"
                                            :menu-props="{ maxHeight: 300 }"
                                        >
                                            <template v-slot:selection="{ item }">
                                                <span>{{ item.raw.flag }} {{
                                                item.raw.code }}</span>
                                            </template>
                                            <template v-slot:item="{ item, props }">
                                                <v-list-item v-bind="props">
                                                    <template v-slot:prepend>
                                                        <span class="me-2">{{
                                                        item.raw.flag }}</span>
                                                    </template>
```

```
                                        <v-list-item-title>{{ item.raw.code }} - {{ item.raw.country }}</v-list-item-title>
                                    </v-list-item>
                                </template>
                            </v-select>
                            <v-text-field
                                label="Phone Number"
                                v-model="addressForm.phone"
                                :rules="addressRules.phone"
                                variant="outlined"
                                style="flex: 1;"
                            ></v-text-field>
                        </div>
                    </v-col>
                </v-row>
                <v-row>
                    <v-col cols="12" md="4">
                        <v-text-field
                            label="Unit Number (Optional)"
                            v-model="addressForm.unit_number"
                            variant="outlined"
                            hint="House, Apartment, suite, unit, etc."
                            persistent-hint
                        ></v-text-field>
                    </v-col>
                    <v-col cols="12" md="8">
                        <v-text-field
                            label="Street Address"
                            v-model="addressForm.street"
                            :rules="addressRules.street"
                            variant="outlined"
                        ></v-text-field>
                    </v-col>
                </v-row>
                <v-row>
                    <v-col cols="12" md="4">
                        <v-text-field
                            label="City"
                            v-model="addressForm.city"
                            :rules="addressRules.city"
                            variant="outlined"
                        ></v-text-field>
                    </v-col>
                    <v-col cols="12" md="8">
                        <v-text-field
                            label="Postcode"
                            v-model="addressForm.postcode"
                            :rules="addressRules.postcode"
                            variant="outlined"
                        ></v-text-field>
                    </v-col>
                </v-row>
                <v-row>
                    <v-col cols="12" md="4">
                        <v-select
                            label="State/Province"
                            v-model="addressForm.state"
                            :items="getStatesForCountry(addressForm.country)"
                            :rules="addressRules.state"
                            variant="outlined"
                            :no-data-text="'No states available for ' + addressForm.country"
                            :menu-props="{ maxHeight: 300 }"
                        ></v-select>
                    </v-col>
                    <v-col cols="12" md="8">
                        <v-select
                            label="Country"
                            v-model="addressForm.country"
                            :items="countries"
                            :rules="addressRules.country"
                            variant="outlined"
                            @update:modelValue="onCountryChange"
                            :menu-props="{ maxHeight: 300 }"
                        ></v-select>
                    </v-col>
                </v-row>
```

```
                                <div class="d-flex gap-2 mt-4">
                                    <v-btn class="profile-btn success-btn"
                                    prepend-icon="mdi-content-save"
                                    @click="saveAddress">
                                        {{ editingAddress ? 'Update Address' : 'Save
                                    Address' }}
                                    </v-btn>
                                    <v-btn class="profile-btn cancel-btn"
                                    @click="cancelAddressForm">
                                        Cancel
                                    </v-btn>
                                </div>
                            </v-form>
                        </div>

                        <!-- Address List -->
                        <div v-if="!showAddressForm">
                            <div v-if="addresses.length === 0" class="text-center
                            text-muted py-8">
                                <v-icon size="64"
                                class="mb-4">mdi-map-marker-outline</v-icon>
                                <p>No shipping addresses found. Add your first address to
                                get started!</p>
                            </div>

                            <v-row v-else>
                                <v-col v-for="address in addresses" :key="address.id"
                                cols="12" md="6" lg="4">
                                    <v-card class="address-card pa-4" elevation="2">
                                        <div class="address-header d-flex
                                        justify-space-between align-start mb-3">
                                            <h5>{{ address.name }}</h5>
                                            <div class="address-actions">
                                                <v-btn
                                                    icon
                                                    size="small"
                                                    variant="text"
                                                    @click="showEditAddress(address)"
                                                    class="me-1"
                                                >
                                                    <v-icon size="20">mdi-pencil</v-icon>
                                                </v-btn>
                                                <v-btn
                                                    icon
                                                    size="small"
                                                    variant="text"
                                                    @click="confirmDeleteAddress(address)"
                                                >
                                                    <v-icon size="20">mdi-delete</v-icon>
                                                </v-btn>
                                            </div>
                                        </div>
                                        <div class="address-details text-muted">
                                            <p class="mb-1"><v-icon size="20"
                                            }}</p>
                                            <p class="mb-1">
                                                <v-icon size="20"
                                                class="me-2">mdi-map-marker</v-icon>
                                                <span v-if="address.unit_number">{{
                                                address.unit_number }}, </span>{{
                                                address.street }}
                                            </p>
                                            <p class="mb-1">{{ address.city }}, {{
                                            address.state }} {{ address.postcode }}</p>
                                            <p class="mb-0"><strong>{{ address.country ||
                                            'Malaysia' }}</strong></p>
                                        </div>
                                    </v-card>
                                </v-col>
                            </v-row>
                        </div>
                    </v-card>
                </v-container>
            </v-main>
```

```
            <!-- Confirmation Modal -->
            <div class="modal fade" id="confirmationModal" tabindex="-1"
            aria-labelledby="confirmationModalLabel" aria-hidden="true">
                <div class="modal-dialog modal-dialog-centered">
                    <div class="modal-content custom-card">
                        <div class="modal-header">
                            <h5 class="modal-title section-title"
                            id="confirmationModalLabel">{{ confirmTitle }}</h5>
                            <button type="button" class="btn-close" data-bs-dismiss="modal"
                            aria-label="Close" @click="closeConfirm"></button>
                        </div>
                        <div class="modal-body text-muted">
                            {{ confirmMessage }}
                        </div>
                        <div class="modal-footer">
                            <button type="button" class="btn profile-btn cancel-btn"
                            data-bs-dismiss="modal" @click="closeConfirm">
                                {{ confirmTitle === 'Success!' || confirmTitle === 'Error'
                                ? 'OK' : 'Cancel' }}
                            </button>
                            <button
                                v-if="confirmTitle !== 'Success!' && confirmTitle !==
                                'Error'"
                                type="button"
                                class="btn profile-btn success-btn"
                                @click="executeConfirmAction"
                            >
                                Confirm
                            </button>
                        </div>
                    </div>
                </div>
            </div>
        </v-app>
    `
}).use(vuetify).mount('#profile-app');
```

Workflow Overview

[profile.php] - HTML layout, loads user session

↓

[profile.js] - Vue app handles UI interaction, form validation, and fetches

↓

[profile_api.php] - Secure backend API handling:

→ update_user (update profile info)

→ get_addresses (load address list)

→ create_address (add address)

→ update_address (edit address)

→ delete_address (remove address)

How It All Works Together

1. User opens profile.php and sees a prefilled Vue form based on session data (window.currentUser).
2. Vue fetches address list from profile_api.php?action=get_addresses.

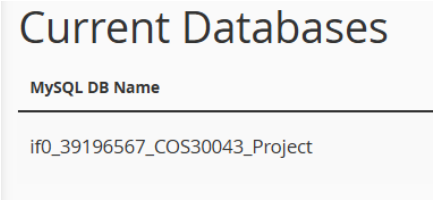
3. User can update name/email/password → submitted to action=update_user.
4. User can manage addresses (add/edit/delete) → all use POST requests with specific action flags.
5. Backend validates, updates DB, and returns JSON responses.
6. Vue displays toast/snackbar messages or validation hints accordingly.

4. Modifications to Code (hosted on InfinityFree Hosting Platform)

I had to modify some of my preexisting codes after I have successfully created the domain for my website on InfinityFree so that the website can be accessible. These modifications were done right after the domain was created.

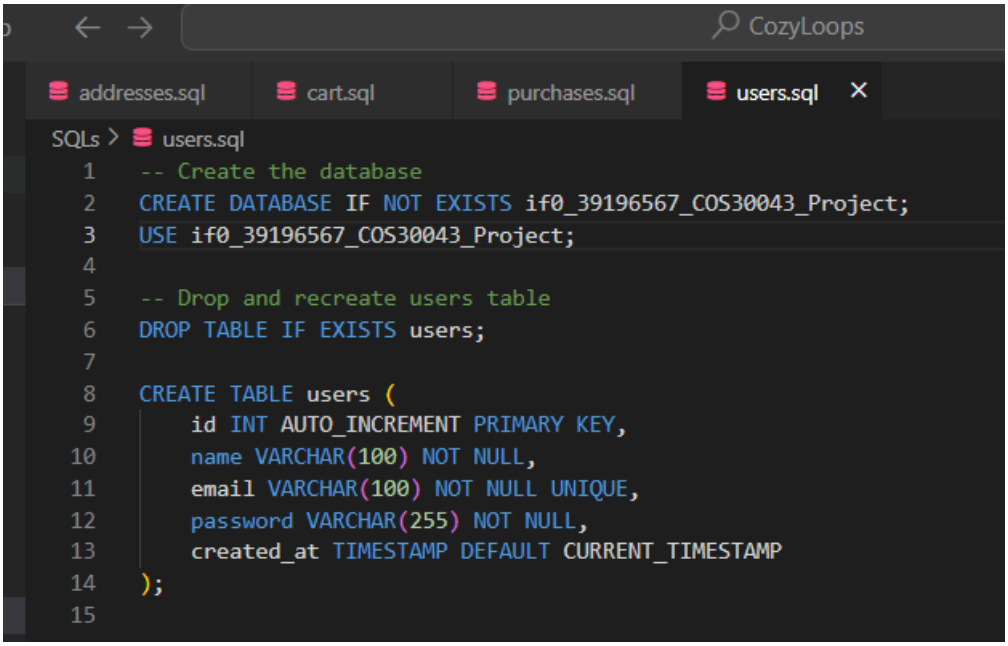
a. Modification to the SQL files

I had to modify the database connection in the SQL files so that they can be imported into the database I had created in phpMyAdmin in InfinityFree hosting platform. The database name in the SQL files is changed to if0_39196567_COS30043_Project as shown in the image below.



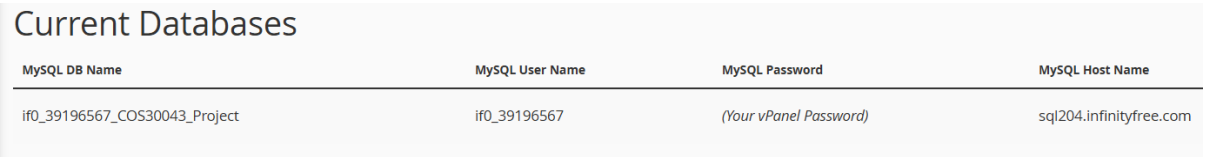
The SQL files that are modified are:

- addresses.sql
- cart.sql
- purchases.sql
- users.sql

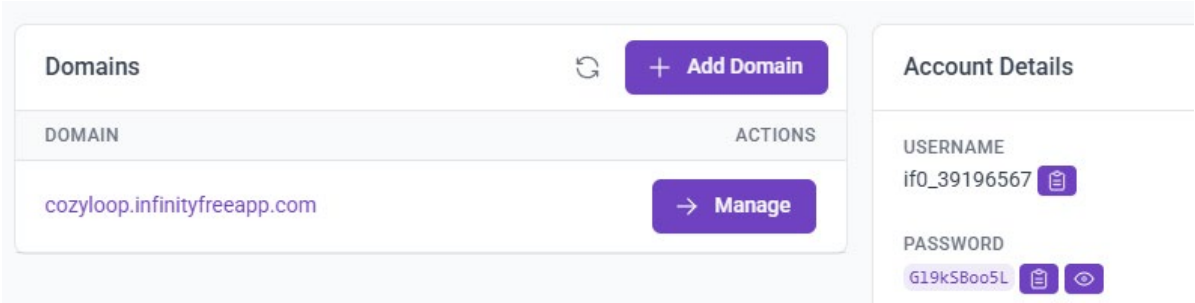


b. Modification to the PHP files

I had to modify the database connection in the PHP files that handle the backend functionalities so that the web pages that communicate with these files can successfully be accessed and is functional online. The database details (database name, username, password and hostname) in the PHP files are changed to the ones shown in the image below.



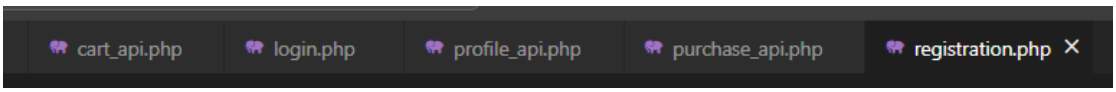
The password for the database can be obtained from the domain homepage. This password was automatically generated when the domain is created.



The PHP files that are modified are:

- cart_api.php
- login.php
- profile_api.php
- purchase_api.php
- registration.php

These files need to be modified because they interact in the database.



Here is an example of what is modified in cart_api.php. This line of code is used in login.php, profile_api.php, purchase_api.php and registration.php as well.

cart_api.php

```php
// Enable CORS if needed
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: GET, POST, PATCH, DELETE, OPTIONS');
header('Access-Control-Allow-Headers: Content-Type');

// Handle preflight requests
if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
    http_response_code(200);
    exit;
}

// Check if user is logged in
if (!isset($_SESSION['user_id']) || empty($_SESSION['user_id'])) {
    http_response_code(401);
    echo json_encode([
        'error' => 'Unauthorized',
        'message' => 'Please log in to manage your cart'
    ]);
    exit;
}

$user_id = $_SESSION['user_id'];

