

COS30043 PROJECT 2025

Name: Alison Au Ching Yi

Student ID: 102768672

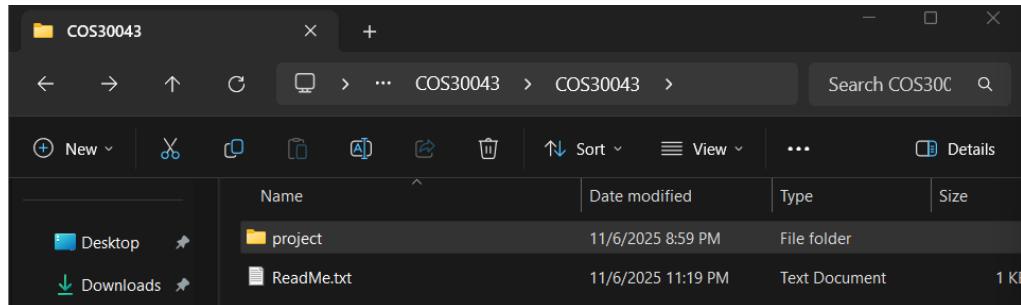


Table of Contents

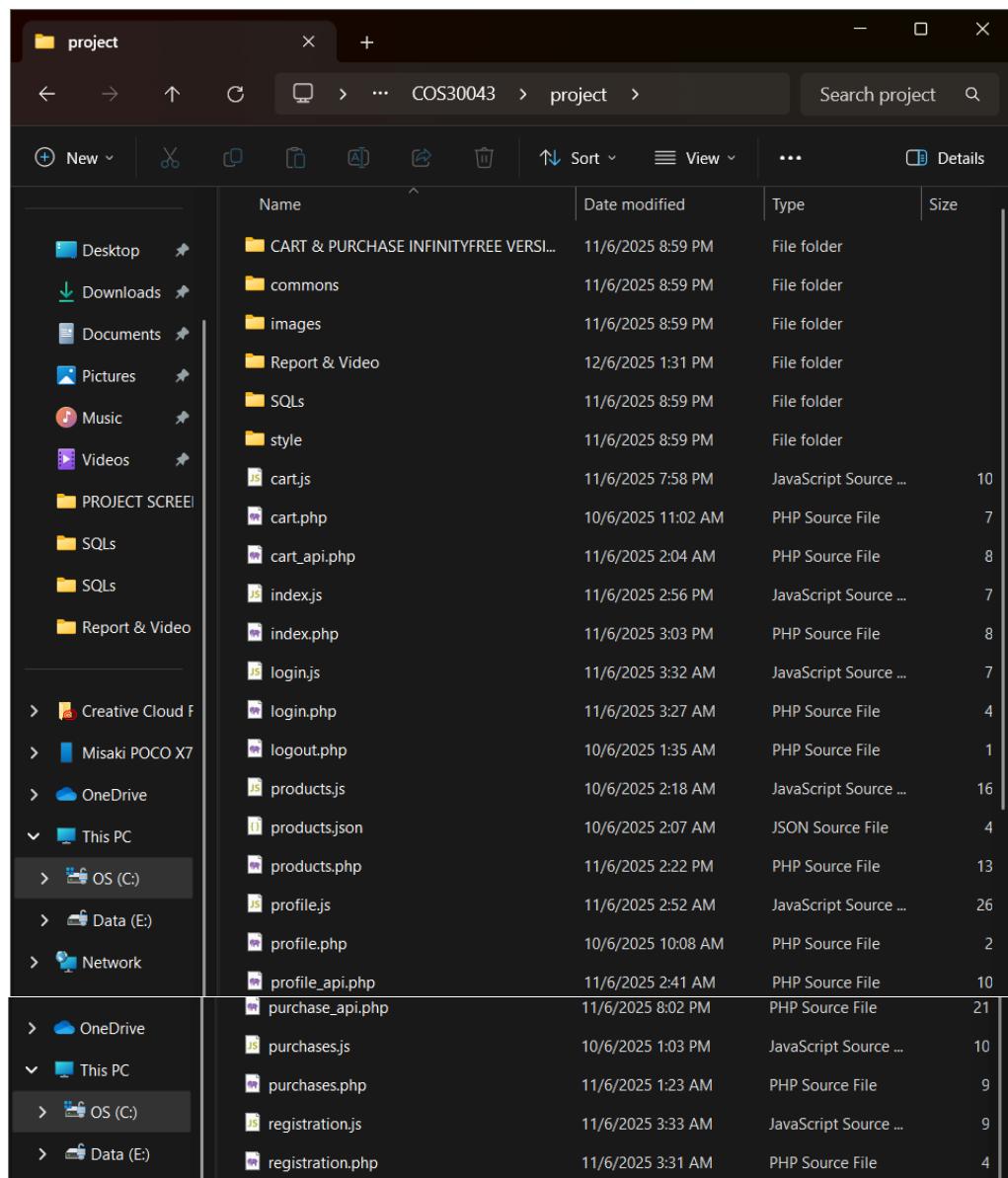
1. Code Folder Structure.....	2
2. Database Creation and Setup.....	6
3. Code Functionality.....	8
a. Use of header.php and footer.php in the Project.....	8
b. Use of Sql codes in the Project.....	11
c. Use of styling codes in the Project	14
d. Use of login.php, login.js and logout.php in the Project	16
e. Use of registration.php, registration.js and registration.php in the Project	24
f. Use of index.php and index.js in the Project.....	30
g. Use of products.php, products.js and products.json in the Project	39
h. Use of cart.php, cart.js and cart_api.php in the Project	55
i. Use of purchases.php, purchases.js and purchases_api.php in the Project	70
j. Use of profile.php, profile.js and profile_api.php in the Project	93
4. Modifications to Code (hosted on InfinityFree Hosting Platform)	114
a. Modification to the SQL files	114
b. Modification to the PHP files	115
5. How the Website was Hosted on InfinityFree.....	117
6. Changes of Code Structure in Cart and Purchases Page (InfinityFree)	122
a. cart_api.php & cart.js	122
b. purchases_api.php & purchases.js	129
7. Design Concepts	137
a. Login & Registration Page.....	138
b. Index Page.....	143
c. Products Page	148
d. Cart Page	153
e. Purchases Page	157
f. Profile Page	160

1. Code Folder Structure

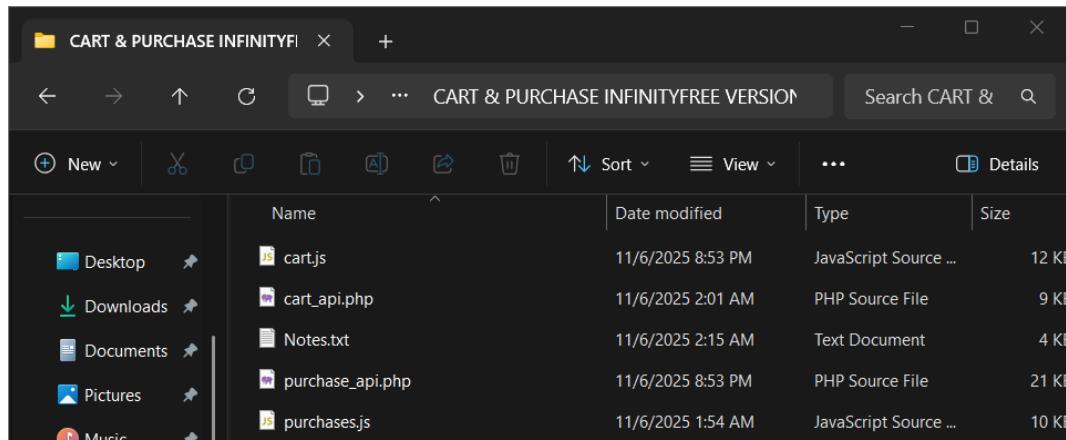
The submitted code is placed in a folder called **COS30043**. This folder should be placed in the xampp/htdocs folder. In **COS30043**, there is a **ReadMe.txt** text file and a folder called **project**.



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.