// Connect to MySQL
$conn = mysqli_connect('sql204.infinityfree.com', 'if0_39196567', 'G19kSBoo5L', 'if0_39196567_COS30043_Project');
if (!$conn) {
```

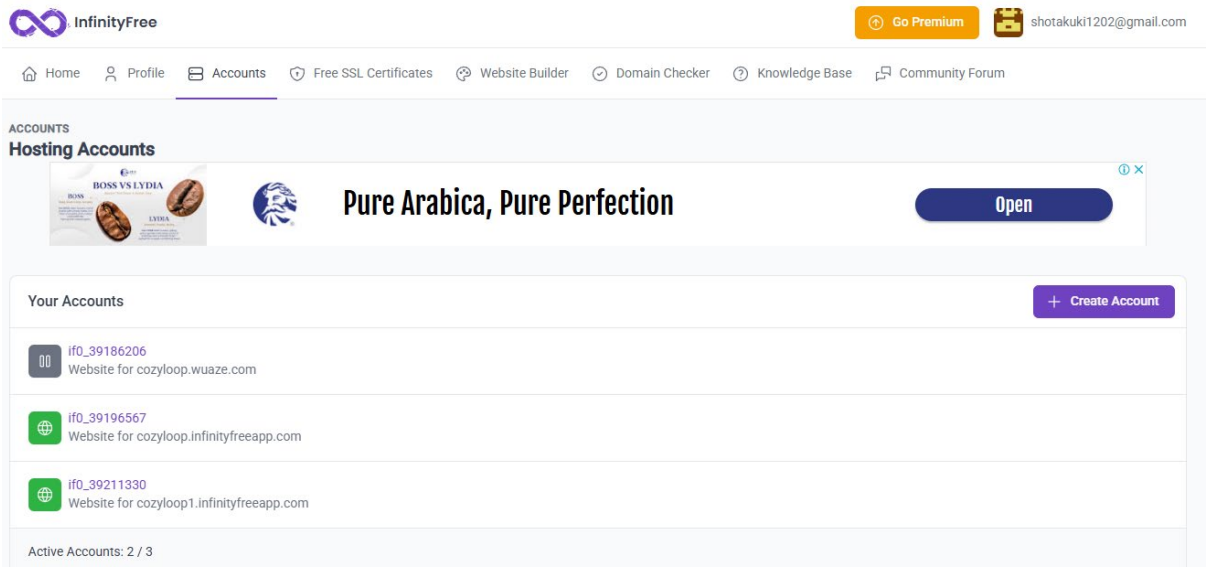
These changes were necessary because free hosting platforms have different database server configurations than local development environments.

5. How the Website was Hosted on InfinityFree

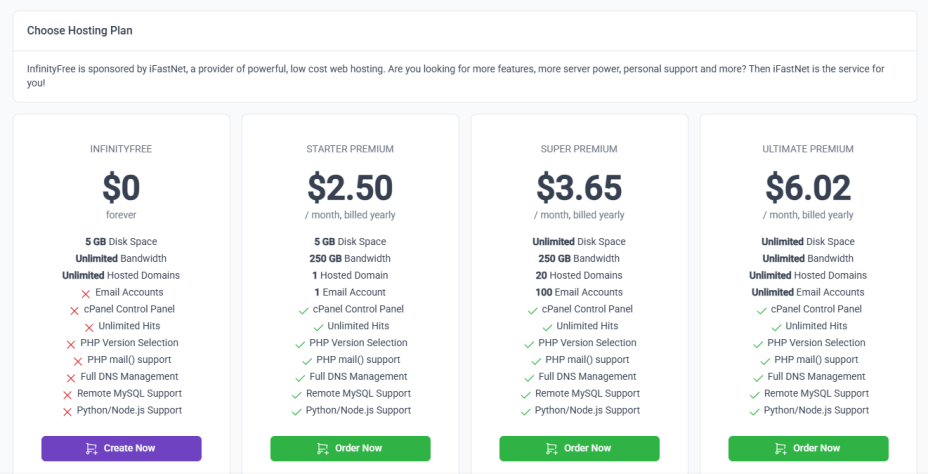
1) I went to the InfinityFree Website by clicking this link: https://www.infinityfree.com

2) I signed up and logged in with my own personal email account.

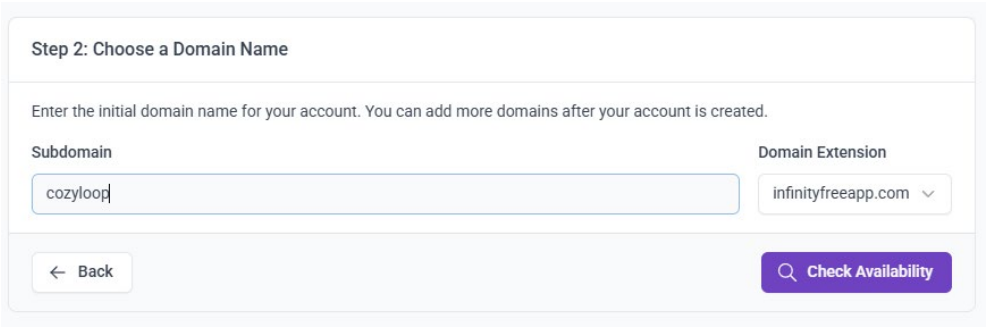
3) The image below is the homepage of InfinityFree after I have logged in. There is a purple button called 'Create Account' and I clicked it to create a new account.



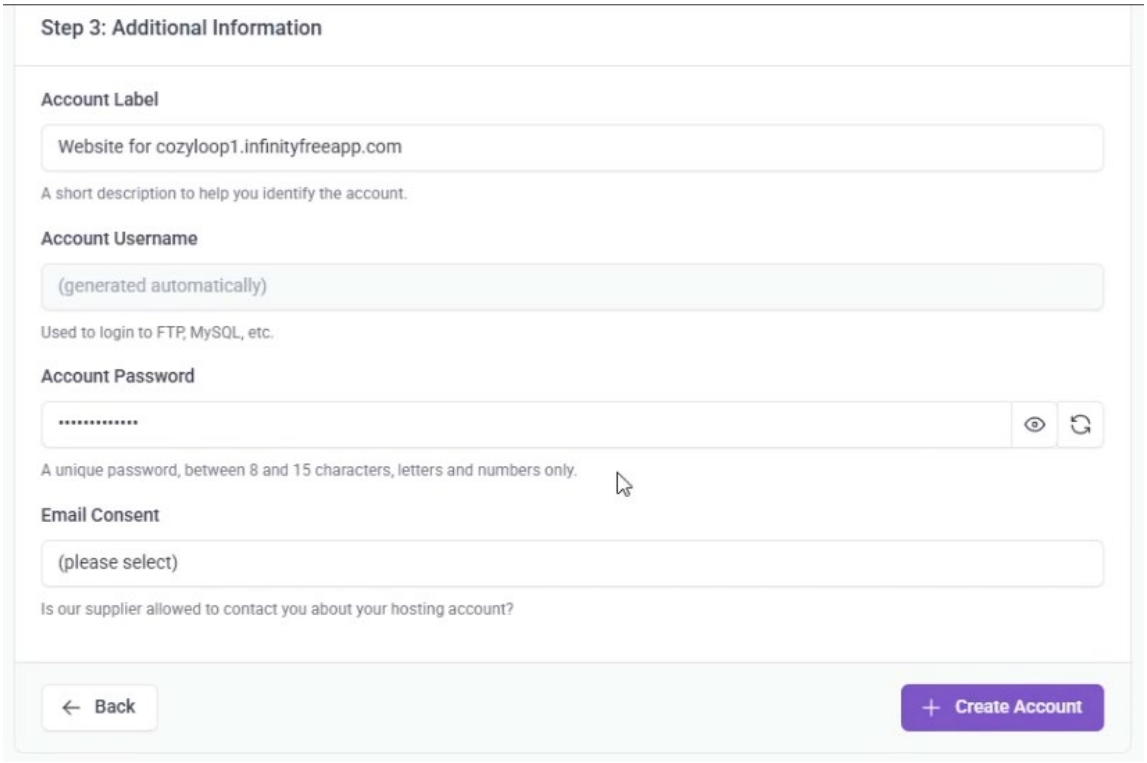
4) Then I chose the $0 plan to create that new account.



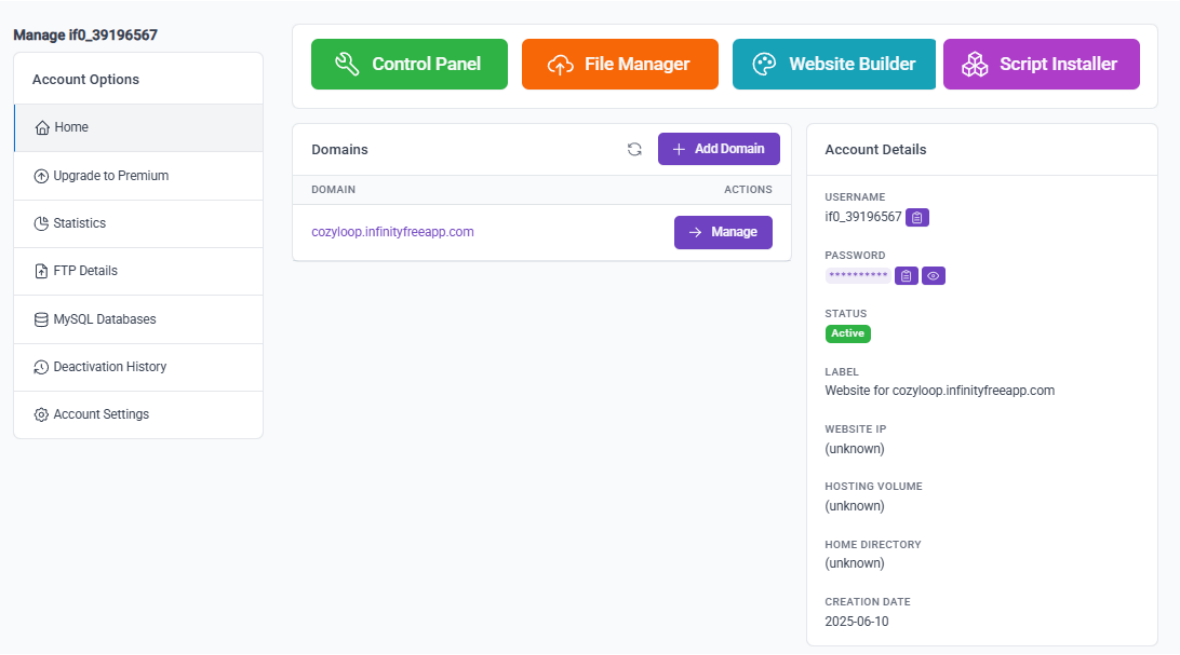
5) Then, I filled it the next form to create a new domain. I named it cozyloop and choose inifinityfreeapp.com as the domain extension.

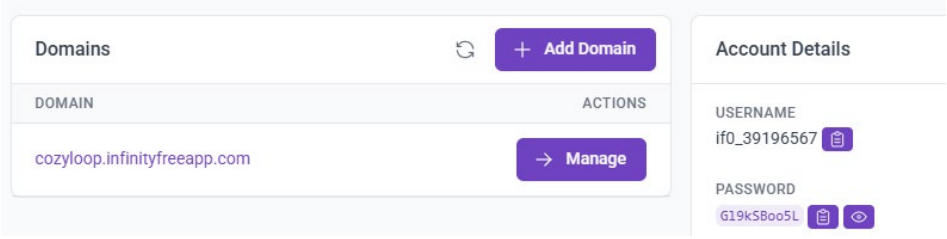


6)Then, I filled up the step 3 form.

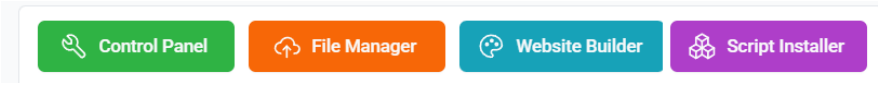


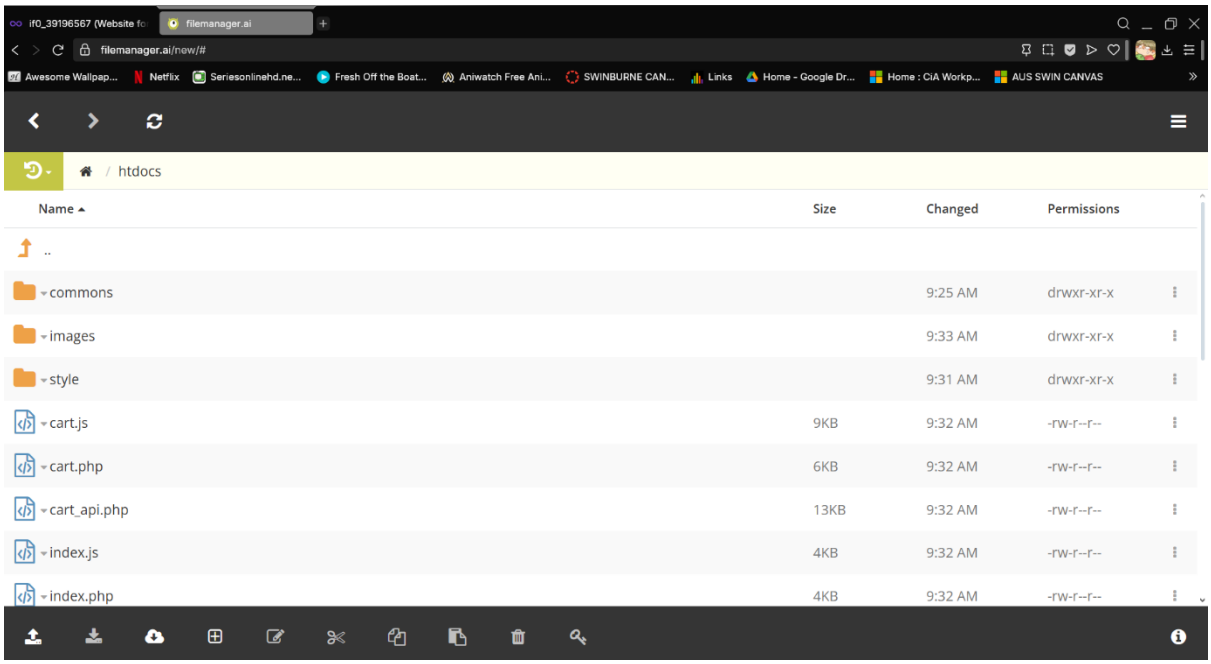
7) After the 3 forms, the new account and new domain have been created. The password and username are automatically generated.

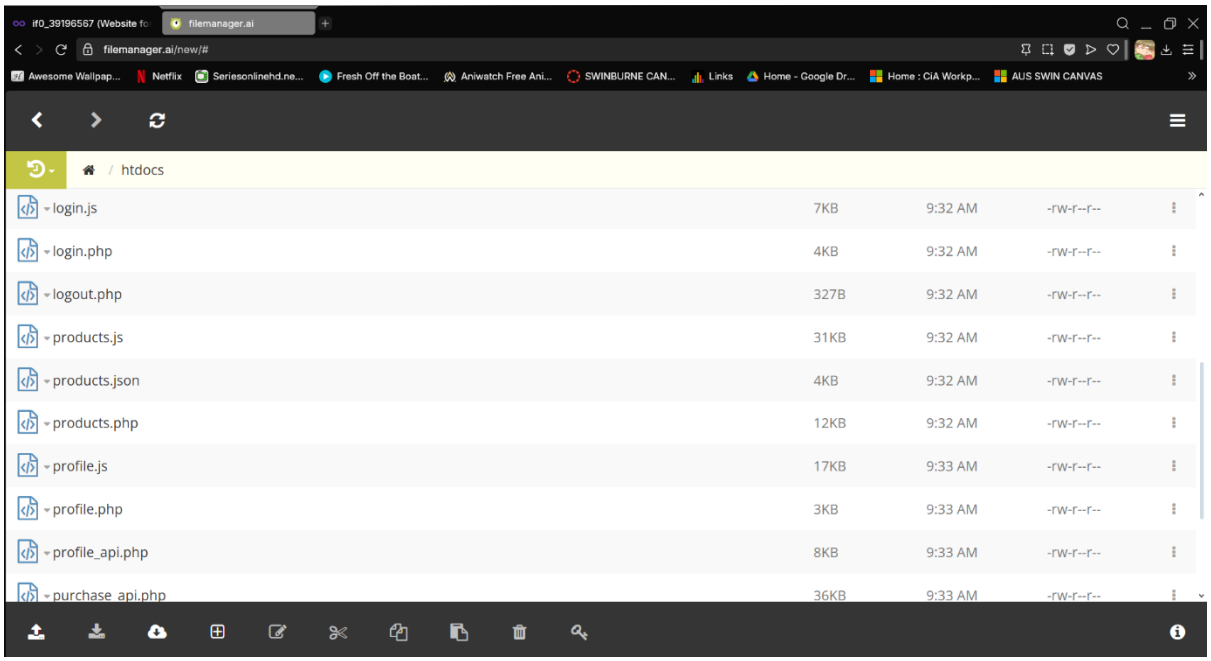




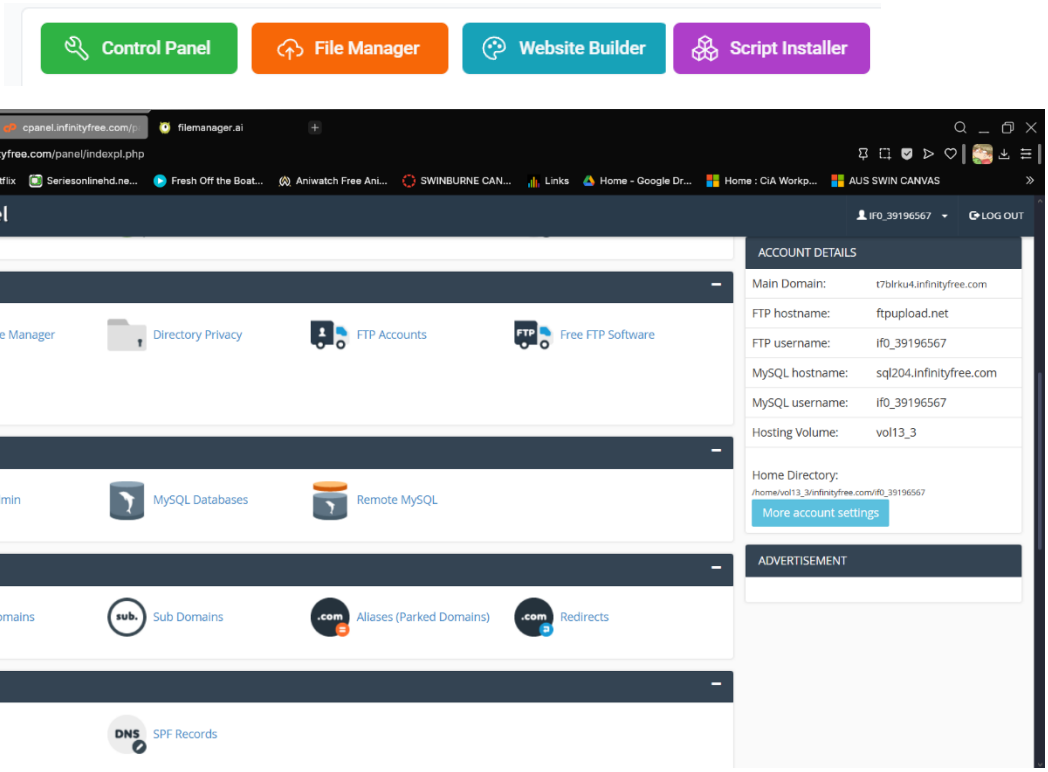
8) Then I placed all the files in the htdocs folder in the file manager.



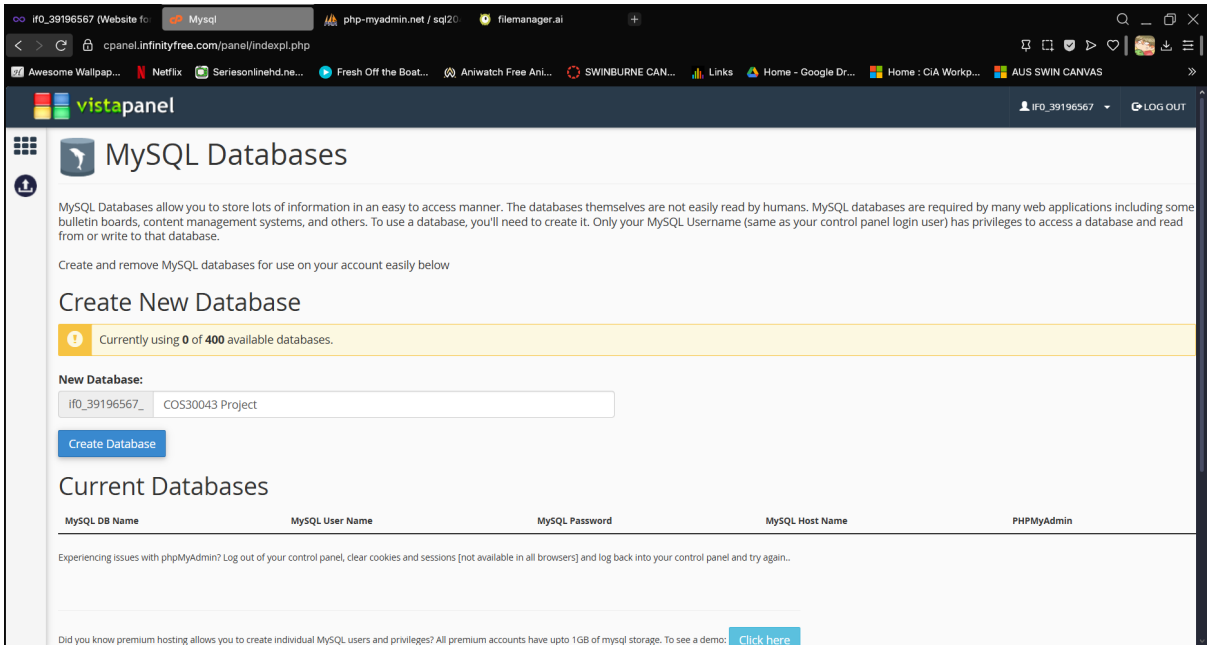




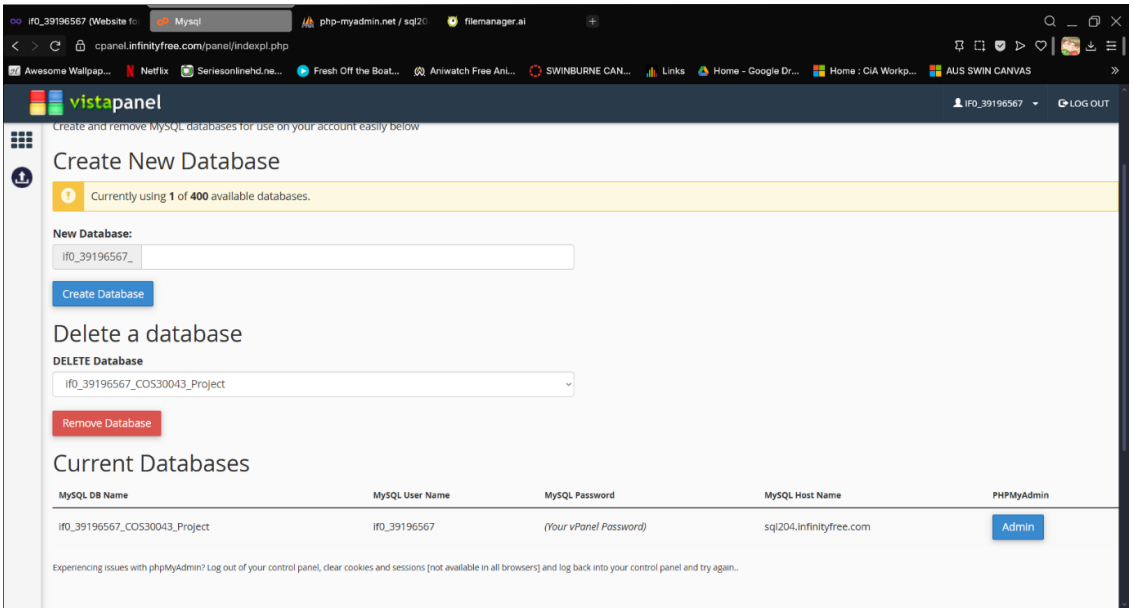
9) Next, I inserted created my database through the control panel. Under Databases, I opened ‘MySQL Databases’



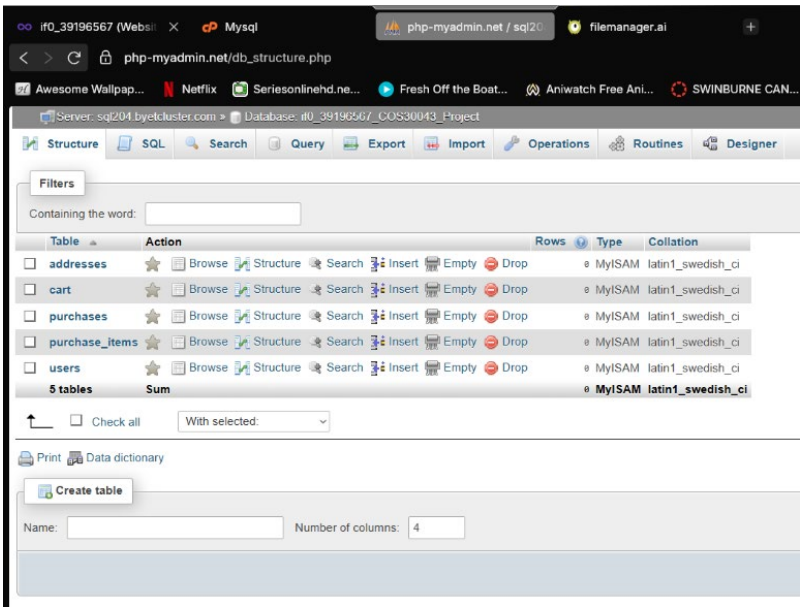
I named my database as ‘COS30043_Project’ and clicked on ‘Create Database’.



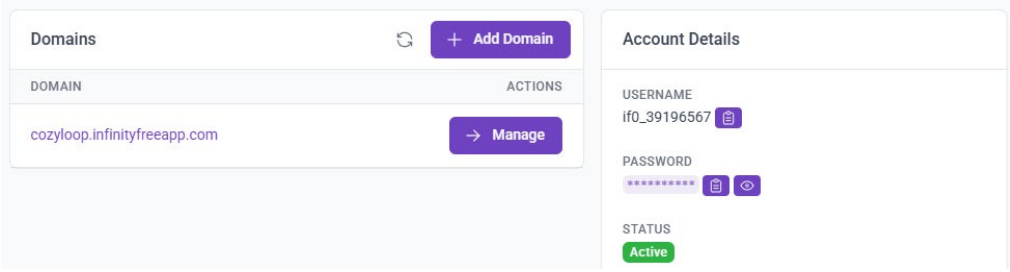
New Database has been successfully created.



10) I clicked on the 'Admin' button to open InfinityFree's phpMyAdmin page and imported all my SQL files into the database.



11) After all that is done, the website has been hosted online and is ready to be used through this link - cozyloop.infinityfreeapp.com. The link can be found on the Domain homepage.



6. Changes of Code Structure in Cart and Purchases Page (InfinityFree)

Why This Change Was Necessary

This change was likely made due to **hosting platform limitations**. Many free hosting platforms, including InfinityFree, have restrictions on HTTP methods:

1. **Limited HTTP Method Support**: Some hosting environments don't properly handle PATCH or DELETE requests, or they may be blocked by server configurations.
2. **Firewall/Proxy Issues**: Free hosting platforms often use proxies or firewalls that may strip or block certain HTTP methods.
3. **Consistency and Reliability**: Using POST for all operations ensures compatibility across different hosting environments.

a. cart_api.php & cart.js

cart_api.php:

1. Major Architectural Restructuring

The most significant change is how HTTP methods are handled:

Original Structure (localhost):

- **POST**: Add items to cart

```
52  try {
53      if ($method === 'POST') {
54          // Add item to cart
55          $input = json_decode(file_get_contents('php://input'), true);
```

- **GET**: Retrieve cart items

```
89      } elseif ($method === 'GET') {
90          // Get all cart items for the current user
91          $stmt = mysqli_prepare($conn, "SELECT * FROM cart WHERE user_id = ? ORDER BY id
            ASC");
```

- **DELETE**: Remove items (with different parameters for clearing all vs removing specific items)

```
118     } elseif ($method === 'DELETE') {
119         // Remove items from cart
120         $input = json_decode(file_get_contents('php://input'), true);
```

- **PATCH**: Update quantities (add_one/remove_one actions)

```
168     } elseif ($method === 'PATCH') {
169         // Update cart items
170         $input = json_decode(file_get_contents('php://input'), true);
```

New Structure (InfinityFree):

- **GET**: Retrieve cart items (unchanged)
- **POST**: Handles ALL cart operations using an action parameter

```php
} elseif ($method === 'POST') {
    // Handle different POST actions: add item, clear cart, remove all of a specific item, add one item, remove one item

    // Determine the action. If 'action' is not set, assume it's 'add_item' for backward compatibility.
    $action = $input['action'] ?? 'add_item';

    // Case 1: Add item to cart (initial add from product page or explicit 'add_item' action)
    if ($action === 'add_item' && isset($input['id'], $input['name'], $input['price'], $input['image'])) {
        // Validate input
        if (!is_numeric($input['price']) || $input['price'] < 0) {
            http_response_code(400);
            echo json_encode(['error' => 'Invalid price']);
            exit;
        }
    // Case 2: Clear entire cart
    elseif ($action === 'clear_cart') {
        $stmt = mysqli_prepare($conn, "DELETE FROM cart WHERE user_id = ?");
        if (!$stmt) {
            throw new Exception('Failed to prepare statement: ' . mysqli_error($conn));
        }
    // Case 3: Remove all instances of a specific product
    elseif ($action === 'remove_all_items' && isset($input['id'])) {
        $productId = intval($input['id']); // This ID is the product_id

        if (!$productId) {
            http_response_code(400);
            echo json_encode(['error' => 'Product ID is required for remove_all_items action']);
            exit;
        }
    // Case 4: Add one instance of a product
    elseif ($action === 'add_one_item' && isset($input['id'], $input['name'], $input['price'], $input['image'])) {
        $productId = $input['id'];
        // Validate input
        if (!is_numeric($input['price']) || $input['price'] < 0) {
            http_response_code(400);
            echo json_encode(['error' => 'Invalid price']);
            exit;
        }
    // Case 5: Remove one instance of a product
    elseif ($action === 'remove_one_item' && isset($input['id'])) {
        $productId = $input['id'];

        $stmt = mysqli_prepare($conn, "DELETE FROM cart WHERE user_id = ? AND product_id = ? LIMIT 1");
        if (!$stmt) {
            throw new Exception('Failed to prepare statement: ' . mysqli_error($conn));
        }
```

- **DELETE**: Deprecated (returns 405 Method Not Allowed)

```php
} elseif ($method === 'DELETE') {
    // All specific DELETE actions (clear_all, remove_all_items)
    // have been moved to the POST method with an 'action' parameter.
    // Therefore, any direct DELETE request at this point is unexpected.
    http_response_code(405); // Method Not Allowed for specific DELETE operations now.
    echo json_encode(['error' => 'DELETE method is not supported for specific cart operations. Please use POST with an action parameter.']);
```

- **PATCH**: Deprecated (returns 405 Method Not Allowed)

```php
} elseif ($method === 'PATCH') {
    // All PATCH actions are now deprecated in favor of POST with 'action' parameter.
    // The original 'remove_one' is now 'remove_one_item' in POST.
    // The original 'add_one' is now 'add_one_item' in POST.
    $input = json_decode(file_get_contents('php://input'), true); // Re-decode for safety
    if (isset($input['action'])) {
        http_response_code(405);
        echo json_encode(['error' => "PATCH method for action '{$input['action']}' is no longer supported. Please use POST instead."]);
    } else {
        http_response_code(405);
        echo json_encode(['error' => 'PATCH method is not supported for this endpoint. Please use POST with an action parameter.']);
    }
```

2. New POST Actions in Production Version

The production version consolidates all operations into POST requests with these actions:

- add_item: Add item to cart (default action for backward compatibility)
- clear_cart: Clear entire cart
- remove_all_items: Remove all instances of a specific product
- add_one_item: Add one instance of a product
- remove_one_item: Remove one instance of a product

3. Backward Compatibility

The production version maintains backward compatibility by defaulting to add_item action:

```
86          // Determine the action. If 'action' is not set, assume it's 'add_item' for backward compatibility.
87          $action = $input['action'] ?? 'add_item';
```

Summary

The restructuring from the local to production version was primarily driven by **hosting platform constraints**. Free hosting services often have limitations on HTTP methods, so consolidating all operations into POST requests with action parameters ensures the API works reliably across different hosting environments. This is a common pattern when moving from local development to shared hosting platforms.

cart.js:

Key Differences and Why They Were Made

1. HTTP Method Changes - All Operations Now Use POST (InfinityFree)

Original (localhost):

DELETE Methods:

```
82          removeAll(item) {
83              this.showConfirm(
84                  'Remove Item',
85                  `Are you sure you want to remove all ${item.name} from cart?`,
86                  () => {
87                      fetch('cart_api.php', {
88                          method: 'DELETE',
89                          headers: { 'Content-Type': 'application/json' },
90                          body: JSON.stringify({ id: item.id })
91                      })
```

```
154         clearCart() {
155             this.showConfirm(
156                 'Clear Cart',
157                 'Are you sure you want to clear the entire cart?',
158                 () => {
159                     fetch('cart_api.php', {
160                         method: 'DELETE',
161                         headers: { 'Content-Type': 'application/json' },
162                         body: JSON.stringify({ clear_all: true })
163                     })
```

PATCH Methods:

```
106         removeOne(item) {
107             fetch('cart_api.php', {
108                 method: 'PATCH',
109                 headers: { 'Content-Type': 'application/json' },
110                 body: JSON.stringify({ id: item.id, action: 'remove_one' })
111             })
```

```
124         addOne(item) {
125             fetch('cart_api.php', {
126                 method: 'PATCH',
127                 headers: { 'Content-Type': 'application/json' },
128                 body: JSON.stringify({
129                     id: item.id,
130                     action: 'add_one',
131                     name: item.name,
132                     price: item.price,
133                     image: item.image
134                 })
135             })
```

Production (InfinityFree):

DELETE Methods using POST instead:

```
82      removeAll(item) {
83          this.showConfirm(
84              'Remove Item',
85              `Are you sure you want to remove all ${item.name} from cart?`,
86              () => {
87                  // Changed method to POST and adjusted body to send 'action: remove_all_items'
88                  fetch('cart_api.php', {
89                      method: 'POST', // Changed to POST
90                      headers: { 'Content-Type': 'application/json' },
91                      body: JSON.stringify({ action: 'remove_all_items', id: item.id }) // Changed payload
92                  })
```

```
178     clearCart() {
179         this.showConfirm(
180             'Clear Cart',
181             'Are you sure you want to clear the entire cart?',
182             () => {
183                 // Changed method to POST and adjusted body to send 'action: clear_cart'
184                 fetch('cart_api.php', {
185                     method: 'POST', // Changed to POST
186                     headers: { 'Content-Type': 'application/json' },
187                     body: JSON.stringify({ action: 'clear_cart' }) // Changed payload
188                 })
```

PATCH Methods using POST instead:

```
114     removeOne(item) {
115         // Changed method to POST and adjusted body to send 'action: remove_one_item'
116         fetch('cart_api.php', {
117             method: 'POST', // Changed to POST
118             headers: { 'Content-Type': 'application/json' },
119             body: JSON.stringify({ action: 'remove_one_item', id: item.id }) // item.id here refers to product_id
120         })
```

```
140     addOne(item) {
141         // Changed method to POST and adjusted body to send 'action: add_one_item'
142         fetch('cart_api.php', {
143             method: 'POST', // Changed to POST
144             headers: { 'Content-Type': 'application/json' },
145             body: JSON.stringify({
146                 action: 'add_one_item', // New action name
147                 id: item.id, // item.id here refers to product_id
148                 name: item.name,
149                 price: item.price,
150                 image: item.image
151             })
152         })
```

2. Updated Action Names

The action names were updated to be more descriptive:

- remove_one → remove_one_item
- add_one → add_one_item
- clear_all: true → action: 'clear_cart'
- New action: remove_all_items

3. Enhanced Error Handling and User Feedback

Production version adds better response handling:

```
                .then((data) => {
                    if (data.success) {
                        this.showInfoMessage('Success', data.message || 'Cart cleared successfully!');
                        this.reloadCart();
                    } else {
                        this.showInfoMessage('Error', data.error || 'Failed to clear cart. Please try again.');
                    }
                })
```

This provides more detailed feedback to users about the success or failure of operations.

4. Simplified Checkout Process

Original version had complex checkout logic:

// Removed selected items from cart after checkout

```
                        // Remove only the selected items from cart
                        await this.removeSelectedItemsFromCart();
```

// Helper method to remove selected items

```
    // Helper method to remove selected items from cart after successful checkout
    async removeSelectedItemsFromCart() {
        const selectedItems = this.groupedItems.filter(item => item.selected);

        for (const item of selectedItems) {
            try {
                await fetch('cart_api.php', {
                    method: 'DELETE',
                    headers: { 'Content-Type': 'application/json' },
                    body: JSON.stringify({ id: item.id })
                });
            } catch (error) {
                console.error(`Failed to remove item ${item.name} from cart:`, error);
            }
        }
    },
```

Production version simplified this:

// Simplified - backend handles cart clearing after successful purchase

```
                const response = await fetch('purchase_api.php', {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json'
                    },
                    body: JSON.stringify({
                        action: 'create_from_cart',
                        selected_items: selectedItems // Send only selected items
                    })
                });
```

The complex cart item removal logic was removed because the backend now handles this automatically.

Summary

The changes from local to production were driven by **hosting platform constraints** and **improved architecture.** The frontend had to adapt to the backend's new POST-only API structure while also gaining better error handling and user feedback mechanisms. This is a common pattern when moving from local development environments to shared hosting platforms that may have HTTP method restrictions.

b. purchases_api.php & purchases.js

purchases_api.php:

1. Major Architectural Restructuring The most significant change is how HTTP methods are handled:

Original Structure (localhost):

- **GET**: Retrieve all purchases for the user

```
try {
    if ($method === 'GET') {
        // Get all purchases for the current user with their items
        $purchasesQuery = "
            SELECT p.*,
                   COUNT(pi.id) as item_count,
                   SUM(pi.quantity) as total_items
            FROM purchases p
            LEFT JOIN purchase_items pi ON p.id = pi.purchase_id
            WHERE p.user_id = ?
            GROUP BY p.id
            ORDER BY p.order_date DESC
        ";
```

- **POST**: Create new purchase from cart items

```
    } elseif ($method === 'POST') {
        // Create new purchase from selected cart items
        $input = json_decode(file_get_contents('php://input'), true);
```

- **PATCH**: Update purchase items or cancel orders (with action parameters)

```
    } elseif ($method === 'PATCH') {
        // Update purchase or purchase items (only for current user)
        $input = json_decode(file_get_contents('php://input'), true);
```

- **DELETE**: Remove specific items from orders or delete entire purchases (with action parameters)

```
    } elseif ($method === 'DELETE') {
        // Delete purchase items or entire purchase (only user's own)
        $input = json_decode(file_get_contents('php://input'), true);
```

New Structure (InfinityFree):

- **GET**: Retrieve all purchases for the user (unchanged)
- **POST**: Handles ALL purchase operations using an action parameter

```
} elseif ($method === 'POST') {
    // Handle different POST actions: create purchase, remove item, cancel order, update items

    if (!isset($input['action'])) {
        http_response_code(400);
        echo json_encode(['error' => 'Missing action parameter for POST request']);
        exit;
    }

    $action = $input['action'];

    if ($action === 'create_from_cart') {
        // Create new purchase from selected cart items for current user
        mysqli_begin_transaction($conn);
    // NEW CASE: Remove specific item from order (moved from DELETE method)
    elseif ($action === 'remove_item_from_order') {
        if (!isset($input['item_id']) || !isset($input['purchase_id'])) {
            http_response_code(400);
            echo json_encode(['error' => 'Missing item_id or purchase_id']);
            exit;
        }
```

```
365         // NEW CASE: Cancel an order (moved from PATCH method)
366         elseif ($action === 'cancel_purchase') {
367             if (!isset($input['purchase_id'])) {
368                 http_response_code(400);
369                 echo json_encode(['error' => 'Missing purchase_id']);
370                 exit;
371             }
411         // NEW CASE: Update items in order (moved from PATCH method)
412         elseif ($action === 'update_items_in_order') {
413             if (!isset($input['purchase_id'], $input['items'])) {
414                 http_response_code(400);
415                 echo json_encode(['error' => 'Missing purchase_id or items']);
416                 exit;
417             }
```

- **PATCH**: Deprecated (returns 405 Method Not Allowed with migration message)

```
540     } elseif ($method === 'PATCH') {
541         // All PATCH actions are now deprecated in favor of POST with 'action' parameter.
542         // The original 'update_items' is now 'update_items_in_order' in POST.
543         // The original 'cancel_order' is now 'cancel_purchase' in POST.
544         $input = json_decode(file_get_contents('php://input'), true); // Re-decode for safety
545         if (isset($input['action'])) {
546             http_response_code(405);
547             echo json_encode(['error' => "PATCH method for action '{$input['action']}' is no longer supported. Please
                use POST instead."]);
548         } else {
549             http_response_code(405);
550             echo json_encode(['error' => 'PATCH method is not supported for this endpoint. Please use POST with an
                action parameter.']);
551         }
```

- **DELETE**: Deprecated (returns 405 Method Not Allowed with migration message)

```
553     } elseif ($method === 'DELETE') {
554         // All DELETE actions are now deprecated in favor of POST with 'action' parameter.
555         // The original 'remove_item' is now 'remove_item_from_order' in POST.
556         // The original 'delete_purchase' is now handled elsewhere if needed (e.g., in POST).
557
558         http_response_code(405); // Method Not Allowed for specific DELETE operations now.
559         echo json_encode(['error' => 'DELETE method is not supported for specific purchase operations. Please use POST
            with an action parameter.']);
```

2. New POST Actions in Production Version The production version consolidates all operations into POST requests with these actions:

- **create_from_cart**: Create new purchase from selected cart items (enhanced with validation)
- **remove_item_from_order**: Remove specific item from a purchase order
- **cancel_purchase**: Cancel an entire purchase order
- **update_items_in_order**: Update quantities of items within an order

3. Enhanced Cart Item Processing The production version includes more sophisticated cart handling:

- **Selective cart clearing**: Only removes purchased items from cart instead of clearing entire cart
- **Precise quantity management**: Tracks and removes exact quantities purchased
- **Item-specific deletion**: Uses cart item IDs for more accurate removal

```php
                // Remove only the selected items from the cart
                foreach ($selectedItems as $item) {
                    $productId = $item['id']; // This is the product_id from the grouped item
                    $quantityToRemove = $item['quantity'];

                    // Fetch specific cart entry IDs to delete
                    $selectCartItemsStmt = mysqli_prepare($conn, "SELECT id FROM cart WHERE user_id = ? AND product_id =
                    ? LIMIT ?");
                    if (!$selectCartItemsStmt) {
                        throw new Exception('Failed to prepare select cart items statement: ' . mysqli_error($conn));
                    }
                    mysqli_stmt_bind_param($selectCartItemsStmt, "iii", $user_id, $productId, $quantityToRemove);
                    mysqli_stmt_execute($selectCartItemsStmt);
                    $cartItemIdsResult = mysqli_stmt_get_result($selectCartItemsStmt);

                    $idsToDelete = [];
                    while ($row = mysqli_fetch_assoc($cartItemIdsResult)) {
                        $idsToDelete[] = $row['id'];
                    }
                    mysqli_stmt_close($selectCartItemsStmt);

                    if (!empty($idsToDelete)) {
                        $idList = implode(',', $idsToDelete);
                        // Prepare and execute statement to delete specific cart items by ID
                        $deleteSpecificCartItemsStmt = mysqli_prepare($conn, "DELETE FROM cart WHERE id IN ($idList)");
                        if (!$deleteSpecificCartItemsStmt) {
                            throw new Exception('Failed to prepare delete specific cart items statement: ' . mysqli_error
                            ($conn));
                        }
                        if (!mysqli_stmt_execute($deleteSpecificCartItemsStmt)) {
                            throw new Exception('Failed to clear specific cart items: ' . mysqli_stmt_error(
                            $deleteSpecificCartItemsStmt));
                        }
                        mysqli_stmt_close($deleteSpecificCartItemsStmt);
                    }
                }
```

4. Backward Compatibility The production version maintains some backward compatibility:

- **Legacy POST format**: Still supports old cart-to-purchase conversion without action parameter
- **Graceful deprecation**: Provides clear error messages explaining the migration path from PATCH/DELETE to POST actions

Summary

The restructuring from the local to production version was primarily driven by **hosting platform constraints**. Free hosting services like InfinityFree often have limitations on HTTP methods like PATCH and DELETE, so consolidating all operations into POST requests with action parameters ensures the API works reliably. Additionally, the production version includes more robust error handling and security measures necessary for a live environment, making it more suitable for public hosting while maintaining the core functionality.

purchases.js:

1. HTTP Method Changes - All Operations Now Use POST (InfinityFree)

Original (localhost):

DELETE Methods:

```
185         removeItemFromOrder(purchaseId, itemId) {
186             this.showConfirm(
187                 'Remove Item',
188                 'Are you sure you want to remove this item from the order?',
189                 () => {
190                     fetch('purchase_api.php', {
191                         method: 'DELETE',
192                         headers: { 'Content-Type': 'application/json' },
193                         body: JSON.stringify({
194                             action: 'remove_item',
195                             item_id: itemId
196                         })
197                     })
198                         .then(res => {
199                             if (!res.ok) {
200                                 throw new Error(`HTTP error! status: ${res.status}`);
201                             }
202                             return res.json();
203                         })
204                         .then(() => {
205                             // After successful item removal, reload purchases.
206                             // Then, check the state of the purchase and exit edit mode if
                                it's empty.
207                             this.loadPurchases().then(() => {
208                                 const updatedPurchase = this.purchases.find(p => p.id ===
                                    purchaseId);
209                                 // If the purchase no longer has items (and implicitly
                                    cancelled by backend)
210                                 if (updatedPurchase && updatedPurchase.items.length === 0)
                                    {
211                                     this.editingOrder = null; // Exit edit mode
212                                 }
213                             });
214                         })
215                         .catch(err => {
216                             console.error('Failed to remove item:', err);
217                             this.showInfoMessage('Error', 'Failed to remove item. Please
                                try again.'); // Using custom message
218                         });
219                 }
220             );
221         },
```

PATCH Methods:

```
150         saveOrderChanges(orderId) {
151             const purchase = this.purchases.find(p => p.id === orderId);
152             if (!purchase) return;
153
154             const updates = purchase.items.map(item => ({
155                 id: item.id,
156                 quantity: item.newQuantity
157             }));
158
159             fetch('purchase_api.php', {
160                 method: 'PATCH',
161                 headers: { 'Content-Type': 'application/json' },
162                 body: JSON.stringify({
163                     action: 'update_items',
164                     purchase_id: orderId,
165                     items: updates
```

```
            })
        })
            .then(res => {
                if (!res.ok) {
                    throw new Error(`HTTP error! status: ${res.status}`);
                }
                return res.json();
            })
            .then(() => {
                this.editingOrder = null;
                this.loadPurchases();
                this.showInfoMessage('Success', 'Order updated successfully!'); //
                Using custom message
            })
            .catch(err => {
                console.error('Failed to update order:', err);
                this.showInfoMessage('Error', 'Failed to update order. Please try
                again.'); // Using custom message
            });
    },
```

```
    cancelOrder(orderId) {
        this.showConfirm(
            'Cancel Order',
            'Are you sure you want to cancel this order?',
            () => {
                fetch('purchase_api.php', {
                    method: 'PATCH',
                    headers: { 'Content-Type': 'application/json' },
                    body: JSON.stringify({
                        action: 'cancel_order',
                        purchase_id: orderId
                    })
                })
                    .then(res => {
                        if (!res.ok) {
                            throw new Error(`HTTP error! status: ${res.status}`);
                        }
                        return res.json();
                    })
                    .then(() => {
                        this.loadPurchases(); // Reload purchases to reflect status
                        change
                        this.showInfoMessage('Success', 'Order cancelled successfully.'
                        ); // Using custom message
                    })
                    .catch(err => {
                        console.error('Failed to cancel order:', err);
                        this.showInfoMessage('Error', 'Failed to cancel order. Please
                        try again.'); // Using custom message
                    });
            }
        );
    },
```

Production (InfinityFree):

DELETE Methods using POST instead:

```
        removeItemFromOrder(purchaseId, itemId) {
            this.showConfirm(
                'Remove Item',
                'Are you sure you want to remove this item from the order?',
                () => {
                    // Changed method to POST and adjusted body
                    fetch('purchase_api.php', {
                        method: 'POST', // Changed to POST
                        headers: { 'Content-Type': 'application/json' },
                        body: JSON.stringify({
                            action: 'remove_item_from_order', // New action name
                            item_id: itemId,
                            purchase_id: purchaseId // Added purchase_id for backend context
                        })
                    })
                        .then(res => {
                            if (!res.ok) {
                                throw new Error(`HTTP error! status: ${res.status}`);
                            }
                            return res.json();
                        })
                        .then((data) => {
                            if (data.success) {
                                this.showInfoMessage('Success', data.message || 'Item removed successfully!');
                                // After successful item removal, reload purchases.
                                // Then, check the state of the purchase and exit edit mode if it's empty.
                                this.loadPurchases().then(() => {
                                    const updatedPurchase = this.purchases.find(p => p.id === purchaseId);
                                    // If the purchase no longer has items (and implicitly cancelled by backend)
                                    if (updatedPurchase && updatedPurchase.items.length === 0) {
                                        this.editingOrder = null; // Exit edit mode
                                    }
                                });
                            } else {
                                this.showInfoMessage('Error', data.error || 'Failed to remove item. Please try again.');
                            }
                        })
                        .catch(err => {
                            console.error('Failed to remove item:', err);
                            this.showInfoMessage('Error', 'An unexpected error occurred while removing item. Please try
                            again.'); // Using custom message
                        });
                }
            );
        },
```

PATCH Methods using POST instead:

```
        saveOrderChanges(orderId) {
            const purchase = this.purchases.find(p => p.id === orderId);
            if (!purchase) return;

            const updates = purchase.items.map(item => ({
                id: item.id,
                quantity: item.newQuantity
            }));

            // Changed method to POST and adjusted body to send 'action: update_items_in_order'
            fetch('purchase_api.php', {
                method: 'POST', // Changed to POST
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({
                    action: 'update_items_in_order', // New action name
                    purchase_id: orderId,
                    items: updates
                })
            })
                .then(res => {
                    if (!res.ok) {
                        throw new Error(`HTTP error! status: ${res.status}`);
                    }
                    return res.json();
                })
                .then((data) => {
                    if (data.success) {
                        this.editingOrder = null;
                        this.loadPurchases();
                        this.showInfoMessage('Success', data.message || 'Order updated successfully!');
```

```
                } else {
                    this.showInfoMessage('Error', data.error || 'Failed to update order. Please try again.');
                }
            })
            .catch(err => {
                console.error('Failed to update order:', err);
                this.showInfoMessage('Error', 'An unexpected error occurred while updating order. Please try again.'
                );
            });
    },
```

2. Updated Action Names The action names were updated to be more descriptive:

- update_items → update_items_in_order
- remove_item → remove_item_from_order
- cancel_order → cancel_purchase

3. Enhanced Error Handling and User Feedback Production version adds better response handling:

```
            .then((data) => {
                if (data.success) {
                    this.editingOrder = null;
                    this.loadPurchases();
                    this.showInfoMessage('Success', data.message || 'Order updated successfully!');
                } else {
                    this.showInfoMessage('Error', data.error || 'Failed to update order. Please try again.');
                }
            })
            .catch(err => {
                console.error('Failed to update order:', err);
                this.showInfoMessage('Error', 'An unexpected error occurred while updating order. Please try again.'
                );
            });
```

This provides more detailed feedback to users about the success or failure of operations, utilizing server response messages rather than generic client-side messages.

4. Additional Context in API Calls Original version sent minimal data:

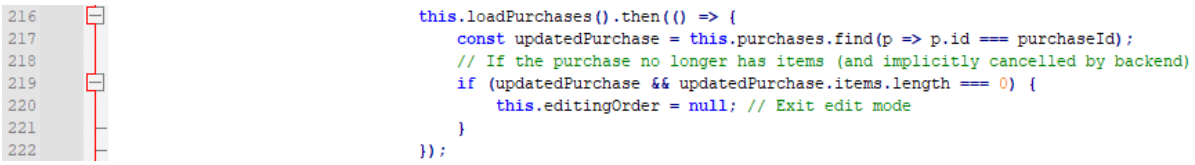
Original (localhost):

```
body: JSON.stringify({
    action: 'remove_item',
    item_id: itemId
})
```

Production (InfinityFree):