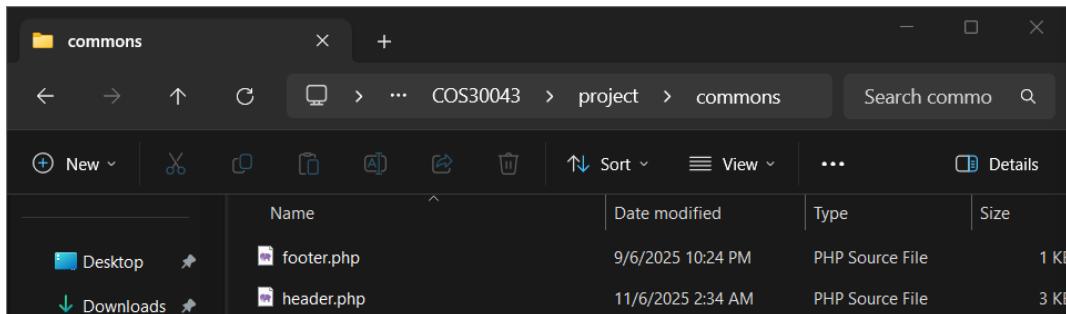
In the **CART & PURCHASE INFINITYFREE VERSION** folder, it contains the code files that are changed to accommodate the limitations of hosting the website on a free website hosting platform called **InfinityFree** - <https://www.infinityfree.com>. It also contains the Notes.txt text file that describes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



A screenshot of a file explorer window titled "CART & PURCHASE INFINITYFREE VERSION". The left sidebar shows standard folder icons for Desktop, Downloads, Documents, Pictures, and Music. The main pane displays a list of files with columns for Name, Date modified, Type, and Size. The files listed are:

	Name	Date modified	Type	Size
	cart.js	11/6/2025 8:53 PM	JavaScript Source ...	12 KB
	cart_api.php	11/6/2025 2:01 AM	PHP Source File	9 KB
	Notes.txt	11/6/2025 2:15 AM	Text Document	4 KB
	purchase_api.php	11/6/2025 8:53 PM	PHP Source File	21 KB
	purchases.js	11/6/2025 1:54 AM	JavaScript Source ...	10 KB

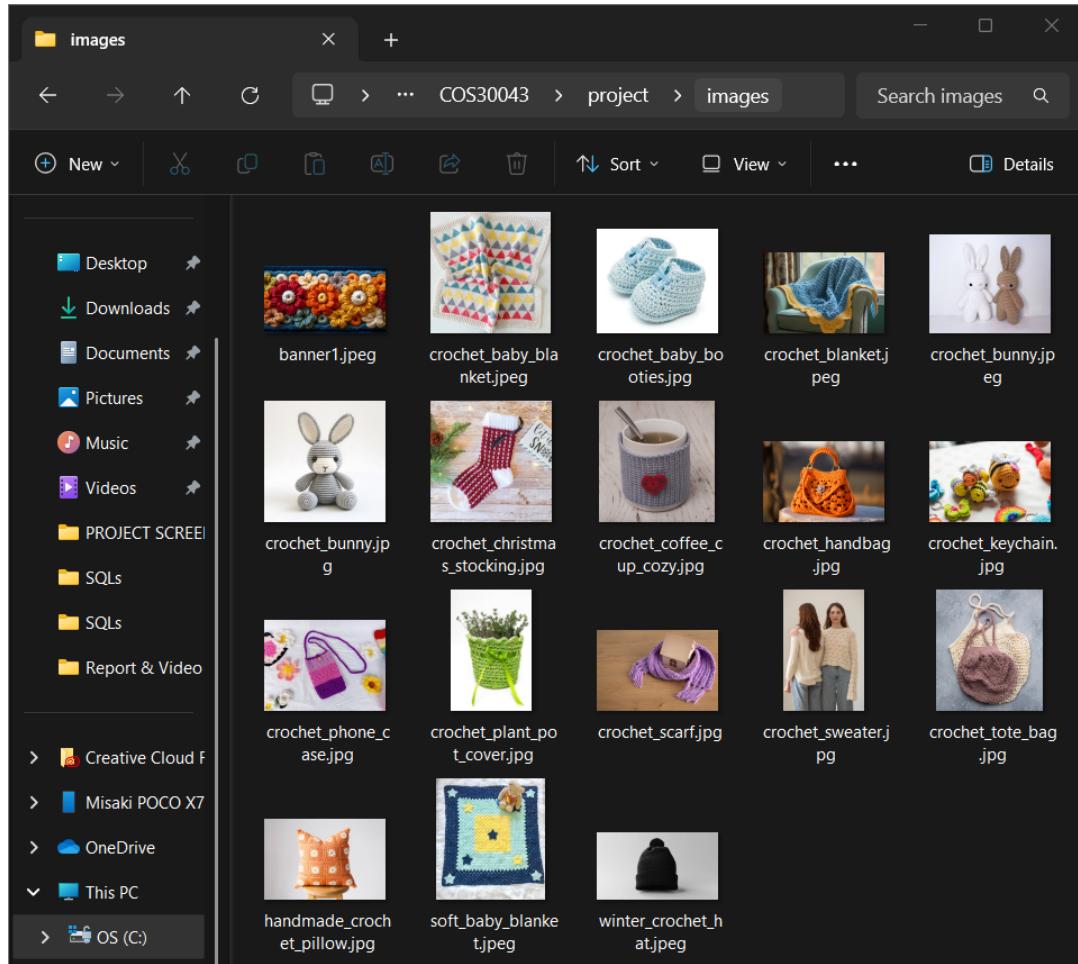
In the **commons** folder, there are the **header.php** and **footer.php** files. These files are the same across all the pages of my website, so it is easier to modify the header and footer through a universal file, that's why the header and footer files are created.



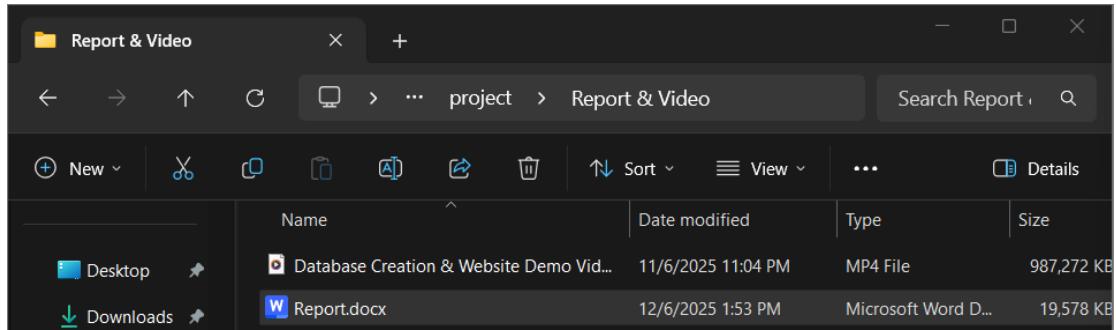
A screenshot of a file explorer window titled "commons". The left sidebar shows standard folder icons for Desktop and Downloads. The main pane displays a list of files with columns for Name, Date modified, Type, and Size. The files listed are:

	Name	Date modified	Type	Size
	footer.php	9/6/2025 10:24 PM	PHP Source File	1 KB
	header.php	11/6/2025 2:34 AM	PHP Source File	3 KB

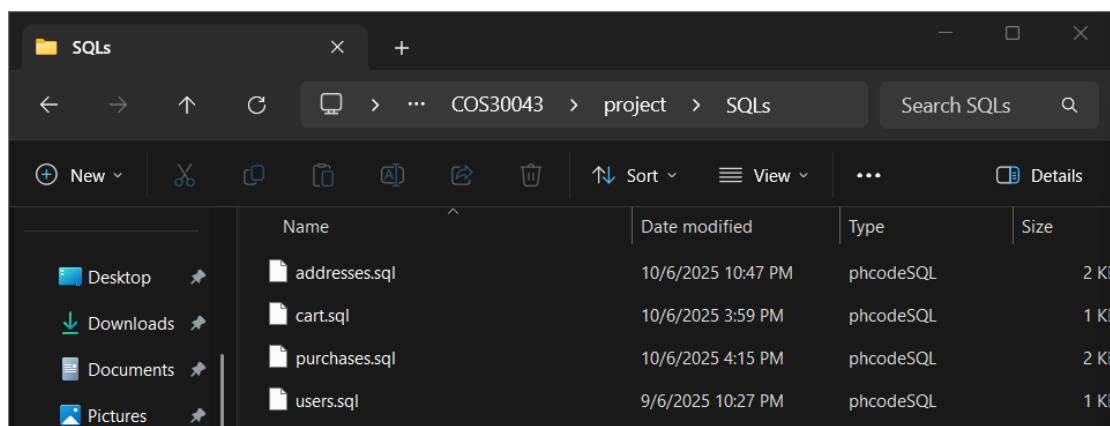
In the **images** folder, there are the various images used on the website such as the product images and the index page banner image. All these images are obtained from **Freepik** - <https://www.freepik.com>.



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



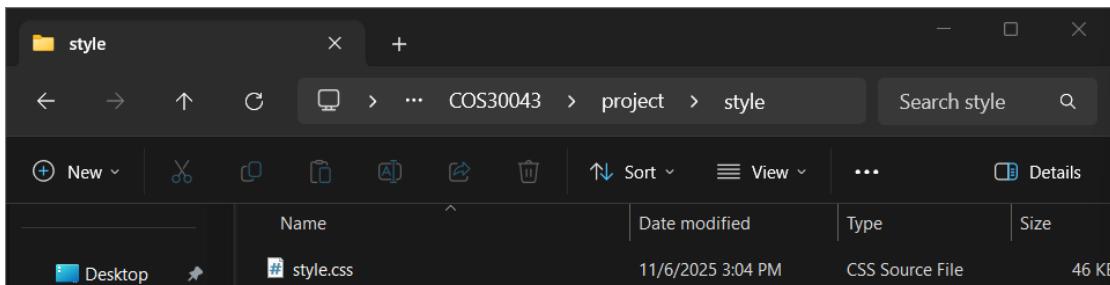
In the **SQLs** folder, it contains the **sql files** needed for table creation in the database.



A screenshot of a Windows File Explorer window titled "SQLs". The path is "COS30043 > project > SQLs". The search bar says "Search SQLs". The table below lists four files:

	Name	Date modified	Type	Size
Desktop	addresses.sql	10/6/2025 10:47 PM	phcodeSQL	2 KB
Downloads	cart.sql	10/6/2025 3:59 PM	phcodeSQL	1 KB
Documents	purchases.sql	10/6/2025 4:15 PM	phcodeSQL	2 KB
Pictures	users.sql	9/6/2025 10:27 PM	phcodeSQL	1 KB

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



A screenshot of a Windows File Explorer window titled "style". The path is "COS30043 > project > style". The search bar says "Search style". The table below lists one file:

	Name	Date modified	Type	Size
Desktop	style.css	11/6/2025 3:04 PM	CSS Source File	46 KB

2. Database Creation and Setup

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

The screenshot shows the 'Databases' section of the phpMyAdmin interface. At the top, there is a search bar with the URL 'localhost/phpmyadmin/index.php?route=/server/databases'. Below the search bar is a navigation menu with tabs: Databases, SQL, Status, User accounts, Export, Import, Settings, and R. The 'Databases' tab is selected. A 'Create database' button is visible. In the main area, a form is displayed with the database name 'project' entered and the character set/collation 'utf8mb4_general_ci' selected. A 'Create' button is highlighted with a mouse cursor. Below the form is a checkbox labeled 'Check all' and a 'Drop' button. The main table lists existing databases: alumni, cart_system, foodege, if0_39196567_cos30043_project, information_schema, lab7, lab8, lab9, laravel, laravel_demo, and laravel_test. Each database entry includes a checkbox, its name, collation, and a 'Check privileges' link.

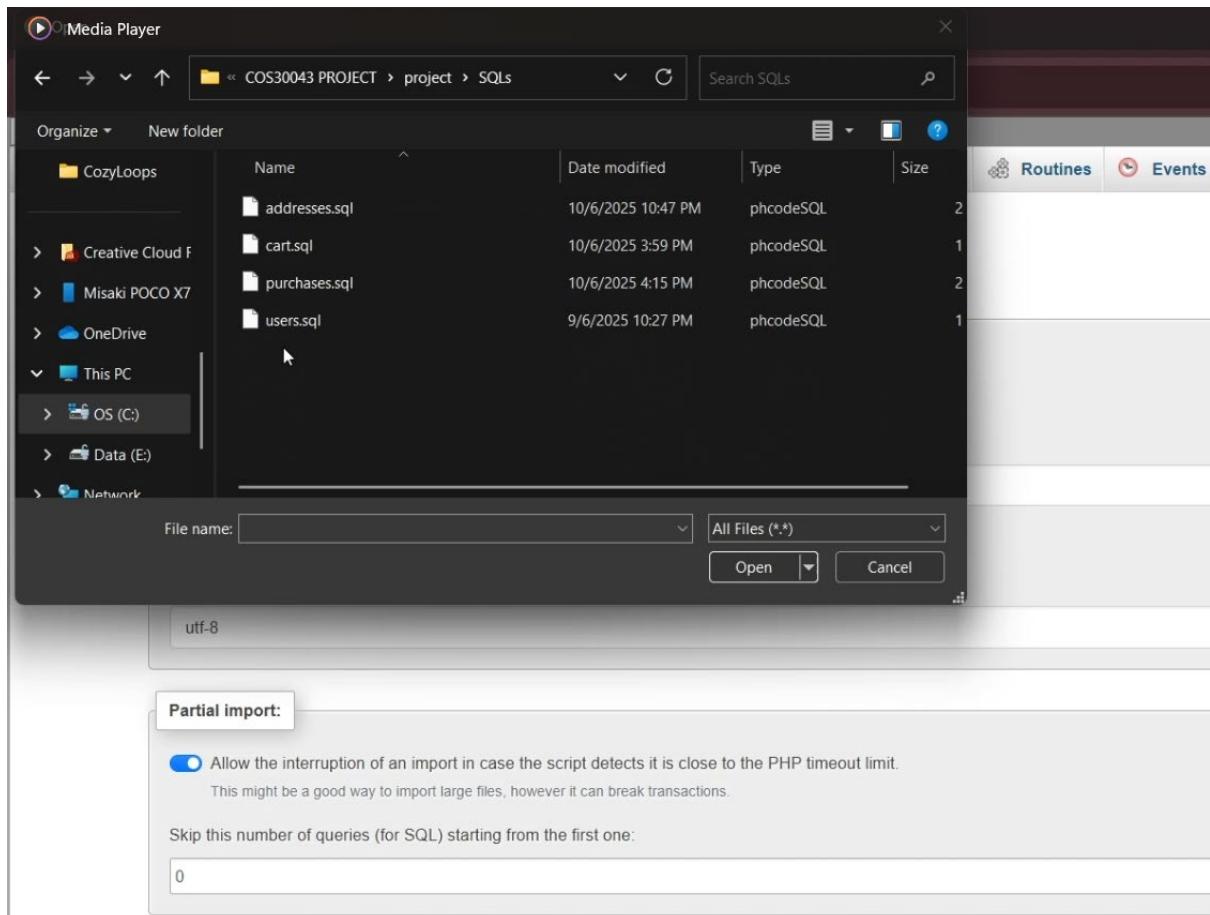
Database	Collation	Action
alumni	utf8mb4_general_ci	Check privileges
cart_system	utf8mb4_general_ci	Check privileges
foodege	utf8mb4_general_ci	Check privileges
if0_39196567_cos30043_project	utf8mb4_general_ci	Check privileges
information_schema	utf8_general_ci	Check privileges
lab7	utf8mb4_general_ci	Check privileges
lab8	utf8mb4_general_ci	Check privileges
lab9	utf8mb4_general_ci	Check privileges
laravel	utf8mb4_general_ci	Check privileges
laravel_demo	utf8mb4_general_ci	Check privileges
laravel_test	utf8mb4_general_ci	Check privileges

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

The screenshot shows the 'Structure' screen for the 'project' database. The URL in the address bar is 'localhost/phpmyadmin/index.php?route=/server/databases'. The navigation menu at the top includes tabs for Structure, SQL, Search, Query, Export, Import, Operations, and Privileges. The 'Import' tab is currently selected and has a mouse cursor over it. A message 'No tables found in database.' is displayed. Below this, there is a 'Create new table' button. A form allows the user to enter the 'Table name' (which is empty) and the 'Number of columns' (which is set to 4). A 'Create' button is located at the bottom of this form.

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.