```
body: JSON.stringify({
    action: 'remove_item_from_order', // New action name
    item_id: itemId,
    purchase_id: purchaseId // Added purchase_id for backend context
})
```

5. Improved State Management After Operations Production version has better post-operation handling:



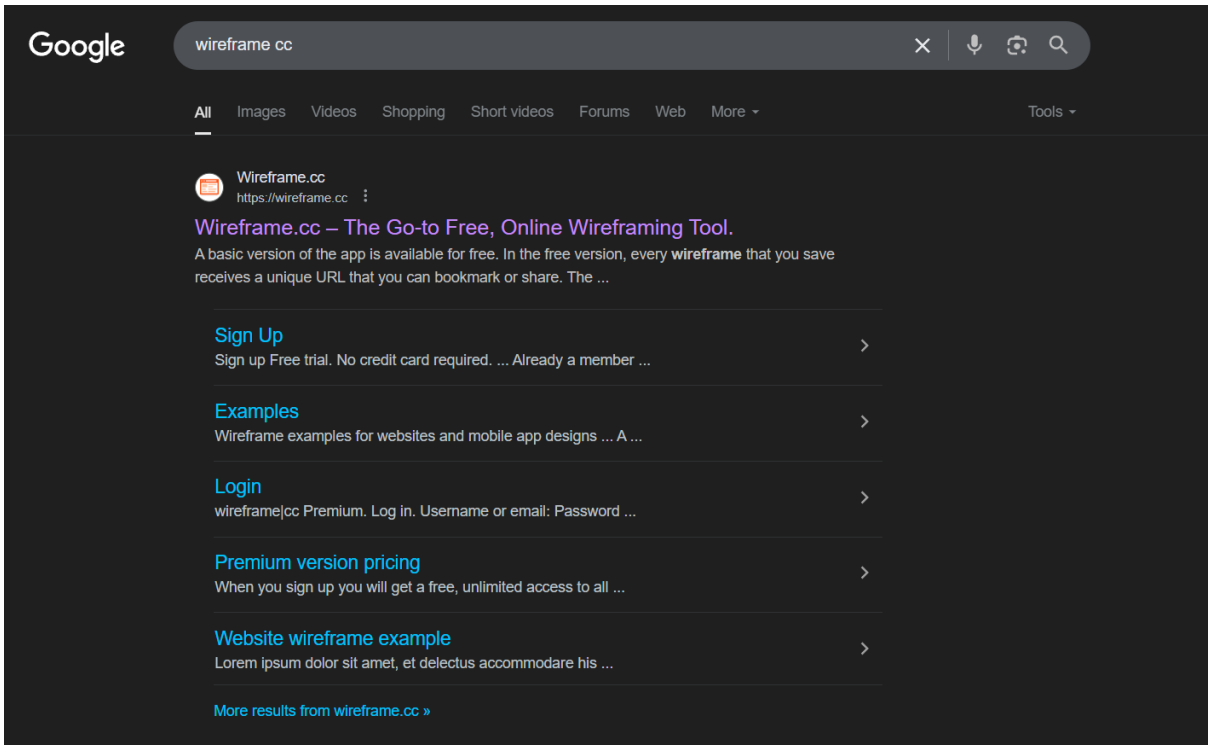
```
this.loadPurchases().then(() => {
    const updatedPurchase = this.purchases.find(p => p.id === purchaseId);
    // If the purchase no longer has items (and implicitly cancelled by backend)
    if (updatedPurchase && updatedPurchase.items.length === 0) {
        this.editingOrder = null; // Exit edit mode
    }
});
```

This ensures the UI properly updates when orders become empty after item removal.

Summary

The changes from local to production were driven by **hosting platform constraints** and **improved error handling**. InfinityFree's shared hosting environment required switching from RESTful HTTP methods (PATCH, DELETE) to POST-only operations, while also necessitating more robust error handling and detailed server communication. The production version provides better user feedback through server-generated messages and includes additional context in API calls for more reliable backend processing.

7. Design Concepts



Link to Wireframe.cc - https://wireframe.cc/

a. Login & Registration Page

Overview

The authentication system uses a clean, split-screen layout that separates functional forms from promotional content, creating an intuitive and conversion-focused user experience.

Key Design Elements

Layout Structure

- **Split-screen approach**: Forms on white background (left), promotional content on colored backgrounds (right)
- **Clear separation** between primary actions and secondary messaging
- **Responsive design** that adapts across devices while maintaining core layout

Visual Design

- **Minimalist interface** reduces cognitive load
- **Color palette**: White forms, rose/pink promotional panels, dark brown action buttons
- **Clean typography** with clear hierarchy and readable labels
- **Modern form fields** with rounded corners and password visibility toggles

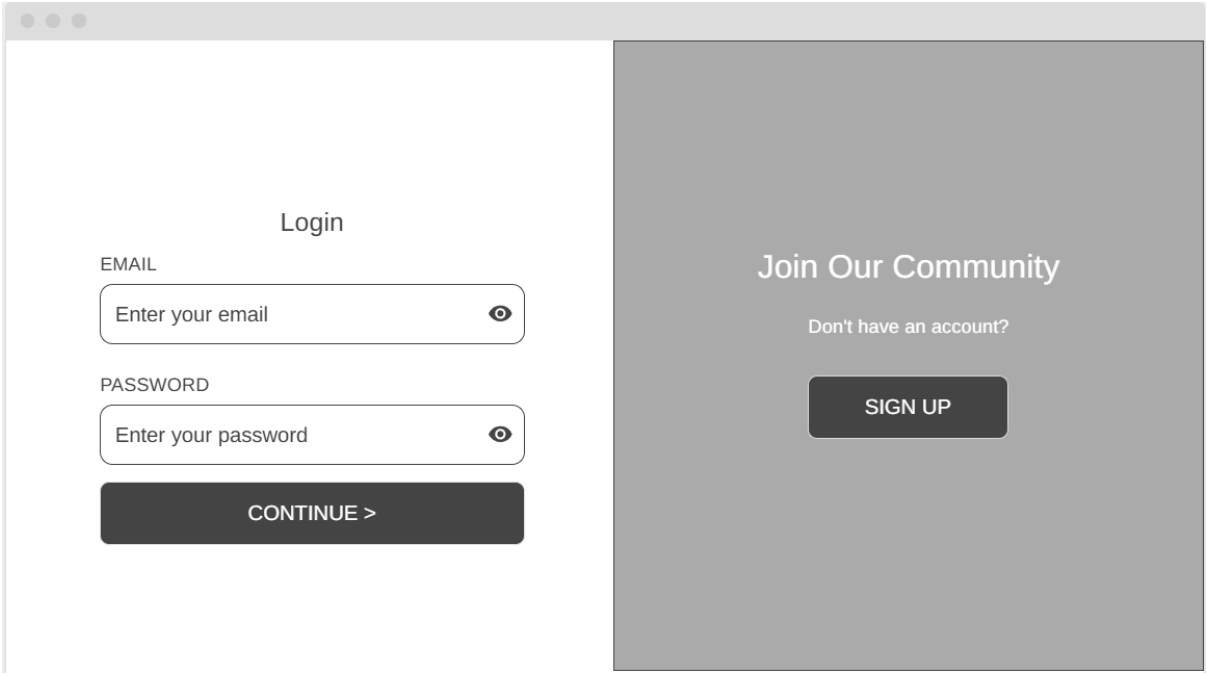
User Experience

- **Progressive disclosure**: Login shows minimal fields, registration adds necessary information
- **Conversion optimization**: Promotional panels drive user engagement without cluttering forms
- **Accessibility**: High contrast, adequate touch targets, logical navigation flow

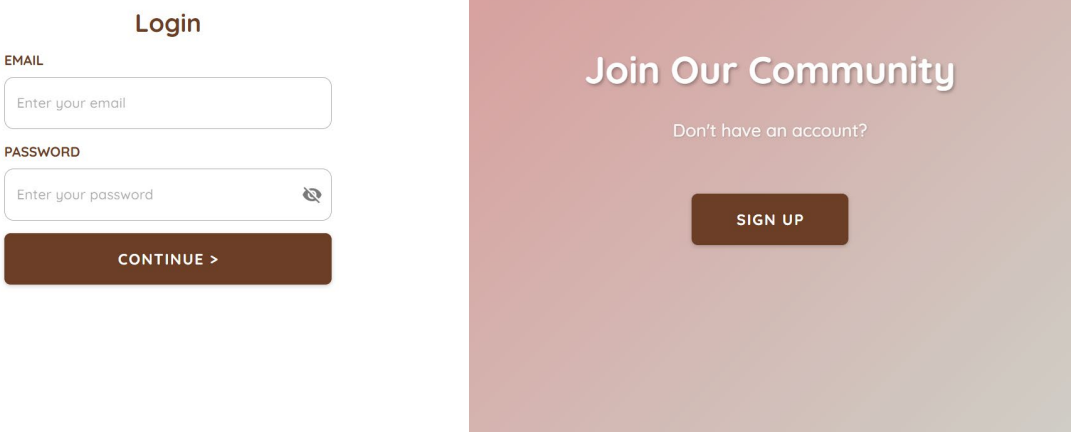
Design Goals

The design prioritizes user trust and ease of use through clean aesthetics, consistent branding, and reduced friction in the authentication process. The community-focused messaging ("Join Our Community") reinforces the platform's social nature while maintaining professional visual standards.

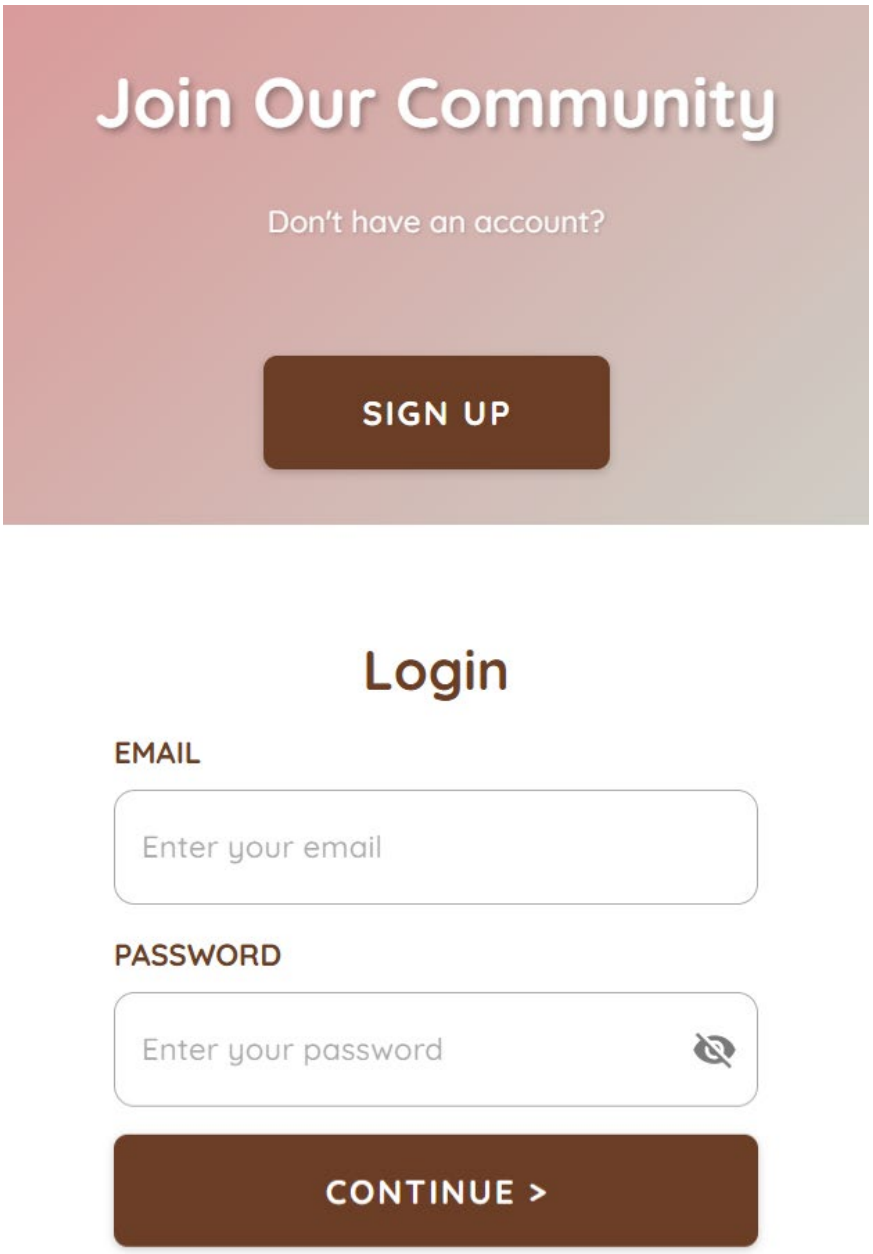
Login Wireframe:



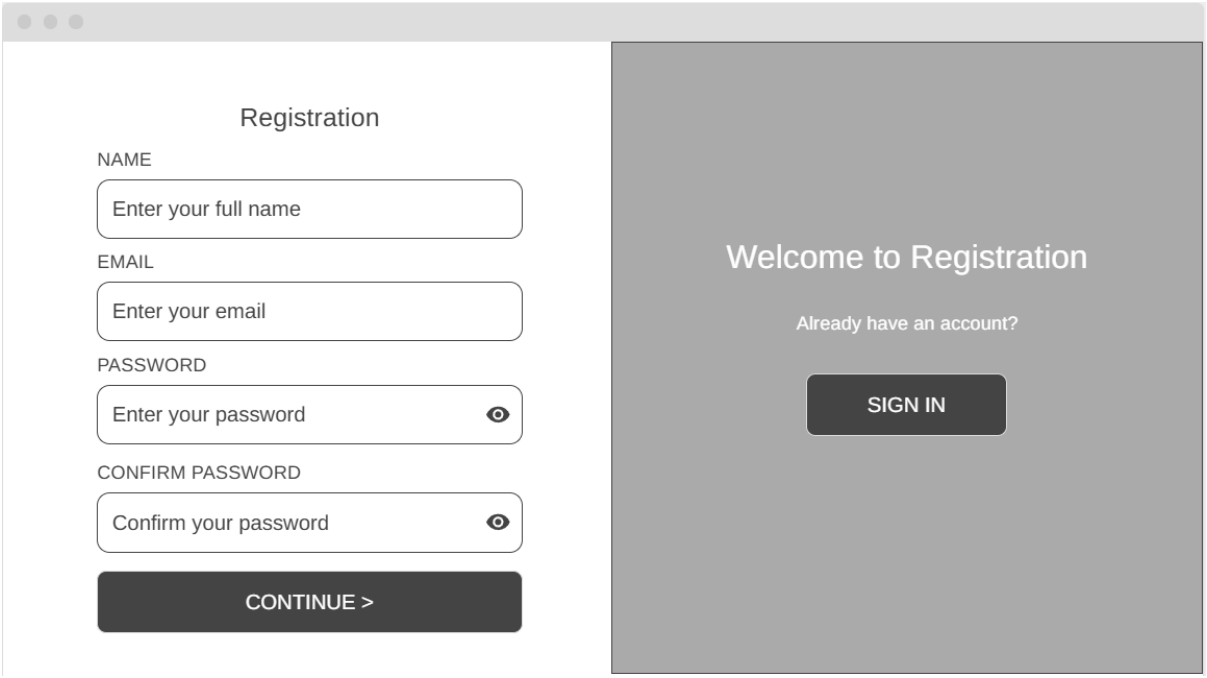
Login Screenshot (Fullscreen):



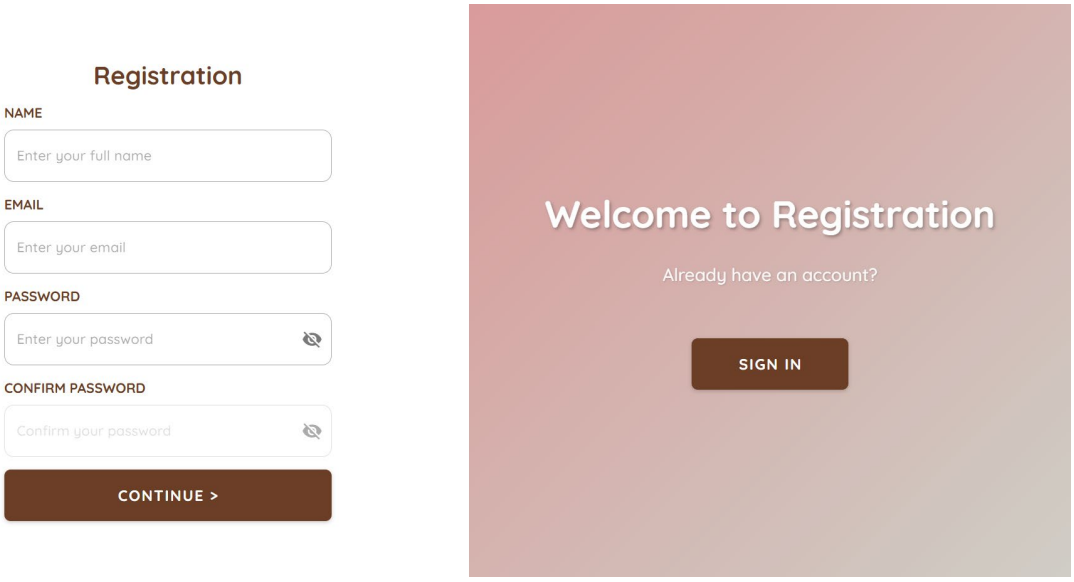
Login Screenshot (Minimized Screen - Responsive):



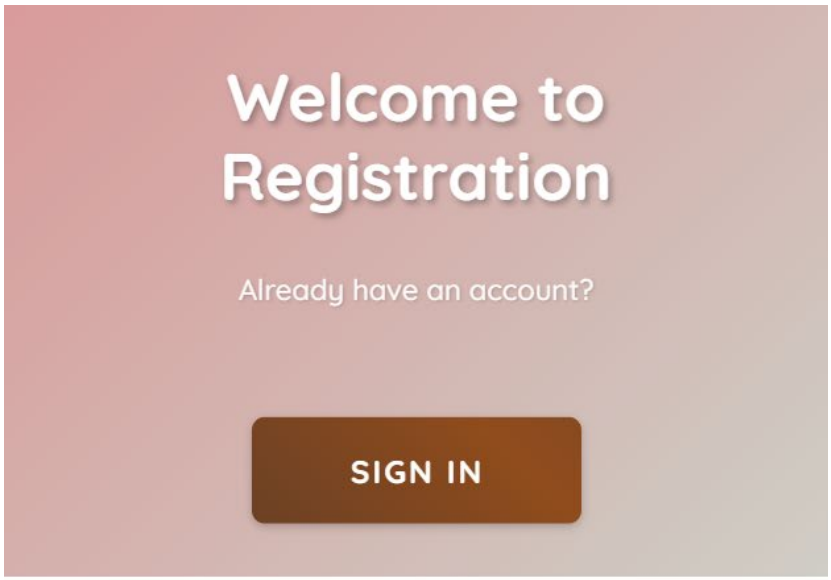
Registration Wireframe:

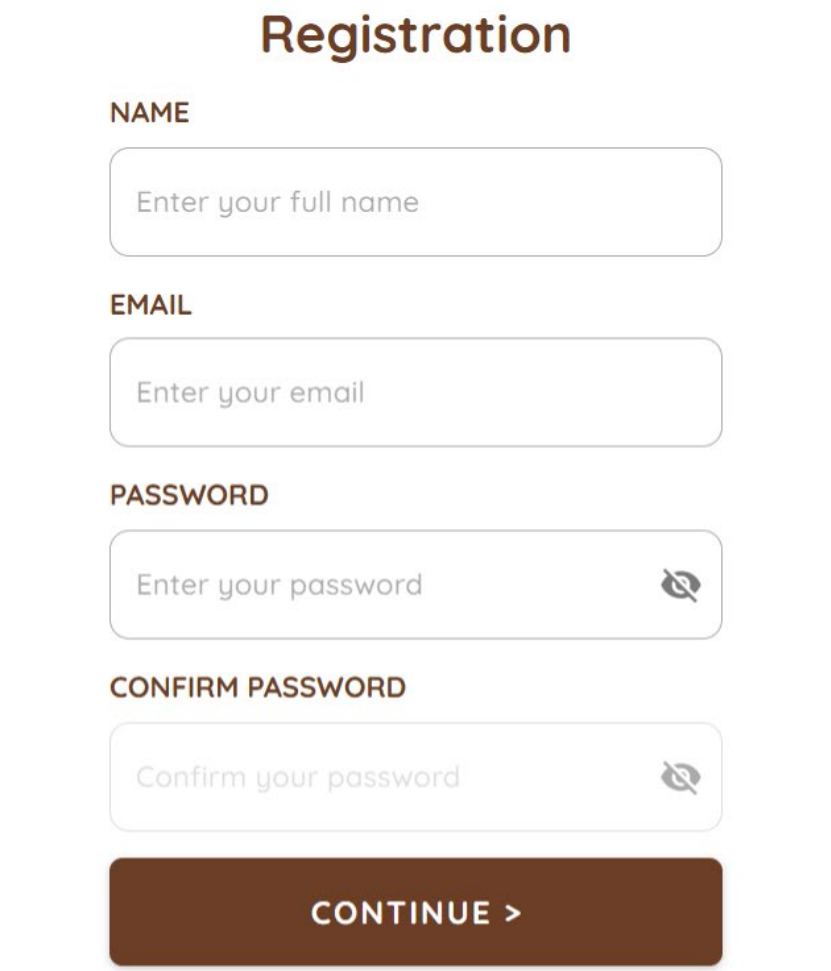


Registration Screenshot (Fullscreen):



Registration Screenshot (Minimized Screen - Responsive):





b. Index Page

Overview

The index page employs a warm, community-focused design that showcases handmade crochet products through a visually rich and emotionally engaging interface. The design balances product discovery with brand storytelling, creating an inviting marketplace experience.

Key Design Elements

Layout Structure

- **Hero section**: Vibrant crochet flower banner with overlaid branding creates immediate visual impact
- **Categorized navigation**: Clean circular category icons with intuitive labels for easy browsing
- **Product showcase**: Grid-based layout featuring curated collections (Best Sellers, New Arrivals)
- **Hierarchical organization**: Clear content sections with consistent spacing and visual flow

Visual Design

- **Colour palette**: Soft pink background with colourful crochet imagery, creating warmth and femininity
- **Typography**: Clean, modern fonts with "Handmade Crochet with Love" as the central brand message