The screenshot shows the phpMyAdmin interface for the "project" database, displaying the following table structure:

Table	Action	Rows	Type	Collation	Size	Overhead
addresses	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	64.0 KiB	-
cart	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	48.0 KiB	-
purchases	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	64.0 KiB	-
purchase_items	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	32.0 KiB	-
users	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	32.0 KiB	-
5 tables	Sum	0	InnoDB	utf8mb4_general_ci	240.0 KiB	0 B

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.



```
<?php
session_start();
if (!isset($_SESSION['logged_in']) || $_SESSION['logged_in'] !== true) {
    header('Location: login.php');
    exit;
}
$user = [
    'id' => $_SESSION['user_id'],
    'name' => $_SESSION['user_name'],
    'email' => $_SESSION['user_email']
];
?>
```

- **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).

```
13 <nav class="navbar navbar-expand-lg sticky-top px-3">
14     <a class="navbar-brand" href="index.php">CozyLoops</a>
15     <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav"
16         aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
17         <span class="navbar-toggler-icon"></span>
18     </button>
19     <div class="collapse navbar-collapse justify-content-end" id="navbarNav">
20         <ul class="navbar-nav align-items-center">
21             <li class="nav-item">
22                 <a class="nav-link <?php echo basename($_SERVER['PHP_SELF']) == 'index.php' ? 'active' : ''; ?>" href="index.php">Home</a>
23             </li>
24             <li class="nav-item">
25                 <a class="nav-link <?php echo basename($_SERVER['PHP_SELF']) == 'products.php' ? 'active' : ''; ?>" href="products.php">Products</a>
26             </li>
27             <li class="nav-item">
28                 <a class="nav-link <?php echo basename($_SERVER['PHP_SELF']) == 'cart.php' ? 'active' : ''; ?>" href="cart.php">Cart</a>
29             </li>
30             <li class="nav-item">
31                 <a class="nav-link <?php echo basename($_SERVER['PHP_SELF']) == 'purchases.php' ? 'active' : ''; ?>" href="purchases.php">Purchases</a>
32             </li>
33             <li class="nav-item dropdown">
34                 <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">
35                     <?php echo htmlspecialchars($user['name']); ?>
36                 </a>
37                 <ul class="dropdown-menu dropdown-menu-end">
38                     <li><a class="dropdown-item" href="profile.php">Profile</a></li>
39                     <li><a class="dropdown-item" href="logout.php">Logout</a></li>
40                 </ul>
41             </li>
42         </ul>
43     </div>
44 </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.

```
1 <footer class="text-center mt-4">
2     <div class="container">
3         <p class="mb-0">&copy; 2025 CozyLoops. All rights reserved.</p>
4     </div>
5 </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 -->           <!-- Footer -->
<?php include 'commons/header.php'; ?> <?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.

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

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

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

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.

```

1  -- Create the database
2  CREATE DATABASE IF NOT EXISTS project;
3  USE project;
4
5  -- Drop and recreate addresses table with new fields
6  DROP TABLE IF EXISTS addresses;
7
8  CREATE TABLE addresses (
9      id INT AUTO_INCREMENT PRIMARY KEY,
10     user_id INT NOT NULL,
11     name VARCHAR(100) NOT NULL,
12     phone VARCHAR(20) NOT NULL,
13     unit_number VARCHAR(50), -- Optional unit number
14     street VARCHAR(255) NOT NULL,
15     city VARCHAR(100) NOT NULL,
16     state VARCHAR(100) NOT NULL, -- New required field
17     postcode VARCHAR(20) NOT NULL,
18     country VARCHAR(100) NOT NULL DEFAULT 'Malaysia', -- New required field with default
19     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
20     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
21
22     -- Add foreign key constraint if users table exists
23     -- FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
24
25     -- Add indexes for better performance
26     INDEX idx_user_id (user_id),
27     INDEX idx_country (country),
28     INDEX idx_state (state)
29 );

```

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.

```

1  -- Create the database
2  CREATE DATABASE IF NOT EXISTS project;
3  USE project;
4
5  -- Drop and recreate users table
6  DROP TABLE IF EXISTS cart;
7
8  CREATE TABLE cart (
9      id INT AUTO_INCREMENT PRIMARY KEY,
10     user_id INT NOT NULL,
11     product_id INT NOT NULL,
12     name VARCHAR(255) NOT NULL,
13     price DECIMAL(10,2) NOT NULL,
14     image VARCHAR(255) NOT NULL,
15     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
16     FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
17     INDEX idx_user_id (user_id),
18     INDEX idx_product_user (product_id, user_id)
19 );

```

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.

```
1 -- Create the database
2 CREATE DATABASE IF NOT EXISTS project;
3 USE project;
4
5 -- Drop and recreate tables
6 DROP TABLE IF EXISTS purchase_items;
7 DROP TABLE IF EXISTS purchases;
8
9 -- Create purchases table to store order information
10 CREATE TABLE IF NOT EXISTS purchases (
11     id INT AUTO_INCREMENT PRIMARY KEY,
12     user_id INT NOT NULL,
13     order_number VARCHAR(50) UNIQUE NOT NULL,
14     total_amount DECIMAL(10,2) NOT NULL,
15     shipping_cost DECIMAL(10,2) DEFAULT 0.00,
16     order_status ENUM('pending', 'processing', 'shipped', 'delivered', 'cancelled') DEFAULT
17     'pending',
18     order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
19     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
20     FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
21     INDEX idx_user_id (user_id),
22     INDEX idx_order_number (order_number)
23 );
24
25 -- Create purchase_items table to store individual items in each order
26 CREATE TABLE IF NOT EXISTS purchase_items (
27     id INT AUTO_INCREMENT PRIMARY KEY,
28     purchase_id INT NOT NULL,
29     product_id INT NOT NULL,
30     name VARCHAR(255) NOT NULL,
31     price DECIMAL(10,2) NOT NULL,
32     quantity INT NOT NULL,
33     image VARCHAR(255),
34     FOREIGN KEY (purchase_id) REFERENCES purchases(id) ON DELETE CASCADE,
35     INDEX idx_purchase_id (purchase_id)
36 );
```

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

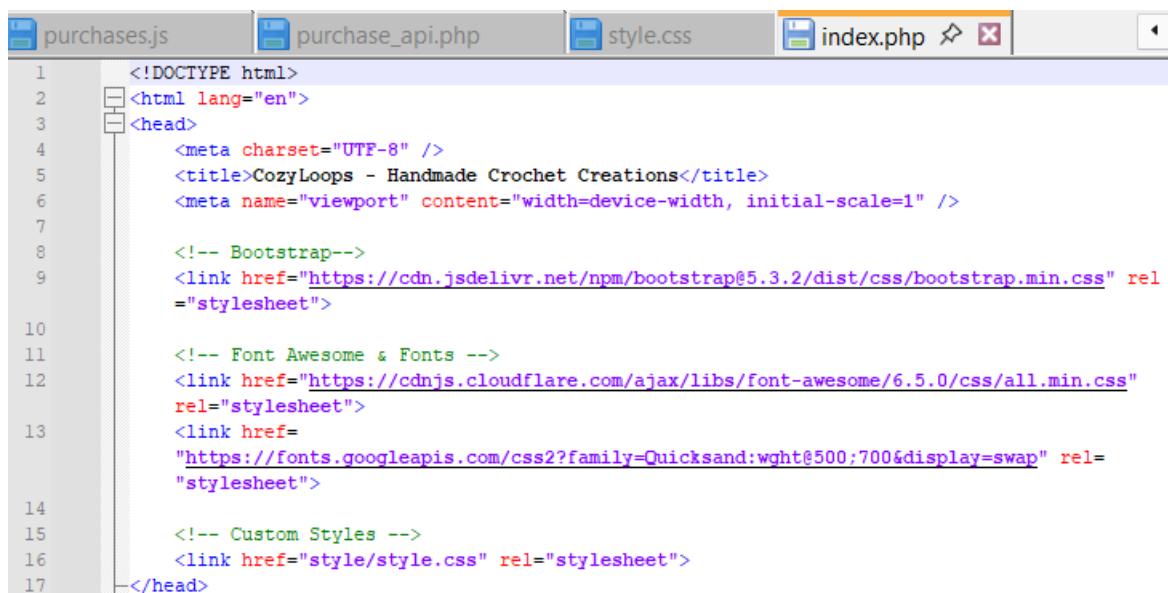
- Uses @keyframes for shimmer and fade-in effects, enhancing user experience.
- Applies hover transformations to elements like category cards (.index_category) and buttons.

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:



```
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=
14         "https://fonts.googleapis.com/css2?family=Quicksand:wght@500;700&display=swap" rel=
15         "stylesheet">
16
17     <!-- Custom Styles -->
18     <link href="style/style.css" rel="stylesheet">
19 </head>
```

- 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.

```
1  <?php
2  // Prevent any output before JSON response
3  ini_set('display_errors', 0);
4  error_reporting(0);
5  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.

```
9   // Handle POST requests for login
10  if (
11      $_SERVER['REQUEST_METHOD'] === 'POST' &&
12      isset($_SERVER['CONTENT_TYPE']) &&
13      strpos($_SERVER['CONTENT_TYPE'], 'application/json') === 0
14  ) {
15      // Clear any previous output
16      ob_clean();
17      header("Content-Type: application/json");
18
19      try {
20          // Parse JSON input
21          $input = json_decode(file_get_contents('php://input'), true);
22          if (!$input || !isset($input['email'], $input['password'])) {
23              http_response_code(400);
24              echo json_encode(['message' => 'Invalid input']);
25              exit;
26          }
27
28          $email = trim($input['email']);
29          $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.

```
31  // Database connection using prepared statements
32  $conn = mysqli_connect('localhost', 'root', '', 'project');
33  if (!$conn) {
34      http_response_code(500);
35      echo json_encode(['message' => 'Database connection failed: ' .
36                      mysqli_connect_error()]);
37      exit;
38
39  mysqli_set_charset($conn, 'utf8');
40
41  // Use prepared statement to prevent SQL injection
42  $query = "SELECT * FROM users WHERE email = ?";
43  $stmt = mysqli_prepare($conn, $query);
44
45  if (!$stmt) {
46      http_response_code(500);
47      echo json_encode(['message' => 'Database query failed']);
48      mysqli_close($conn);
49      exit;
50  }
```

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.

```
52     mysqli_stmt_bind_param($stmt, "s", $email);
53     mysqli_stmt_execute($stmt);
54     $result = mysqli_stmt_get_result($stmt);
55
56     if ($user = mysqli_fetch_assoc($result)) {
57         if (password_verify($password, $user['password'])) {
58             // Store user data in session
59             $_SESSION['user_id'] = $user['id'];
60             $_SESSION['user_name'] = $user['name'];
61             $_SESSION['user_email'] = $user['email'];
62             $_SESSION['logged_in'] = true;
63
64             echo json_encode([
65                 'message' => 'Login successful',
66                 'user' => [
67                     'id' => $user['id'],
68                     'name' => $user['name'],
69                     'email' => $user['email']
70                 ],
71                 'redirectUrl' => 'index.php'
72             ]);
73         } else {
74             http_response_code(401);
75             echo json_encode(['message' => 'Incorrect password']);
76         }
77     } else {
78         http_response_code(404);
79         echo json_encode(['message' => 'User not found']);
80     }
81
82     mysqli_stmt_close($stmt);
83     mysqli_close($conn);
84 } catch (Exception $e) {
85     http_response_code(500);
86     echo json_encode(['message' => 'Server error: ' . $e->getMessage()]);
87 }
88
89 exit;
90 }
```

5. Redirect Logged-in Users

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

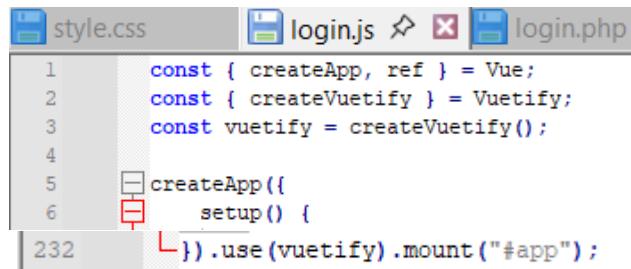
```
92     // Check if user is already logged in via session
93     if (isset($_SESSION['logged_in']) && $_SESSION['logged_in'] === true) {
94         header('Location: index.php');
95         exit;
96     }
97 ?>
```

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

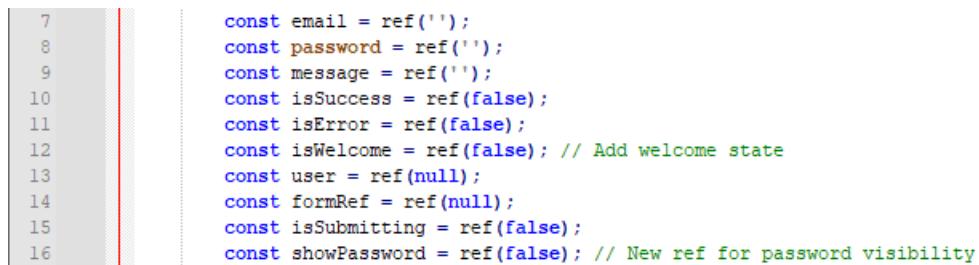
- Mounts a Vue app on the #app div in login.php.
- Uses Vuetify UI components for styling and layout.



```
style.css login.js login.php
1 const { createApp, ref } = Vue;
2 const { createVuetify } = Vuetify;
3 const vuetify = createVuetify();
4
5 createApp({
6   setup() {
232     L}).use(vuetify).mount("#app");
```

2. Reactive Form Inputs

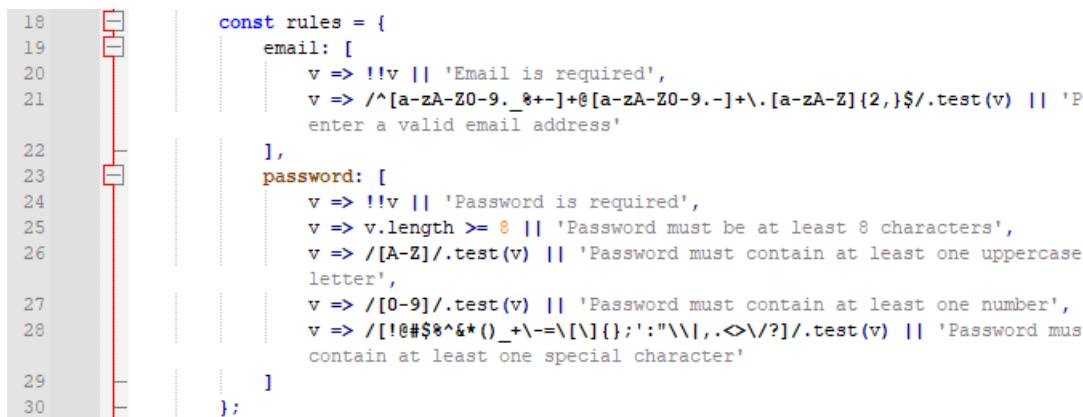
- Data bindings for user input.
- Linked to v-text-field components for live updates and validation.



```
7 const email = ref('');
8 const password = ref('');
9 const message = ref('');
10 const isSuccess = ref(false);
11 const isError = ref(false);
12 const isWelcome = ref(false); // Add welcome state
13 const user = ref(null);
14 const formRef = ref(null);
15 const isSubmitting = ref(false);
16 const showPassword = ref(false); // New ref for password visibility
```

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.



```
18 const rules = [
19   {
20     email: [
21       v => !!v || 'Email is required',
22       v => /^[a-zA-Z0-9._8+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\$/.test(v) || 'Please
23       enter a valid email address'
24     ],
25     password: [
26       v => !!v || 'Password is required',
27       v => v.length >= 8 || 'Password must be at least 8 characters',
28       v => /[A-Z]/.test(v) || 'Password must contain at least one uppercase
29       letter',
30       v => /[0-9]/.test(v) || 'Password must contain at least one number',
31       v => /[!@#$%^&*()_+=\{\}';':"\\"|,.<>\?]/.test(v) || 'Password must
32       contain at least one special character'
33     ]
34   };
35 }
```

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    } catch (error) {
88      console.error(error);
89    }
90  }
91
```

```

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
126     // Function to get custom alert class based on message type
127     const getAlertClass = () => {
128         if (isError.value) return 'login-error-message';
129         if (isSuccess.value) return 'login-success-message';
130         if (isWelcome.value) return 'welcome-message-elegant'; // Use existing welcome
131         styling
132         return '';
133     };

```

6. Password Visibility Toggle

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

```

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

```

7. Navigation to Registration

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

```

121
122     const redirectToRegister = () => {
123         window.location.href = 'registration.php';
124     };

```

8. Template for Login Page

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

```
150 template: ` 151 <v-app> 152   <v-main style="padding: 0 !important;"> 153     <div class="login-wrapper"> 154       <!-- Left side - Login Form --> 155       <div class="login-form-section"> 156         <div class="login-container"> 157           <h1 class="login-title">Login</h1> 158           <!-- Custom Alert for messages --> 159           <div v-if="message" :class="getAlertClass()" class="alert-message"> 160             {{ message }} 161           </div> 162           <v-form ref="formRef" lazy-validation @keyup.enter="loginUser" 163             class="login-form"> 164             <div class="form-group"> 165               <label for="email" class="form-label">EMAIL</label> 166               <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"> 167                 </v-text-field> 168             </div> 169             <div class="form-group"> 170               <label for="password" class="form-label">PASSWORD</label> 171               <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"> 172                 </v-text-field> 173             </div> 174             <v-btn @click="loginUser" :loading="isSubmitting" :disabled="isSubmitting" class="btn-brown large-btn" block> 175               <span v-if="isSubmitting">Signing In...</span> 176               <span v-else>Continue ></span> 177             </v-btn> 178           </v-form> 179         </div> 180       </div> 181     </div> 182   </v-main> 183 </v-app> 184 </div> 185 </div> 186 </div> 187 </div> 188 </div> 189 </div> 190 </div> 191 </div> 192 </div> 193 </div> 194 </div> 195 </div> 196 </div> 197 </div> 198 </div> 199 </div> 200 </div> 201 </div> 202 </div> 203 </div> 204 </div> 205 </div> 206 </div> 207 </div> 208 </div> 209 </div> 210 </div> 211 </div> 212 </div> 213 </div> 214 </div>
```

```

215 <!-- Right side - Welcome/Register Section -->
216 <div class="welcome-section">
217   <div class="welcome-content">
218     <h1 class="welcome-title">Join Our Community</h1>
219     <p class="welcome-text">Don't have an account?</p>
220     <v-btn
221       @click="redirectToRegister"
222       class="btn-brown large-btn signup-btn">
223       Sign Up
224     </v-btn>
225   </div>
226 </div>
227 </v-main>
228 </v-app>
229
230
231

```

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.