- **Photography**: High-quality product images with consistent styling and green "NEW" badges for freshness
- **Texture**: Hero banner showcases tactile crochet flowers, emphasizing the handmade quality

User Experience

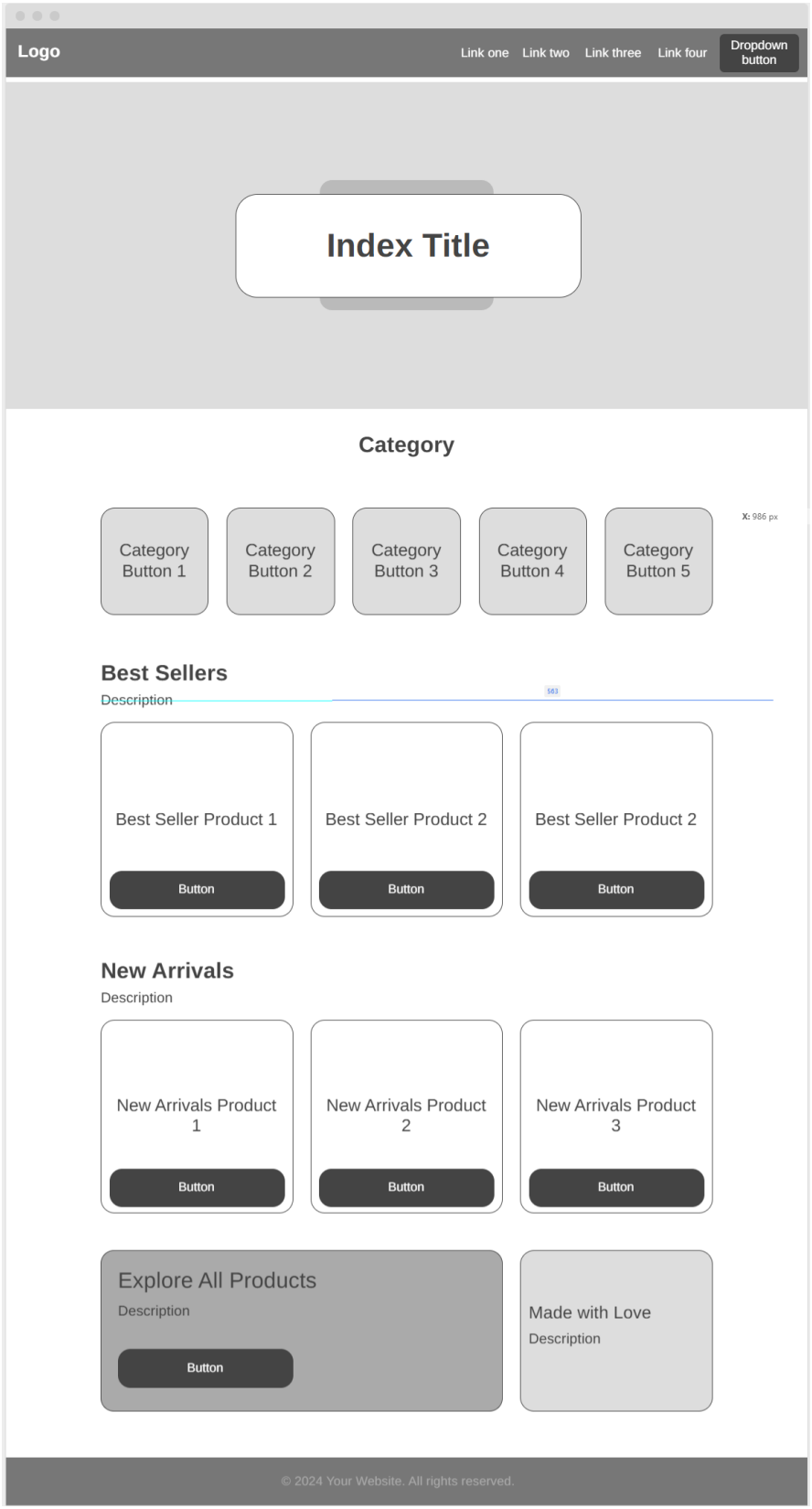
- **Intuitive navigation**: Category-based shopping with clear visual icons (Accessories, Clothing, Home Decor, etc.)
- **Product discovery**: Featured sections highlight popular items and new arrivals with pricing and quick actions
- **Call-to-action optimization**: Green buttons and "View Details" links guide user engagement
- **Trust building**: "Made with Love" messaging reinforces artisan quality and personal touch

Design Goals

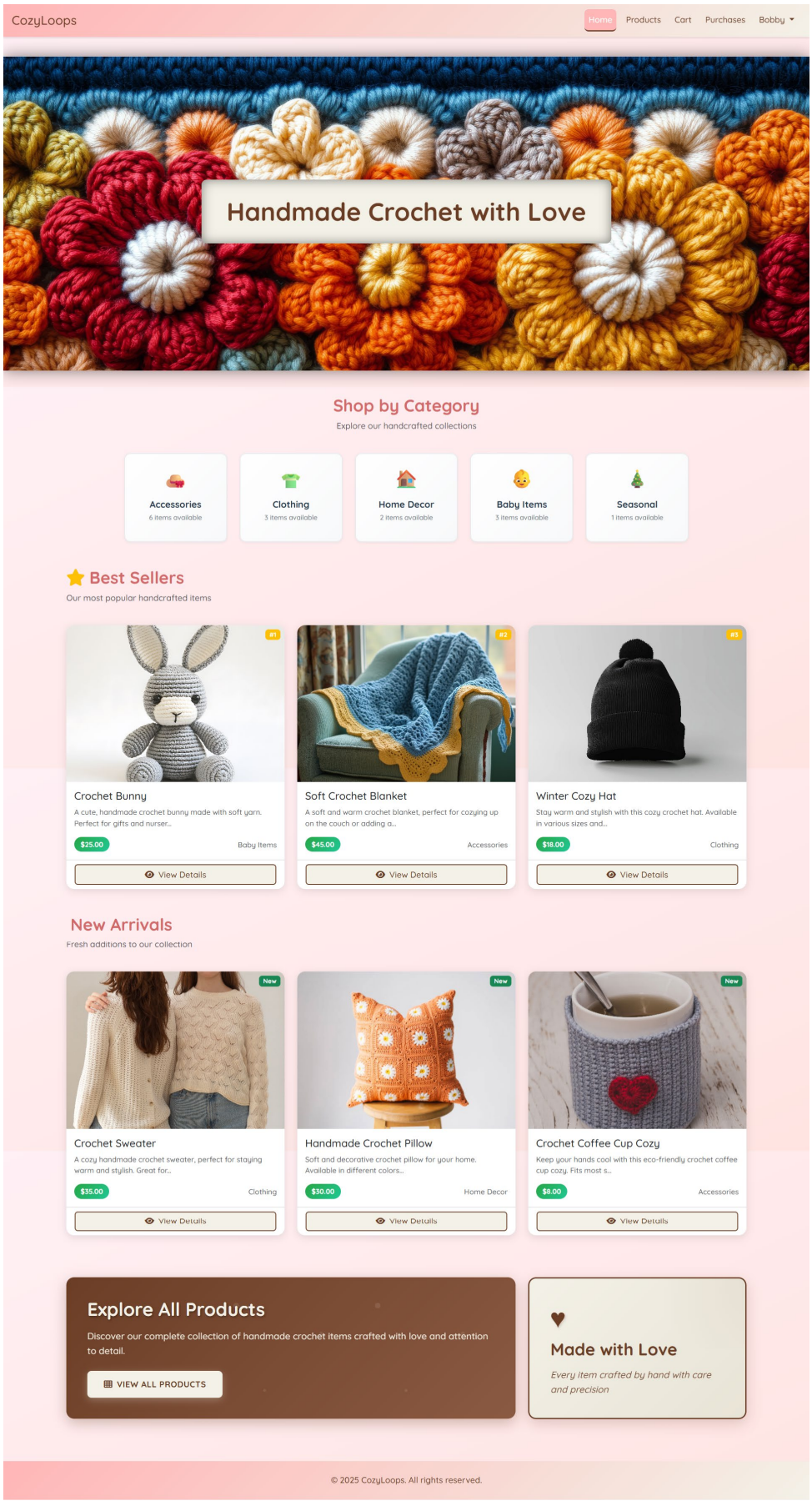
The design cultivates a sense of community and craftsmanship through warm colours, personal messaging, and high-quality product presentation. The layout prioritizes easy browsing while

maintaining the handmade aesthetic that differentiates the brand from mass-market competitors, creating an emotional connection between customers and artisan creators.

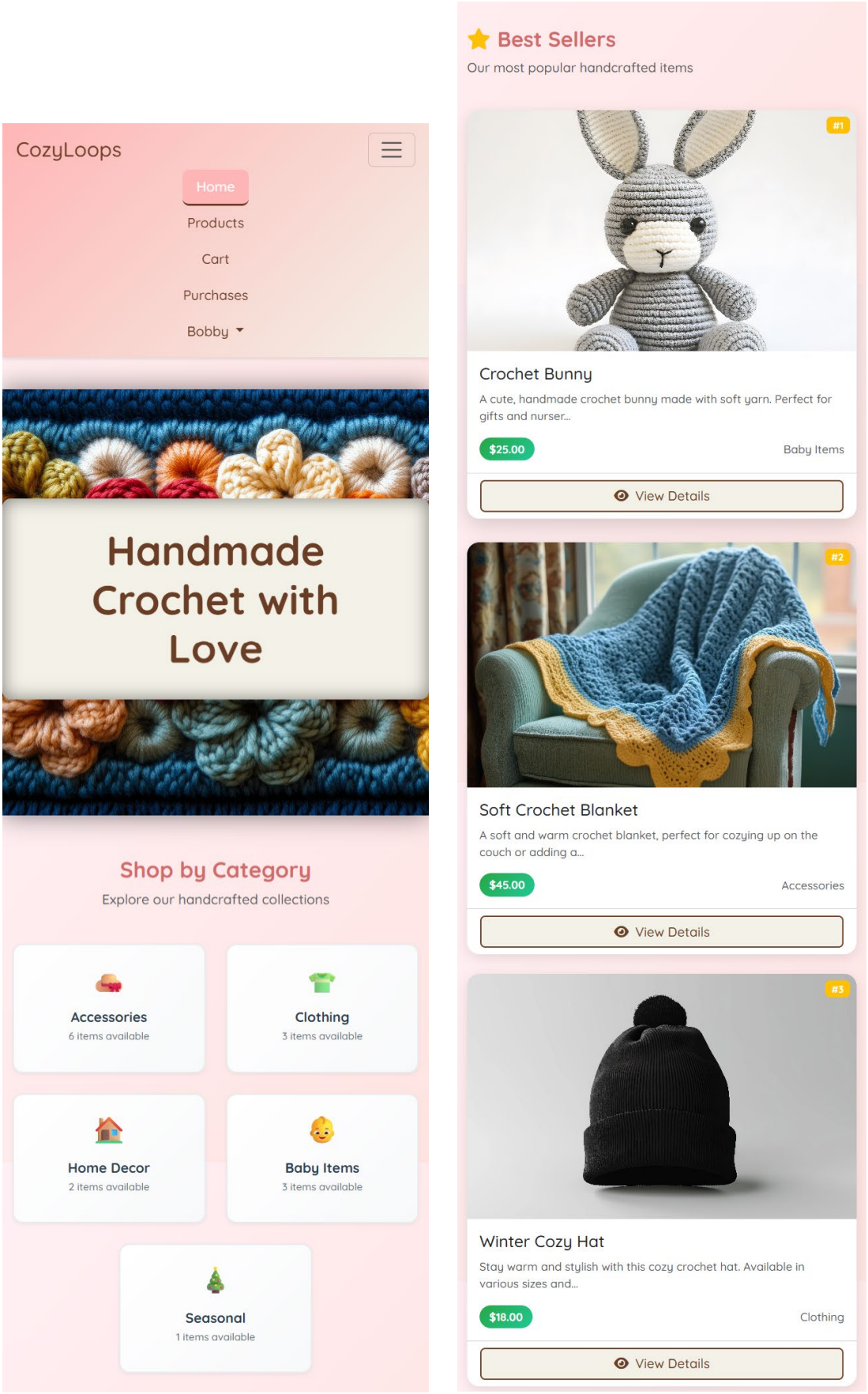
Index Wireframe:



Index Screenshot (Fullscreen):



Index Screenshot (Minimized Screen - Responsive):



New Arrivals

Fresh additions to our collection

Crochet Sweater

A cozy handmade crochet sweater, perfect for staying warm and stylish. Great for...

$35.00 Clothing

View Details

Handmade Crochet Pillow

Soft and decorative crochet pillow for your home. Available in different colors...

$30.00 Home Decor

View Details

Crochet Coffee Cup Cozy

Keep your hands cool with this eco-friendly crochet coffee cup cozy. Fits most s...

$8.00 Accessories

View Details

Explore All Products

Discover our complete collection of handmade crochet items crafted with love and attention to detail.

VIEW ALL PRODUCTS

♥

Made with Love

Every item crafted by hand with care and precision

c. Products Page

Overview

The product showcase page utilizes a comprehensive filtering and discovery system that transforms product browsing into an engaging, efficient experience. The design balances robust functionality with visual appeal, enabling customers to easily navigate through the handmade crochet collection while maintaining the brand's warm, artisanal aesthetic.

Key Design Elements

Layout Structure

- **Hierarchical navigation**: Category-based filtering with numbered badges showing available items per category
- **Advanced filtering bar**: Integrated search, price range, sorting, and view mode controls in a cohesive interface
- **Grid-based product display**: Consistent 3-column layout optimizing product visibility and comparison
- **Pagination system**: Clean navigation with page indicators and "Jump to page" functionality

Visual Design

- **Colour consistency**: Maintains pink brand palette with brown accent buttons for product actions
- **Product photography**: High-quality, well-lit images showcasing texture and craftsmanship details
- **Typography hierarchy**: Clear product names, descriptions, and pricing with consistent formatting
- **Visual indicators**: Green pricing prominently displayed for quick scanning

User Experience

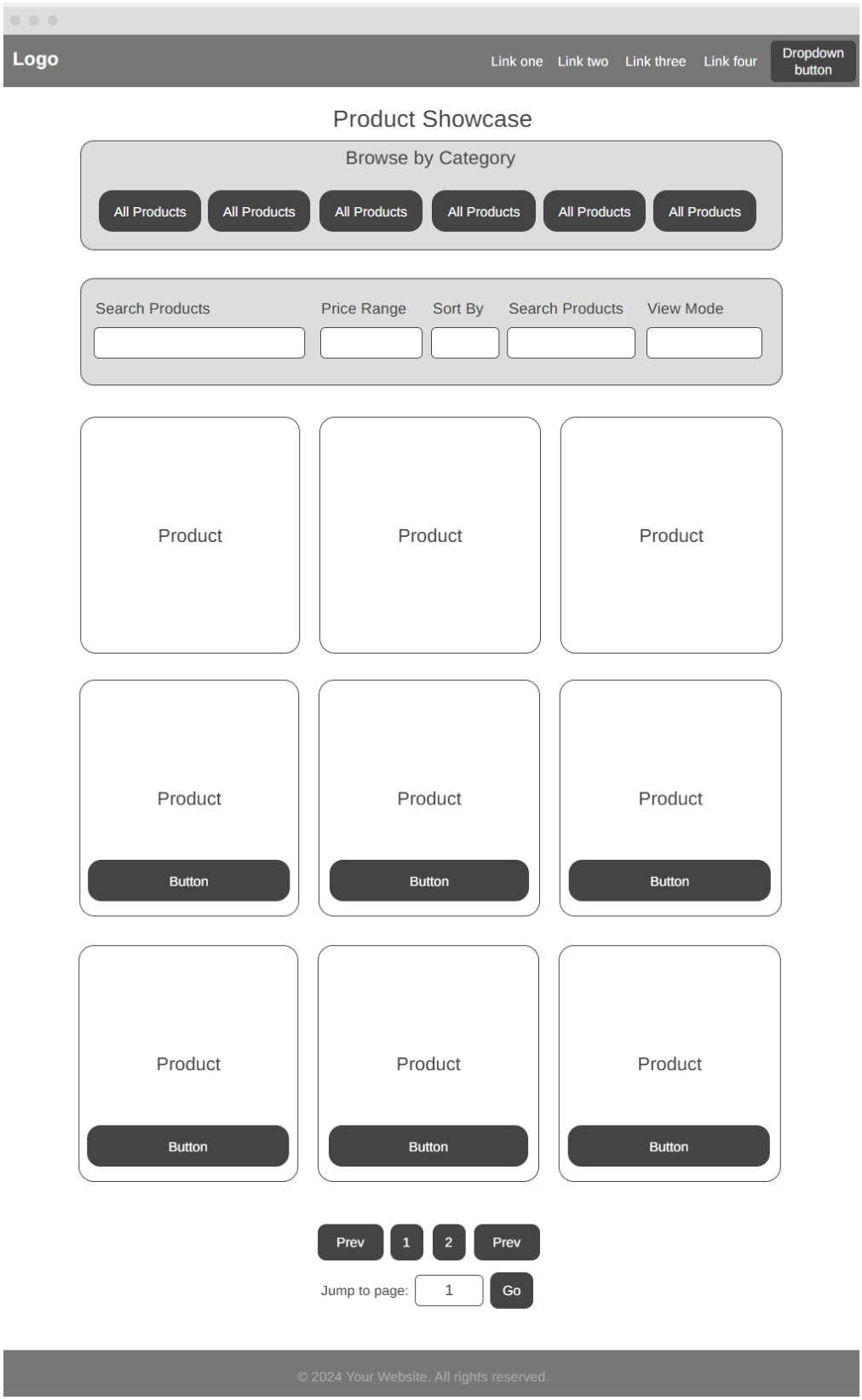
- **Multi-faceted filtering**: Category pills, search functionality, price ranges, and sorting options provide comprehensive control
- **Product information density**: Balanced display showing essential details without overwhelming users
- **Interaction feedback**: Hover states, active filters, and clear call-to-action buttons guide user behaviour
- **Responsive grid**: View mode toggles allow users to customize their browsing experience

Design Goals

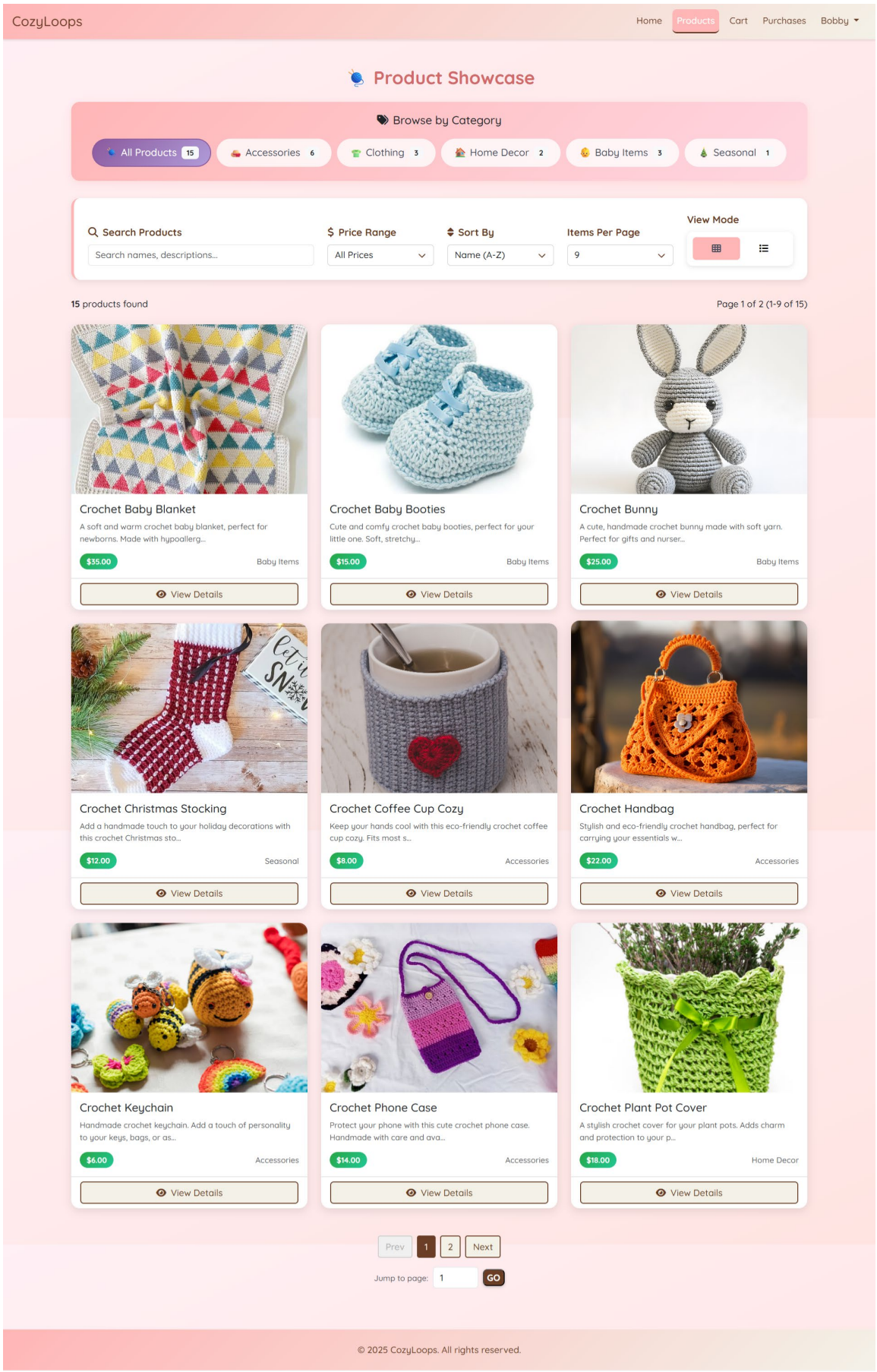
The design prioritizes product discoverability and user control while maintaining the handcrafted brand identity. The comprehensive filtering system empowers customers to find specific items efficiently, while the clean grid layout showcases the quality and variety of

handmade products, encouraging exploration and increasing conversion potential through improved user experience.

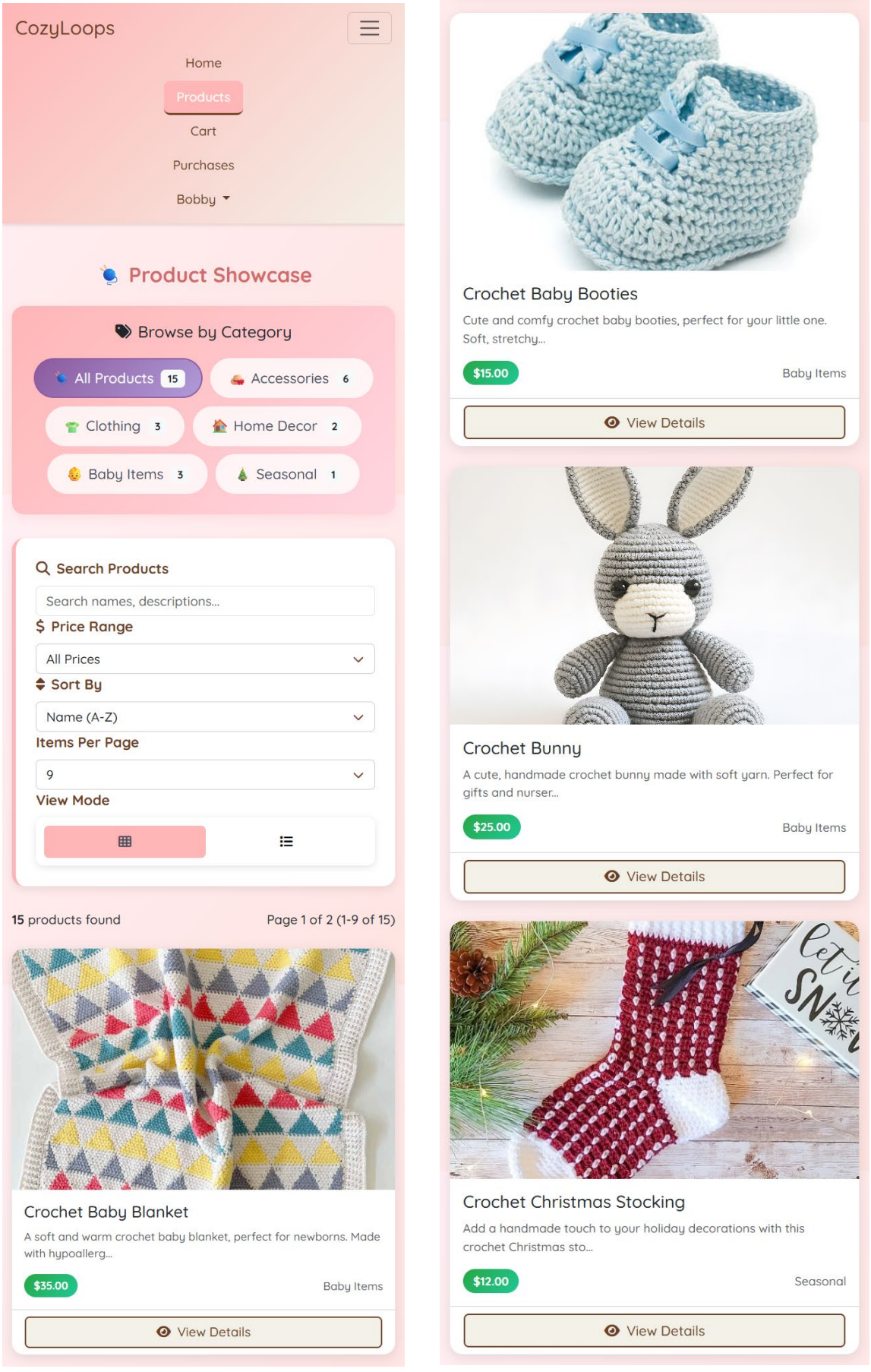
Products Wireframe:



Products Screenshot (Fullscreen):



Products Screenshot (Minimized Screen - Responsive):



Crochet Coffee Cup Cozy

Keep your hands cool with this eco-friendly crochet coffee cup cozy. Fits most s...

$8.00 Accessories

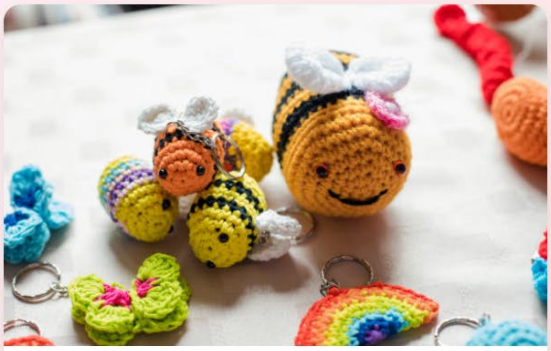
View Details

Crochet Handbag

Stylish and eco-friendly crochet handbag, perfect for carrying your essentials w...

$22.00 Accessories

View Details



Crochet Keychain

Handmade crochet keychain. Add a touch of personality to your keys, bags, or as...

$6.00 Accessories

View Details

Crochet Phone Case

Protect your phone with this cute crochet phone case. Handmade with care and ava...

$14.00 Accessories

View Details

Crochet Plant Pot Cover

A stylish crochet cover for your plant pots. Adds charm and protection to your p...

$18.00 Home Decor

View Details

Prev 1 2 Next

Jump to page: 1 GO

d. Cart Page

Overview

The cart page employs a dual-state design approach that transforms from an inviting empty state to a comprehensive shopping management interface. The design emphasizes user engagement through playful messaging and efficient cart management functionality.

Key Design Elements

Layout Structure

- **Adaptive interface**: Empty state features centred messaging with ample whitespace, populated state utilizes full-width card layout
- **Progressive enhancement**: Free shipping incentive bar appears when items are added, encouraging higher order values
- **Organized product display**: Individual item cards with product images, details, and quantity controls for easy management
- **Clear summary section**: Distinct order summary with transparent pricing breakdown and prominent checkout action

Visual Design

- **Colour palette**: Consistent soft pink background with white content cards and yellow or green accent for shipping promotions
- **Typography**: Playful "Cozy Cart" branding with friendly messaging ("Start shopping to add some cozy items!")
- **Visual hierarchy**: Product images prominently displayed with clear price information and total highlighting
- **Interactive elements**: Checkboxes for bulk selection, quantity spinners, delete buttons for item management, clear cart button and a proceed to checkout button

User Experience

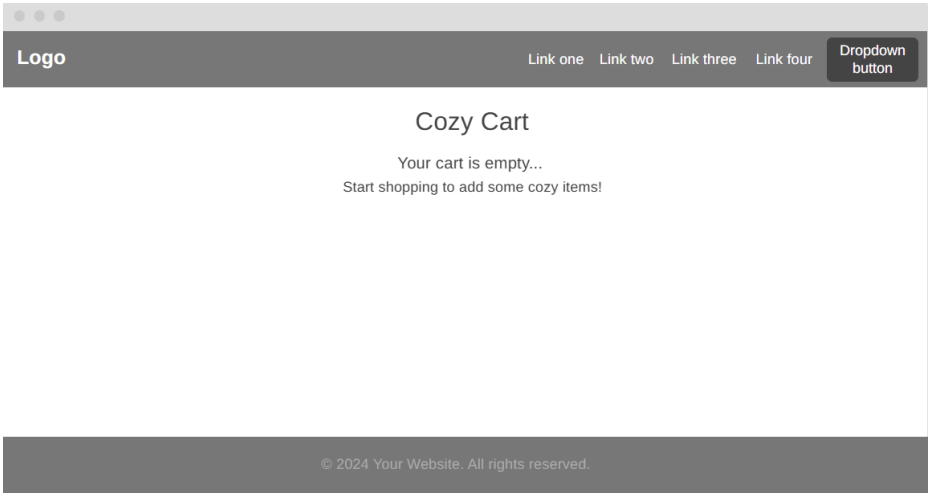
- **Emotional engagement**: Empty cart messaging maintains brand personality while encouraging action
- **Bulk operations**: "Select all items" functionality with clear cart management options
- **Purchase motivation**: Free shipping threshold prominently displayed to increase average order value
- **Conversion optimization**: Large "PROCEED TO CHECKOUT" button maintains focus on purchase completion

Design Goals

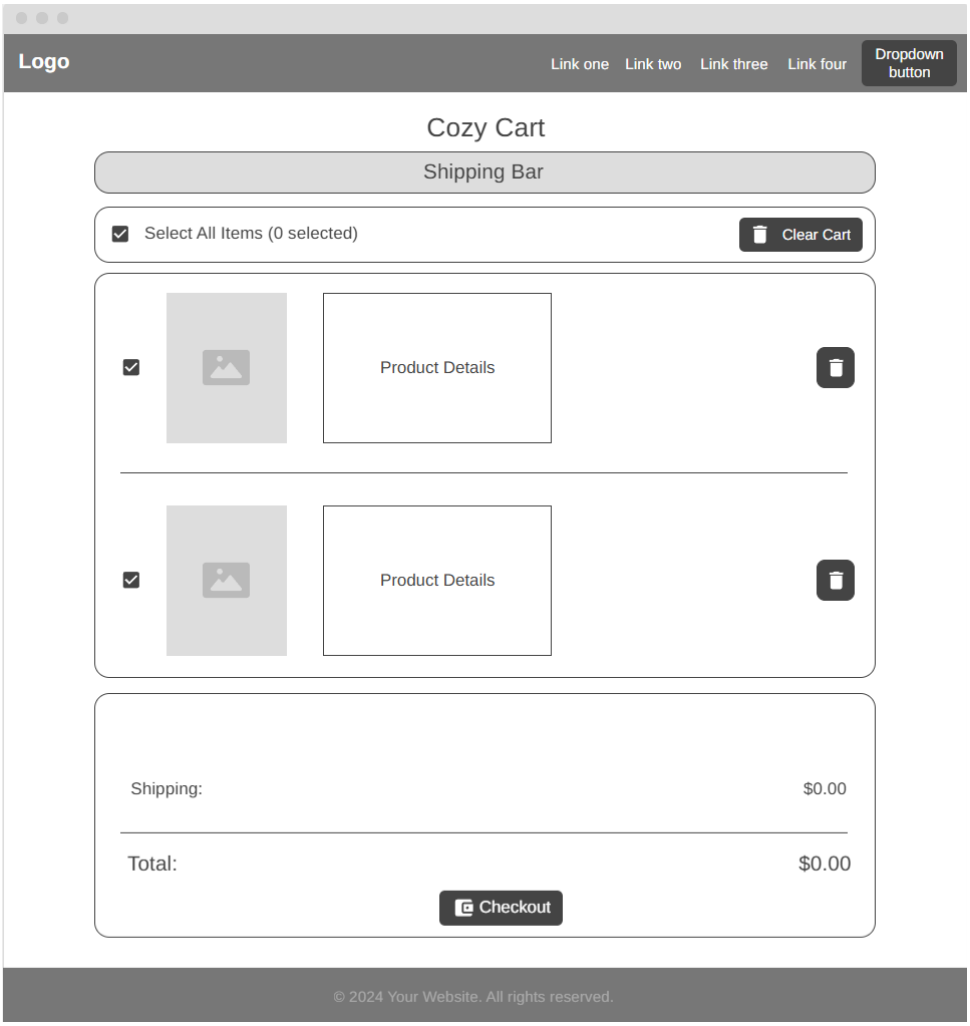
The design creates a seamless transition from browsing to purchasing while maintaining the warm, community-focused brand identity. The cart experience reduces abandonment through clear pricing, shipping incentives, and intuitive management tools, while the playful "cozy"

messaging reinforces the handmade, personal nature of the products and keeps users engaged throughout the shopping process.

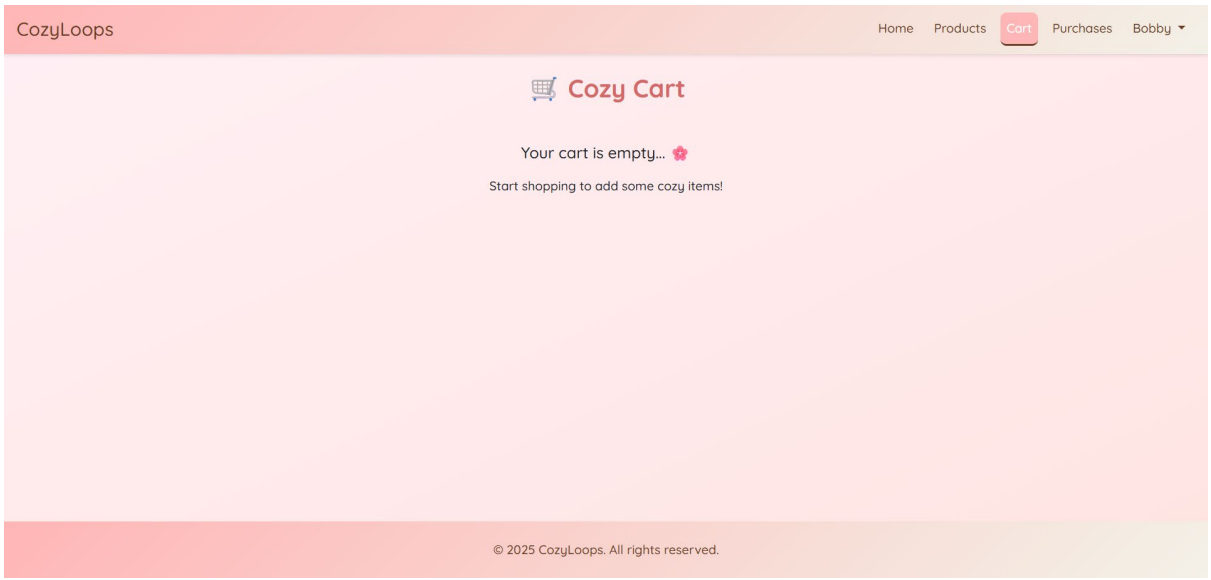
Cart Wireframe (Empty):



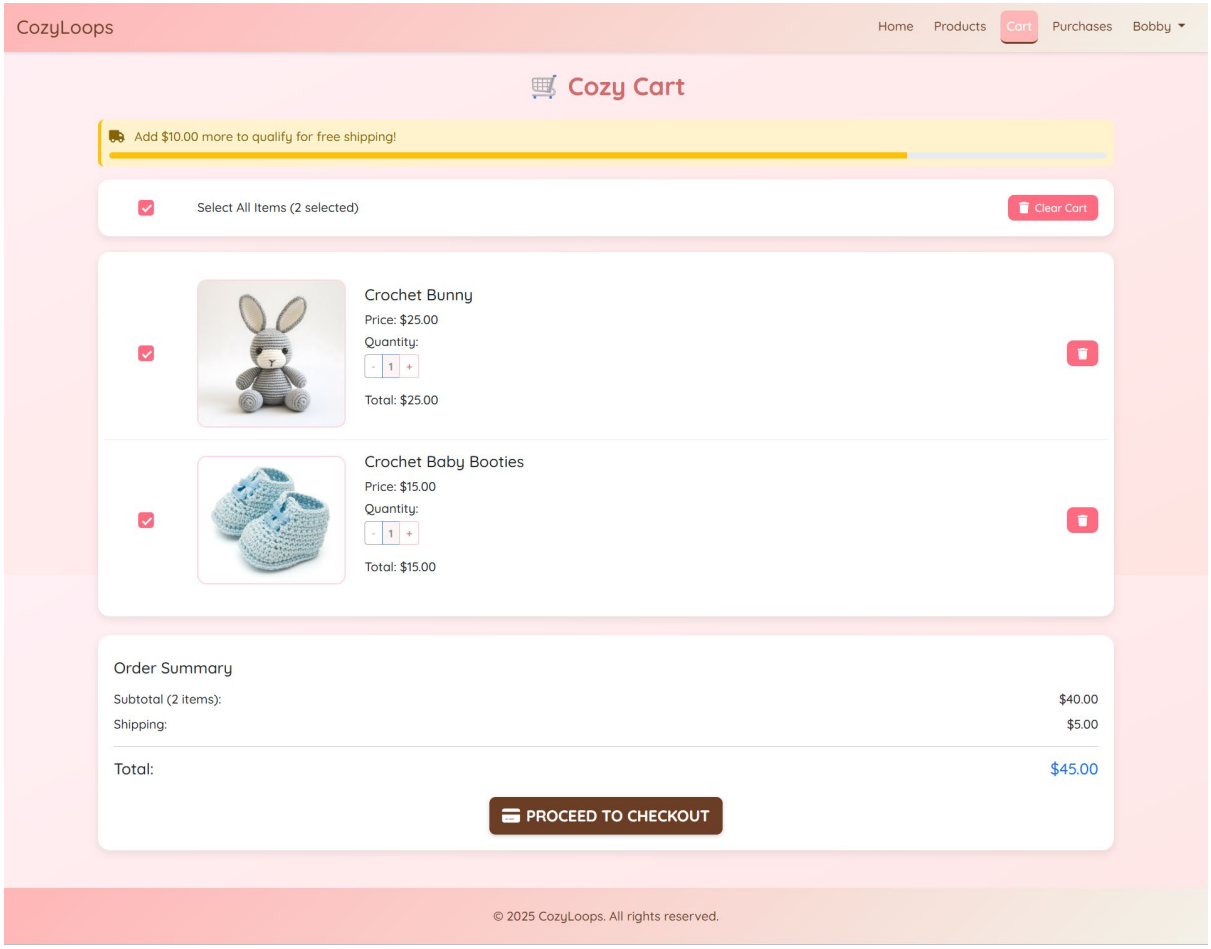
Cart Wireframe (With Products):



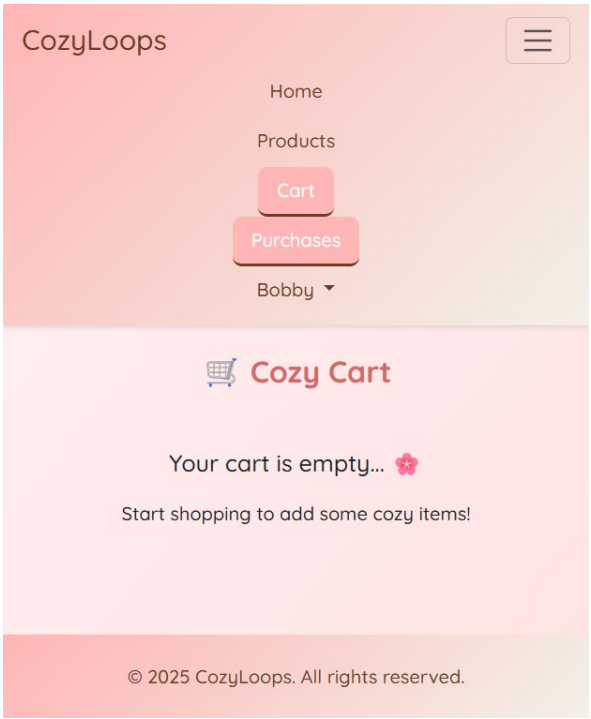
Cart Screenshot (Empty Fullscreen):

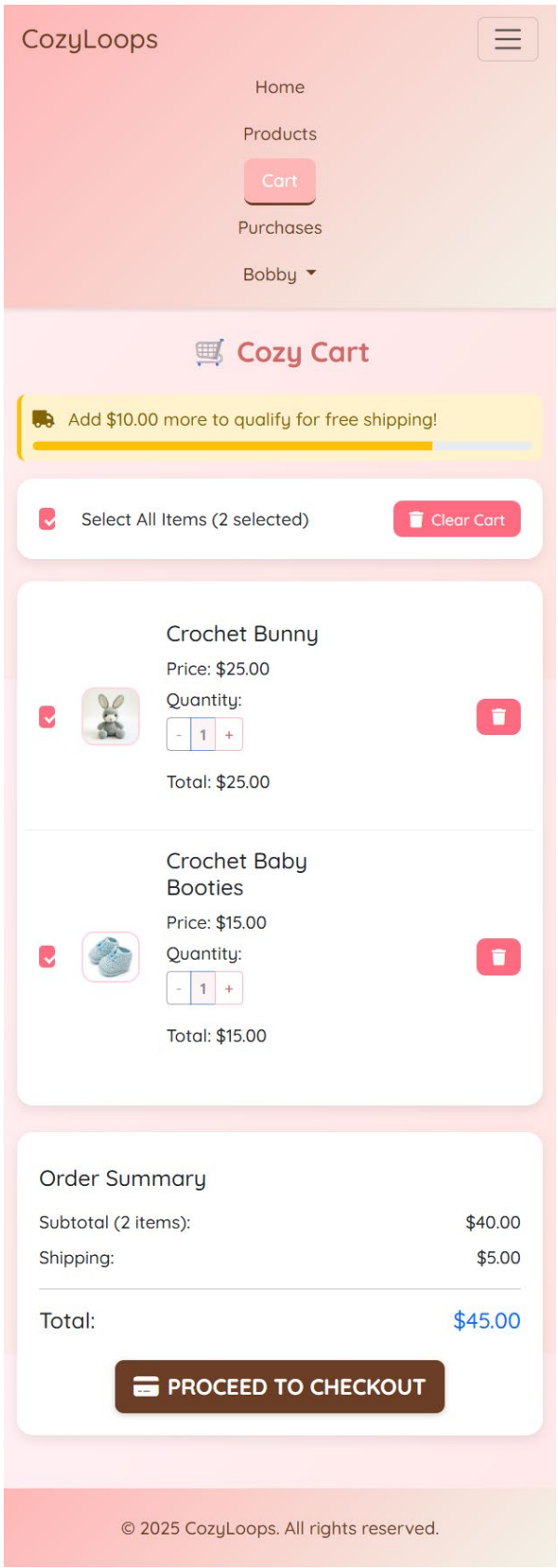


Cart Screenshot (With Products Fullscreen):



Cart Screenshot (Minimized Screen - Responsive):





e. Purchases Page

Overview

The purchase history page provides a comprehensive order management interface that balances detailed transaction records with intuitive filtering capabilities. The design prioritizes order tracking and reorder functionality while maintaining clear information hierarchy for efficient account management.

Key Design Elements

Layout Structure

- **Advanced filtering system**: Multi-parameter search including order search, status filtering, and chronological sorting options
- **Order card layout**: Individual order containers with clear timestamps, order numbers, and status indicators
- **Product-level detail**: Each order displays item images, descriptions, quantities, and pricing breakdown