```

1  <?php
2   session_start();

```

2. Clear Session Data

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

```

4   // Clear all session variables
5   $_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.

```

7   // Destroy the session cookie
8   if (isset($_COOKIE[session_name()])) {
9     setcookie(session_name(), '', time() - 42000, '/');
10

```

4. Destroy the Session

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

```

12  // Destroy the session
13  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.

```

15  // Redirect to login page
16  header('Location: login.php');
17  exit;
18 ?>

```

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.

```
9      // Decode JSON input
10     $input = json_decode(file_get_contents('php://input'), true);
11     if (!$input || !isset($input['name'], $input['email'], $input['password'])) {
12         http_response_code(400);
13         echo json_encode(['message' => 'Invalid input']);
14         exit;
15     }
16
17     // Extract values
18     $name = $input['name'];
19     $email = $input['email'];
20     $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).

```
22    // Connect to MySQL
23    $conn = mysqli_connect('localhost', 'root', '', 'project');
24    if (!$conn) {
25        http_response_code(500);
26        echo json_encode(['message' => 'Database connection failed']);
27        exit;
28    }
29
30    // Ensure UTF-8 encoding
31    mysqli_set_charset($conn, 'utf8');
32
33    // Check if email already exists
34    $stmt = $conn->prepare("SELECT id FROM users WHERE email = ?");
35    $stmt->bind_param("s", $email);
36    $stmt->execute();
37    $stmt->store_result();
38
39    if ($stmt->num_rows > 0) {
40        http_response_code(409);
41        echo json_encode(['message' => 'Email already registered']);
42        exit;
43    }
```

3. Password Hashing

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

```
45      // Hash the password
46      $passwordHash = password_hash($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.

```
48     // Insert new user
49     $stmt = $conn->prepare("INSERT INTO users (name, email, password) VALUES (?, ?, ?)");
50     $stmt->bind_param("sss", $name, $email, $passwordHash);
51
52     if ($stmt->execute()) {
53         http_response_code(201);
54         echo json_encode(['message' => 'User registered successfully', 'redirectUrl' =>
55                         'login.php']);
56     } else {
57         http_response_code(500);
58         echo json_encode(['message' => 'Failed to register user']);
59     }
60
61     // Clean up
62     $stmt->close();
63     mysqli_close($conn);
64     exit;
65 }
```

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.

```
6   createApp({
7     setup() {
8       const name = ref('');
9       const email = ref('');
10      const password = ref('');
11      const confirmPassword = ref('');
12      const message = ref('');
13      const isSuccess = ref(false);
14      const isError = ref(false);
15      const isWelcome = ref(false);
16      const formRef = ref(null);
17      const isSubmitting = ref(false);
18      const showPassword = ref(false); // New ref for password visibility
19      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.

```

21      const rules = {
22        name: [
23          (v) => !!v || 'Name is required',
24          (v) => /^[A-Za-z\s]+$/ .test(v) || 'Only letters allowed'
25        ],
26
27        email: [
28          (v) => !!v || 'Email is required',
29          (v) => /^[^@\s]+@[^\s]+\.[^\s]+$/ .test(v) || 'Invalid email'
30        ],
31
32        password: [
33          (v) => !!v || 'Password is required',
34          (v) => v.length >= 8 || 'Password must be at least 8 characters',
35          (v) => /[A-Z]/ .test(v) || 'Password must contain at least one uppercase letter',
36          (v) => /[0-9]/ .test(v) || 'Password must contain at least one number',
37          (v) => /[^!@#$%^&*()_+=\{\}:\":\\|.,<>\\/?]/ .test(v) || 'Password must contain at least one special character',
38        ],
39
40        confirmPassword: [
41          (v) => !!v || 'Confirm Password is required',
42          (v) => v === password.value || 'Passwords do not match'
43        ],
44      };

```

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.

```

57      const registerUser = async () => {
58        const { valid } = await formRef.value.validate();
59        if (!valid) {
60          isError.value = true;
61          isSuccess.value = false;
62          isWelcome.value = false;
63          message.value = "Please correct all form errors before submitting.";
64          // Add shake animation to form
65          const registrationContainer = document.querySelector('.login-container');
66          if (registrationContainer) {
67            registrationContainer.classList.add('error-shake');
68            setTimeout(() => {
69              registrationContainer.classList.remove('error-shake');
70            }, 500);
71          }
72          return;
73        }
74
75        isSuccess.value = false;
76        isError.value = false;
77        isWelcome.value = false;
78        message.value = '';
79
80        try {
81          isSubmitting.value = true;
82          const response = await fetch("registration.php", {
83            method: "POST",
84            headers: { "Content-Type": "application/json" },
85            body: JSON.stringify({
86              name: name.value,
87              email: email.value,
88              password: password.value
89            })
90          });

```

```

92         const result = await response.json();
93
94         if (response.ok) {
95             message.value = result.message;
96             isSuccess.value = true;
97
98             // After 1 second, replace with welcome message and change styling
99             setTimeout(() => {
100                 message.value = `Welcome, ${name.value}!`;
101                 // Change to welcome styling
102                 isSuccess.value = false;
103                 isError.value = false;
104                 isWelcome.value = true;
105
106                 clearForm();
107                 formRef.value.resetValidation();
108
109                 // After another 2 seconds, redirect
110                 setTimeout(() => {
111                     if (result.redirectUrl) {
112                         window.location.href = result.redirectUrl;
113                     } else {
114                         window.location.href = 'login.php';
115                     }
116                 }, 2000);
117             }, 1000);
118
119         } else {
120             message.value = result.message || "Registration failed.";
121             isError.value = true;
122             isSuccess.value = false;
123             isWelcome.value = false;
124
125             // Add shake animation to form on error
126             const registrationContainer = document.querySelector('.login-container');
127
128             if (registrationContainer) {
129                 registrationContainer.classList.add('error-shake');
130                 setTimeout(() => {
131                     registrationContainer.classList.remove('error-shake');
132                 }, 500);
133             }
134         }
135     } catch (error) {
136         message.value = "Network error: " + error.message;
137         isError.value = true;
138         isSuccess.value = false;
139         isWelcome.value = false;
140
141         // Add shake animation to form on network error
142         const registrationContainer = document.querySelector('.login-container');
143         if (registrationContainer) {
144             registrationContainer.classList.add('error-shake');
145             setTimeout(() => {
146                 registrationContainer.classList.remove('error-shake');
147             }, 500);
148         }
149     } finally {
150         isSubmitting.value = false;
151     }
152 };

```

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.

```

153         // Function to get custom alert class based on message type
154         const getAlertClass = () => {
155             if (isError.value) return 'login-error-message';
156             if (isSuccess.value) return 'login-success-message';
157             if (isWelcome.value) return 'welcome-message-elegant';
158             return '';
159         };

```

5. Password Visibility Toggle

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

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

6. Navigation to Login

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

```
53 const redirectToLogin = () => {  
54   window.location.href = 'login.php';  
55 };
```

7. Template for Registration Page

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

```
180 template: `  
181 <v-app>  
182   <v-main style="padding: 0 !important;">  
183     <div class="login-wrapper">  
184       <!-- Left side - Registration Form -->  
185       <div class="login-form-section">  
186         <div class="login-container">  
187           <h1 class="login-title">Registration</h1>  
188  
189           <!-- Custom Alert for messages -->  
190           <div v-if="message" :class="getAlertClass()" class="alert-message">  
191             {{ message }}  
192           </div>  
193  
194           <v-form ref="formRef" lazy-validation  
195             @keyup.enter="registerUser" class="login-form">  
196             <div class="form-group">  
197               <label for="name" class="form-label">NAME</label>  
198               <v-text-field  
199                 v-model="name"  
200                 :rules="rules.name"  
201                 required  
202                 variant="outlined"  
203                 density="comfortable"  
204                 placeholder="Enter your full name"  
205                 hide-details="auto"  
206                 class="large-input"  
207               ></v-text-field>  
208             </div>  
209  
210           <div class="form-group">  
211             <label for="email" class="form-label">EMAIL</label>  
212             <v-text-field  
213               v-model="email"  
214               :rules="rules.email"  
215               type="email"  
216               required  
217               variant="outlined"  
218               density="comfortable"  
219               placeholder="Enter your email"  
220               hide-details="auto"  
221               class="large-input"  
222             ></v-text-field>  
223           </div>  
224         </div>  
225       </div>  
226     </div>
```

```

228             <div class="form-group">
229                 <label for="password" class="form-label">PASSWORD</label>
230                 <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>
231             </div>
232
233             <div class="form-group">
234                 <label for="confirmPassword" class="form-label">CONFIRM PASSWORD</label>
235                 <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>
236             </div>
237
238             <v-btn @click="registerUser" :loading="isSubmitting" :disabled="isSubmitting" class="btn-brown large-btn" block>
239                 >
240                     <span v-if="isSubmitting">Creating Account...</span>
241                     <span v-else>Continue </span>
242                 </v-btn>
243             </v-form>
244         </div>
245     </div>
246
247     <!-- Right side - Welcome/Login Section -->
248     <div class="welcome-section">
249         <div class="welcome-content">
250             <h1 class="welcome-title">Welcome to Registration</h1>
251             <p class="welcome-text">Already have an account?</p>
252             <v-btn @click="redirectToLogin" class="btn-brown large-btn signup-btn" >
253                 >
254                     Sign In
255                 </v-btn>
256             </div>
257         </div>
258     </v-main>
259
260     </v-app>
261
262     <script>
263         window.addEventListener('load', () => {
264             // Your code here
265         });
266     </script>
267
268     <script>
269         // Your code here
270     </script>
271
272     <script>
273         // Your code here
274     </script>
275
276     <script>
277         // Your code here
278     </script>
279
280     <script>
281         // Your code here
282     </script>
283
284     <script>
285         // Your code here
286     </script>
287
288     <script>
289         // Your code here
290     </script>
291
292     <script>
293         // Your code here
294     </script>
295
296     <script>
297         // Your code here
298     </script>
299
300     <script>
301         // Your code here
302     </script>
303
304     <script>
305         // Your code here
306     </script>
307
308     <script>
309         // Your code here
310     </script>
311
312     <script>
313         // Your code here
314     </script>
315
316     <script>
317         // Your code here
318     </script>
319
320     <script>
321         // Your code here
322     </script>
323
324     <script>
325         // Your code here
326     </script>
327
328     <script>
329         // Your code here
330     </script>
331
332     <script>
333         // Your code here
334     </script>
335
336     <script>
337         // Your code here
338     </script>
339
340     <script>
341         // Your code here
342     </script>
343
344     <script>
345         // Your code here
346     </script>
347
348     <script>
349         // Your code here
350     </script>
351
352     <script>
353         // Your code here
354     </script>
355
356     <script>
357         // Your code here
358     </script>
359
360     <script>
361         // Your code here
362     </script>
363
364     <script>
365         // Your code here
366     </script>
367
368     <script>
369         // Your code here
370     </script>
371
372     <script>
373         // Your code here
374     </script>
375
376     <script>
377         // Your code here
378     </script>
379
380     <script>
381         // Your code here
382     </script>
383
384     <script>
385         // Your code here
386     </script>
387
388     <script>
389         // Your code here
390     </script>
391
392     <script>
393         // Your code here
394     </script>
395
396     <script>
397         // Your code here
398     </script>
399
399     <script>
400         // Your code here
401     </script>
402
403     <script>
404         // Your code here
405     </script>
406
407     <script>
408         // Your code here
409     </script>
410
411     <script>
412         // Your code here
413     </script>
414
415     <script>
416         // Your code here
417     </script>
418
419     <script>
420         // Your code here
421     </script>
422
423     <script>
424         // Your code here
425     </script>
426
427     <script>
428         // Your code here
429     </script>
430
431     <script>
432         // Your code here
433     </script>
434
435     <script>
436         // Your code here
437     </script>
438
439     <script>
440         // Your code here
441     </script>
442
443     <script>
444         // Your code here
445     </script>
446
447     <script>
448         // Your code here
449     </script>
450
451     <script>
452         // Your code here
453     </script>
454
455     <script>
456         // Your code here
457     </script>
458
459     <script>
460         // Your code here
461     </script>
462
463     <script>
464         // Your code here
465     </script>
466
467     <script>
468         // Your code here
469     </script>
470
471     <script>
472         // Your code here
473     </script>
474
475     <script>
476         // Your code here
477     </script>
478
479     <script>
480         // Your code here
481     </script>
482
483     <script>
484         // Your code here
485     </script>
486
487     <script>
488         // Your code here
489     </script>
490
491     <script>
492         // Your code here
493     </script>
494
495     <script>
496         // Your code here
497     </script>
498
499     <script>
500         // Your code here
501     </script>
502
503     <script>
504         // Your code here
505     </script>
506
507     <script>
508         // Your code here
509     </script>
510
511     <script>
512         // Your code here
513     </script>
514
515     <script>
516         // Your code here
517     </script>
518
519     <script>
520         // Your code here
521     </script>
522
523     <script>
524         // Your code here
525     </script>
526
527     <script>
528         // Your code here
529     </script>
530
531     <script>
532         // Your code here
533     </script>
534
535     <script>
536         // Your code here
537     </script>
538
539     <script>
540         // Your code here
541     </script>
542
543     <script>
544         // Your code here
545     </script>
546
547     <script>
548         // Your code here
549     </script>
550
551     <script>
552         // Your code here
553     </script>
554
555     <script>
556         // Your code here
557     </script>
558
559     <script>
560         // Your code here
561     </script>
562
563     <script>
564         // Your code here
565     </script>
566
567     <script>
568         // Your code here
569     </script>
570
571     <script>
572         // Your code here
573     </script>
574
575     <script>
576         // Your code here
577     </script>
578
579     <script>
580         // Your code here
581     </script>
582
583     <script>
584         // Your code here
585     </script>
586
587     <script>
588         // Your code here
589     </script>
590
591     <script>
592         // Your code here
593     </script>
594
595     <script>
596         // Your code here
597     </script>
598
599     <script>
600         // Your code here
601     </script>
602
603     <script>
604         // Your code here
605     </script>
606
607     <script>
608         // Your code here
609     </script>
610
611     <script>
612         // Your code here
613     </script>
614
615     <script>
616         // Your code here
617     </script>
618
619     <script>
620         // Your code here
621     </script>
622
623     <script>
624         // Your code here
625     </script>
626
627     <script>
628         // Your code here
629     </script>
630
631     <script>
632         // Your code here
633     </script>
634
635     <script>
636         // Your code here
637     </script>
638
639     <script>
640         // Your code here
641     </script>
642
643     <script>
644         // Your code here
645     </script>
646
647     <script>
648         // Your code here
649     </script>
650
651     <script>
652         // Your code here
653     </script>
654
655     <script>
656         // Your code here
657     </script>
658
659     <script>
660         // Your code here
661     </script>
662
663     <script>
664         // Your code here
665     </script>
666
667     <script>
668         // Your code here
669     </script>
670
671     <script>
672         // Your code here
673     </script>
674
675     <script>
676         // Your code here
677     </script>
678
679     <script>
680         // Your code here
681     </script>
682
683     <script>
684         // Your code here
685     </script>
686
687     <script>
688         // Your code here
689     </script>
690
691     <script>
692         // Your code here
693     </script>
694
695     <script>
696         // Your code here
697     </script>
698
699     <script>