- **Action-oriented design**: Edit/reorder buttons and status badges prominently positioned for quick access

Visual Design

- **Colour palette**: Consistent soft pink background with white order cards and color-coded status badges (yellow for pending, red for cancelled)
- **Typography**: Clear order numbering (#ORD-2025-xxxx) with readable timestamps and product descriptions
- **Status visualization**: Prominent status badges with intuitive colour coding for immediate order recognition
- **Financial clarity**: Right-aligned pricing with clear subtotal, shipping, and total breakdowns

User Experience

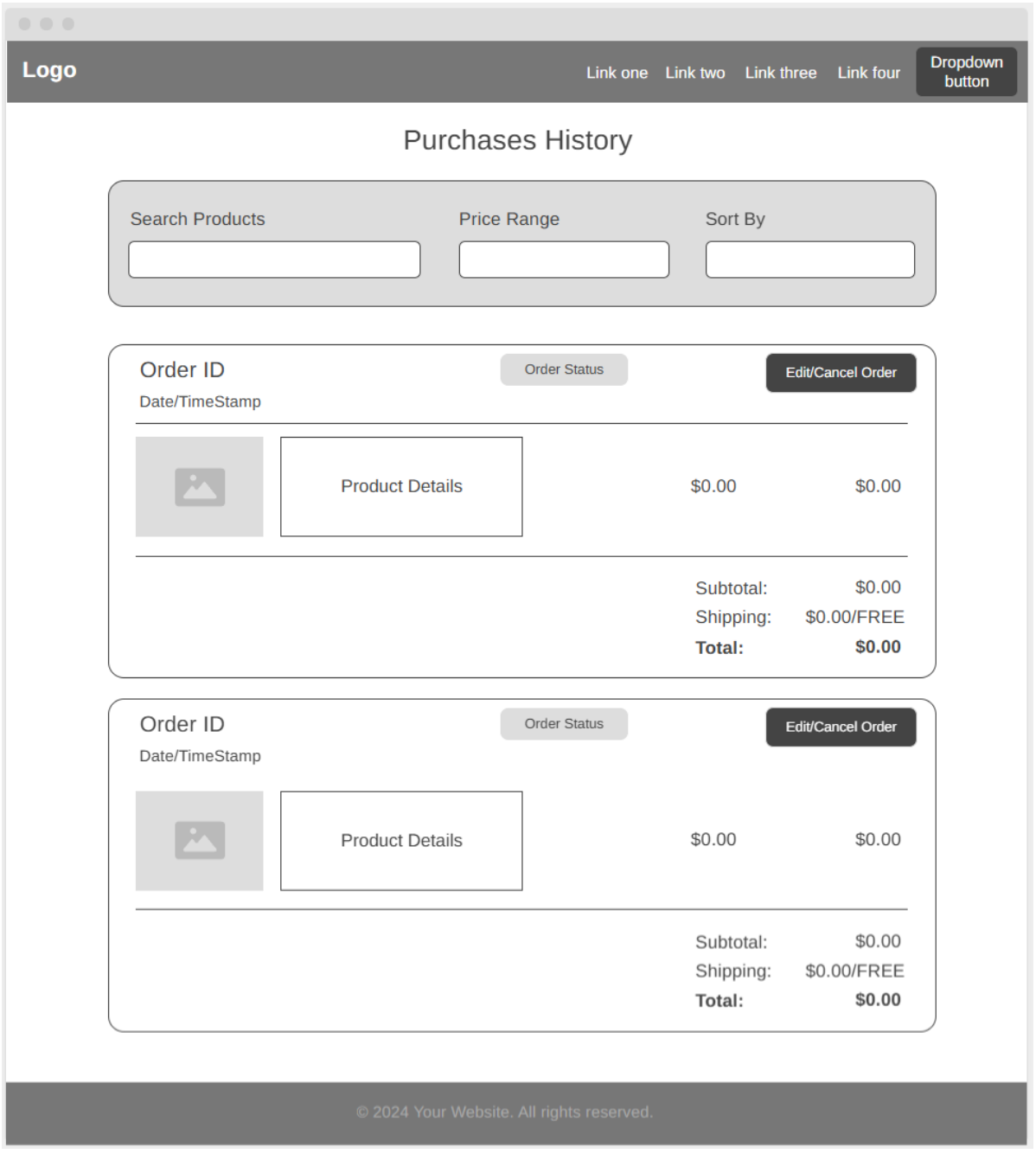
- **Search functionality**: Comprehensive order search by number or item name with placeholder guidance
- **Status management**: Filter orders by status with dropdown selection for focused viewing
- **Chronological organization**: Sort options (Newest First) for logical order browsing
- **Quick actions**: Direct edit/reorder capabilities and detailed order summaries for efficient reordering

Design Goals

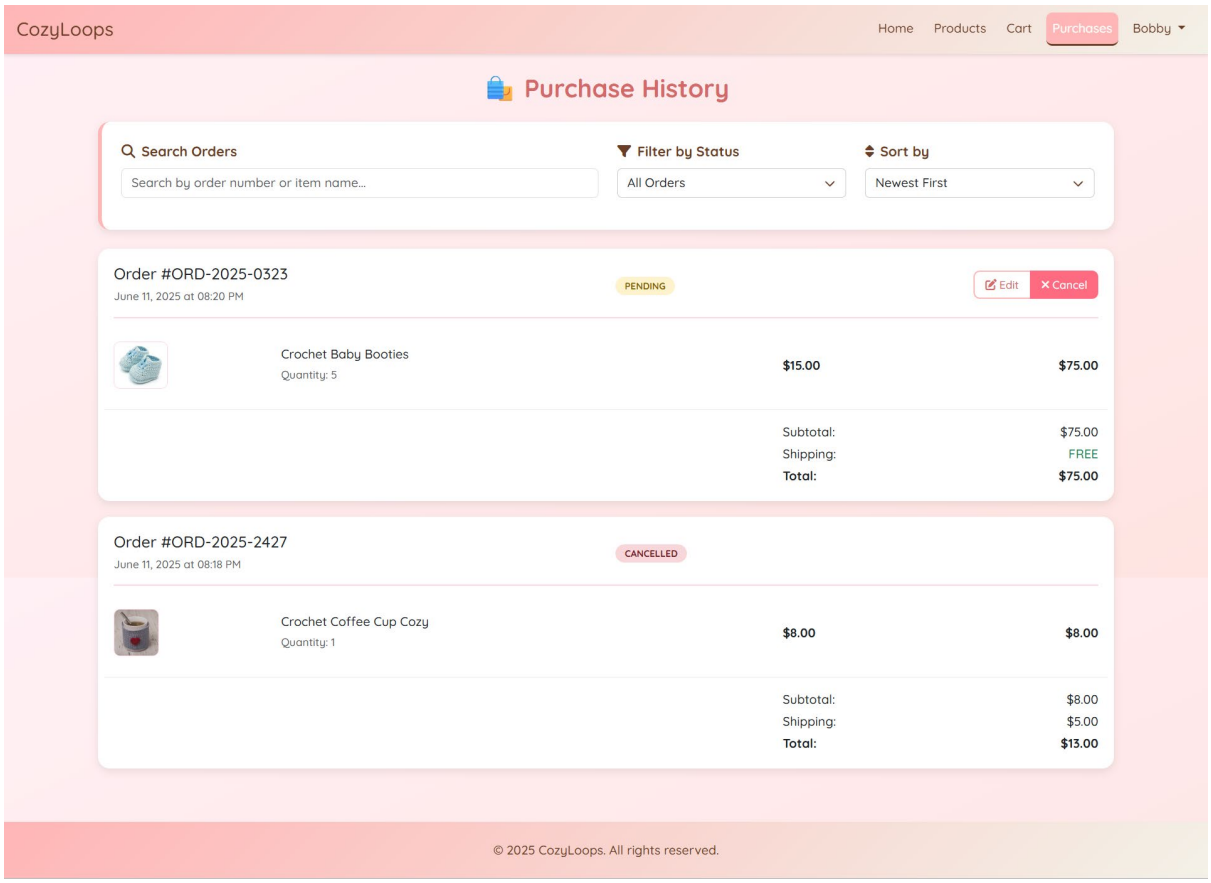
The design creates a professional account management experience that transforms complex order data into an organized, actionable interface. The system reduces customer service inquiries through comprehensive self-service capabilities while encouraging repeat purchases

through streamlined reorder functionality, maintaining the brand's friendly approach to customer relationship management through clear communication and intuitive navigation.

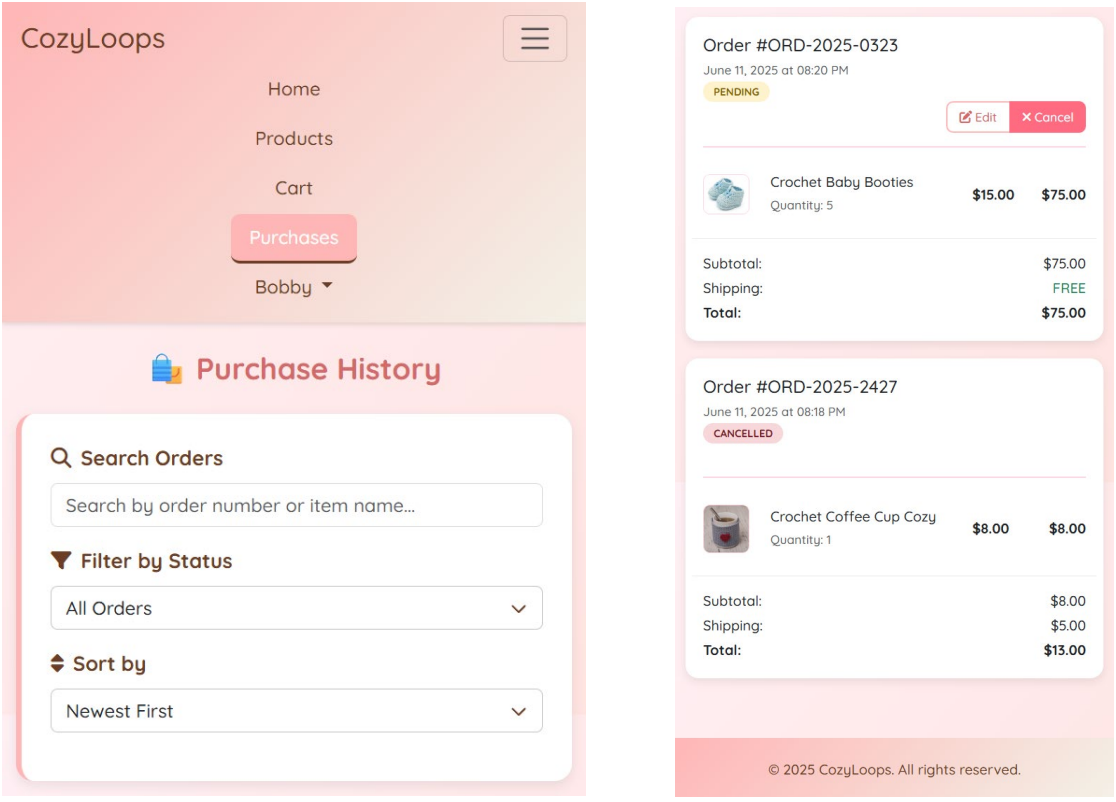
Purchases Wireframe:



Purchases Screenshot (Fullscreen):



Purchases Screenshot (Minimized Screen - Responsive):



f. Profile Page

Overview

The user profile page employs a dual-mode interface that seamlessly transitions between viewing and editing states, providing comprehensive account management while maintaining the brand's warm, accessible aesthetic. The design prioritizes user control and data security through intuitive form organization and clear action hierarchies.

Key Design Elements

Layout Structure

- **Sectioned organization**: Clear separation between User Profile and Shipping Addresses with distinct content cards
- **Modal-style editing**: Edit modes utilize expanded forms with comprehensive field sets for complete data management
- **Progressive disclosure**: View state shows essential information, edit state reveals full form capabilities
- **Consistent action placement**: Edit buttons and form controls positioned consistently across sections

Visual Design

- **Colour palette**: Soft pink background with cream-colored content cards maintaining brand warmth
- **Typography**: Clear section headers with readable form labels and placeholder text guidance
- **Interactive elements**: Brown accent buttons for primary actions, subtle edit icons for quick access
- **Form design**: Clean input fields with proper spacing and logical grouping (contact info, address details)

User Experience

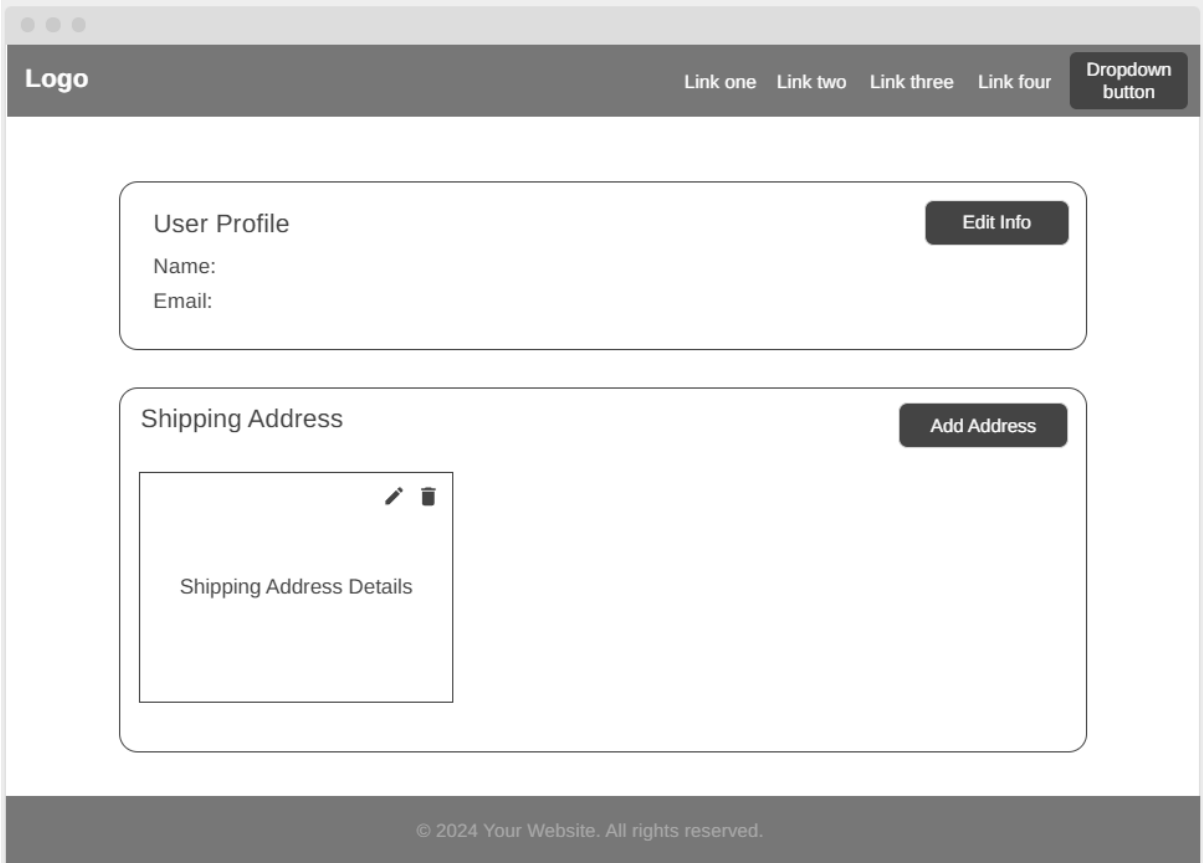
- **Context switching**: Smooth transition between view and edit modes without page refreshes
- **Data organization**: Logical grouping of personal information separate from shipping logistics
- **Security features**: Password confirmation fields and clear save/cancel options for secure updates
- **Address management**: Multiple address support with individual edit capabilities and clear labelling

Design Goals

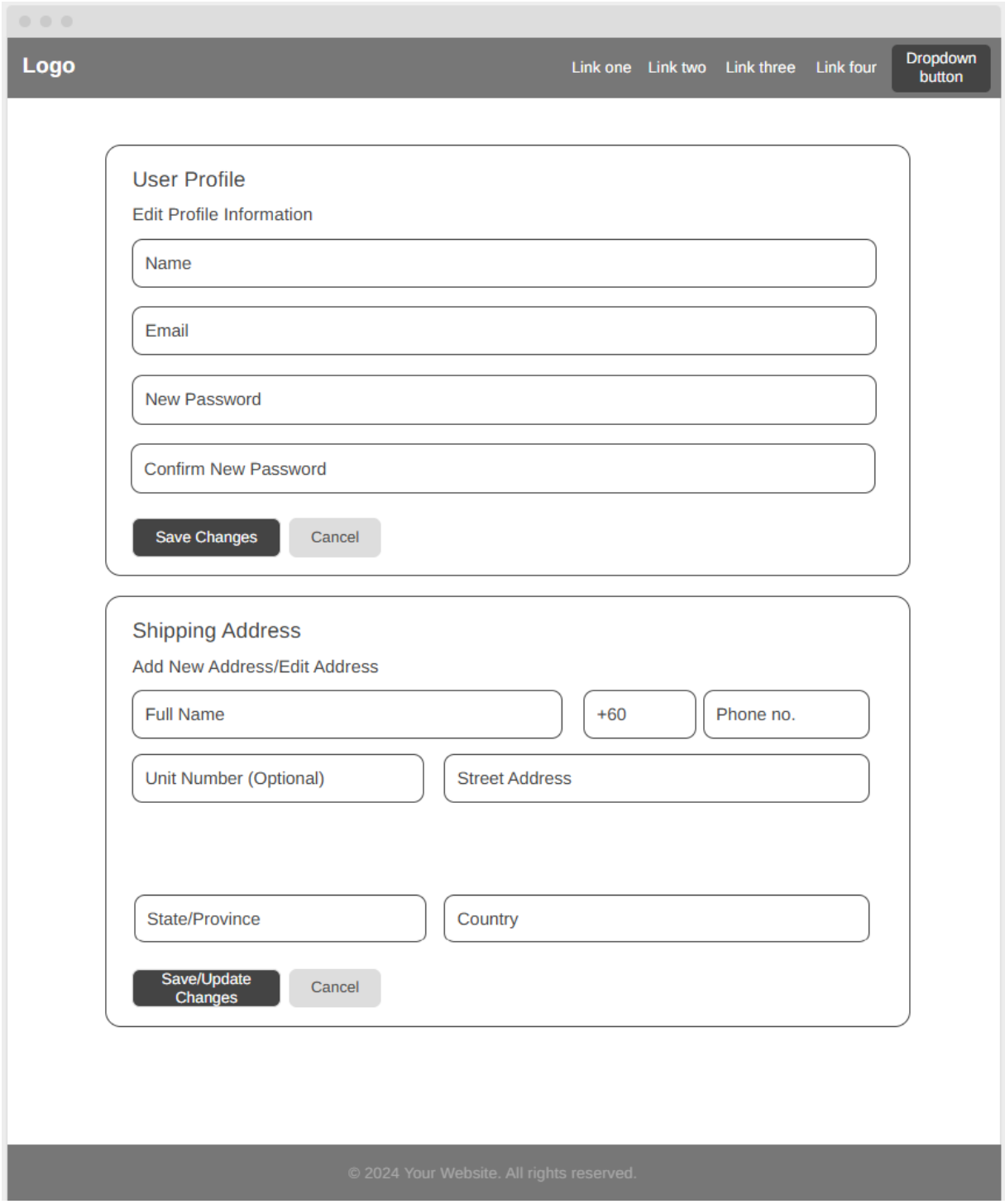
The design creates a comprehensive yet approachable account management experience that builds user confidence through clear information organization and secure editing capabilities.

The interface reduces complexity by separating profile management from shipping logistics while maintaining consistent visual language that reinforces the brand's personal, handmade values through warm colors and intuitive interactions that make account maintenance feel welcoming rather than administrative.

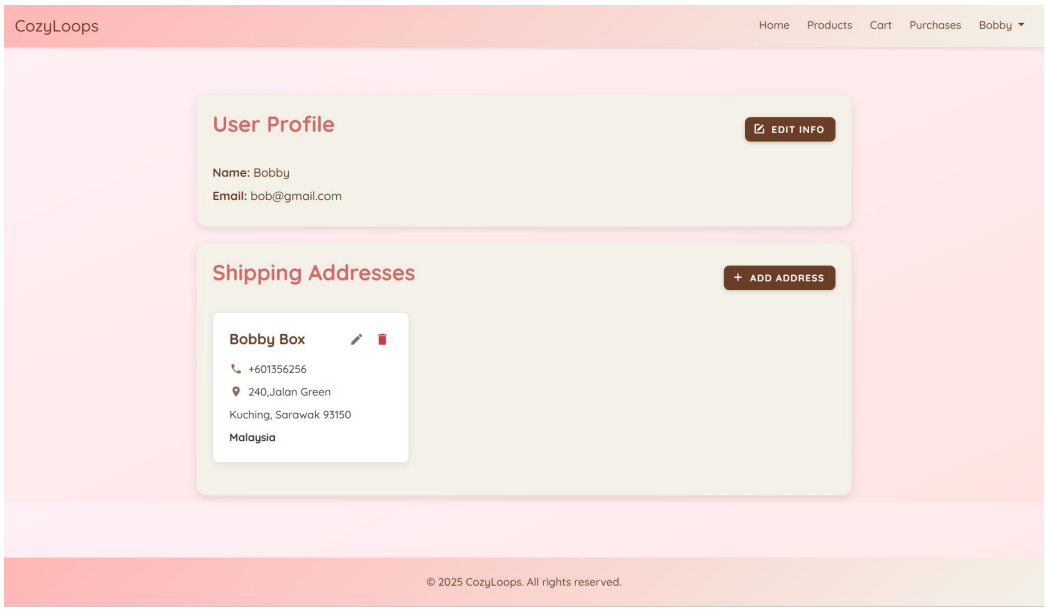
Profile Wireframe (Main View):



Profile Wireframe (Edit View):



Profile Screenshot (Main View Fullscreen):



Profile Screenshot (Main View Fullscreen):