700         // Your code here
701     </script>
702
703     <script>
704         // Your code here
705     </script>
706
707     <script>
708         // Your code here
709     </script>
710
711     <script>
712         // Your code here
713     </script>
714
715     <script>
716         // Your code here
717     </script>
718
719     <script>
720         // Your code here
721     </script>
722
723     <script>
724         // Your code here
725     </script>
726
727     <script>
728         // Your code here
729     </script>
730
731     <script>
732         // Your code here
733     </script>
734
735     <script>
736         // Your code here
737     </script>
738
739     <script>
740         // Your code here
741     </script>
742
743     <script>
744         // Your code here
745     </script>
746
747     <script>
748         // Your code here
749     </script>
750
751     <script>
752         // Your code here
753     </script>
754
755     <script>
756         // Your code here
757     </script>
758
759     <script>
760         // Your code here
761     </script>
762
763     <script>
764         // Your code here
765     </script>
766
767     <script>
768         // Your code here
769     </script>
770
771     <script>
772         // Your code here
773     </script>
774
775     <script>
776         // Your code here
777     </script>
778
779     <script>
780         // Your code here
781     </script>
782
783     <script>
784         // Your code here
785     </script>
786
787     <script>
788         // Your code here
789     </script>
790
791     <script>
792         // Your code here
793     </script>
794
795     <script>
796         // Your code here
797     </script>
798
799     <script>
800         // Your code here
801     </script>
802
803     <script>
804         // Your code here
805     </script>
806
807     <script>
808         // Your code here
809     </script>
810
811     <script>
812         // Your code here
813     </script>
814
815     <script>
816         // Your code here
817     </script>
818
819     <script>
820         // Your code here
821     </script>
822
823     <script>
824         // Your code here
825     </script>
826
827     <script>
828         // Your code here
829     </script>
830
831     <script>
832         // Your code here
833     </script>
834
835     <script>
836         // Your code here
837     </script>
838
839     <script>
840         // Your code here
841     </script>
842
843     <script>
844         // Your code here
845     </script>
846
847     <script>
848         // Your code here
849     </script>
850
851     <script>
852         // Your code here
853     </script>
854
855     <script>
856         // Your code here
857     </script>
858
859     <script>
860         // Your code here
861     </script>
862
863     <script>
864         // Your code here
865     </script>
866
867     <script>
868         // Your code here
869     </script>
870
871     <script>
872         // Your code here
873     </script>
874
875     <script>
876         // Your code here
877     </script>
878
879     <script>
880         // Your code here
881     </script>
882
883     <script>
884         // Your code here
885     </script>
886
887     <script>
888         // Your code here
889     </script>
890
891     <script>
892         // Your code here
893     </script>
894
895     <script>
896         // Your code here
897     </script>
898
899     <script>
900         // Your code here
901     </script>
902
903     <script>
904         // Your code here
905     </script>
906
907     <script>
908         // Your code here
909     </script>
910
911     <script>
912         // Your code here
913     </script>
914
915     <script>
916         // Your code here
917     </script>
918
919     <script>
920         // Your code here
921     </script>
922
923     <script>
924         // Your code here
925     </script>
926
927     <script>
928         // Your code here
929     </script>
930
931     <script>
932         // Your code here
933     </script>
934
935     <script>
936         // Your code here
937     </script>
938
939     <script>
940         // Your code here
941     </script>
942
943     <script>
944         // Your code here
945     </script>
946
947     <script>
948         // Your code here
949     </script>
950
951     <script>
952         // Your code here
953     </script>
954
955     <script>
956         // Your code here
957     </script>
958
959     <script>
960         // Your code here
961     </script>
962
963     <script>
964         // Your code here
965     </script>
966
967     <script>
968         // Your code here
969     </script>
970
971     <script>
972         // Your code here
973     </script>
974
975     <script>
976         // Your code here
977     </script>
978
979     <script>
980         // Your code here
981     </script>
982
983     <script>
984         // Your code here
985     </script>
986
987     <script>
988         // Your code here
989     </script>
990
991     <script>
992         // Your code here
993     </script>
994
995     <script>
996         // Your code here
997     </script>
998
999     <script>
1000        // Your code here
1001    </script>
1002
1003    <script>
1004        // Your code here
1005    </script>
1006
1007    <script>
1008        // Your code here
1009    </script>
1010
1011    <script>
1012        // Your code here
1013    </script>
1014
1015    <script>
1016        // Your code here
1017    </script>
1018
1019    <script>
1020        // Your code here
1021    </script>
1022
1023    <script>
1024        // Your code here
1025    </script>
1026
1027    <script>
1028        // Your code here
1029    </script>
1030
1031    <script>
1032        // Your code here
1033    </script>
1034
1035    <script>
1036        // Your code here
1037    </script>
1038
1039    <script>
1040        // Your code here
1041    </script>
1042
1043    <script>
1044        // Your code here
1045    </script>
1046
1047    <script>
1048        // Your code here
1049    </script>
1050
1051    <script>
1052        // Your code here
1053    </script>
1054
1055    <script>
1056        // Your code here
1057    </script>
1058
1059    <script>
1060        // Your code here
1061    </script>
1062
1063    <script>
1064        // Your code here
1065    </script>
1066
1067    <script>
1068        // Your code here
1069    </script>
1070
1071    <script>
1072        // Your code here
1073    </script>
1074
1075    <script>
1076        // Your code here
1077    </script>
1078
1079    <script>
1080        // Your code here
1081    </script>
1082
1083    <script>
1084        // Your code here
1085    </script>
1086
1087    <script>
1088        // Your code here
1089    </script>
1090
1091    <script>
1092        // Your code here
1093    </script>
1094
1095    <script>
1096        // Your code here
1097    </script>
1098
1099    <script>
1100       // Your code here
1101    </script>
1102
1103    <script>
1104       // Your code here
1105    </script>
1106
1107    <script>
1108       // Your code here
1109    </script>
1110
1111    <script>
1112       // Your code here
1113    </script>
1114
1115    <script>
1116       // Your code here
1117    </script>
1118
1119    <script>
1120       // Your code here
1121    </script>
1122
1123    <script>
1124       // Your code here
1125    </script>
1126
1127    <script>
1128       // Your code here
1129    </script>
1130
1131    <script>
1132       // Your code here
1133    </script>
1134
1135    <script>
1136       // Your code here
1137    </script>
1138
1139    <script>
1140       // Your code here
1141    </script>
1142
1143    <script>
1144       // Your code here
1145    </script>
1146
1147    <script>
1148       // Your code here
1149    </script>
1150
1151    <script>
1152       // Your code here
1153    </script>
1154
1155    <script>
1156       // Your code here
1157    </script>
1158
1159    <script>
1160       // Your code here
1161    </script>
1162
1163    <script>
1164       // Your code here
1165    </script>
1166
1167    <script>
1168       // Your code here
1169    </script>
1170
1171    <script>
1172       // Your code here
1173    </script>
1174
1175    <script>
1176       // Your code here
1177    </script>
1178
1179    <script>
1180       // Your code here
1181    </script>
1182
1183    <script>
1184       // Your code here
1185    </script>
1186
1187    <script>
1188       // Your code here
1189    </script>
1190
1191    <script>
1192       // Your code here
1193    </script>
1194
1195    <script>
1196       // Your code here
1197    </script>
1198
1199    <script>
1200       // Your code here
1201    </script>
1202
1203    <script>
1204       // Your code here
1205    </script>
1206
1207    <script>
1208       // Your code here
1209    </script>
1210
1211    <script>
1212       // Your code here
1213    </script>
1214
1215    <script>
1216       // Your code here
1217    </script>
1218
1219    <script>
1220       // Your code here
1221    </script>
1222
1223    <script>
1224       // Your code here
1225    </script>
1226
1227    <script>
1228       // Your code here
1229    </script>
1230
1231    <script>
1232       // Your code here
1233    </script>
1234
1235    <script>
1236       // Your code here
1237    </script>
1238
1239    <script>
1240       // Your code here
1241    </script>
1242
1243    <script>
1244       // Your code here
1245    </script>
1246
1247    <script>
1248       // Your code here
1249    </script>
1250
1251    <script>
1252       // Your code here
1253    </script>
1254
1255    <script>
1256       // Your code here
1257    </script>
1258
1259    <script>
1260       // Your code here
1261    </script>
1262
1263    <script>
1264       // Your code here
1265    </script>
1266
1267    <script>
1268       // Your code here
1269    </script>
1270
1271    <script>
1272       // Your code here
1273    </script>
1274
1275    <script>
1276       // Your code here
1277    </script>
1278
1279    <script>
1280       // Your code here
1281    </script>
1282
1283    <script>
1284       // Your code here
1285    </script>
1286
1287    <script>
1288       // Your code here
1289    </script>
1290
1291    <script>
1292       // Your code here
1293    </script>
1294
1295    <script>
1296       // Your code here
1297    </script>
1298
1299    <script>
1300       // Your code here
1301    </script>
1302
1303    <script>
1304       // Your code here
1305    </script>
1306
1307    <script>
1308       // Your code here
1309    </script>
1310
1311    <script>
1312       // Your code here
1313    </script>
1314
1315    <script>
1316       // Your code here
1317    </script>
1318
1319    <script>
1320       // Your code here
1321    </script>
1322
1323    <script>
1324       // Your code here
1325    </script>
1326
1327    <script>
1328       // Your code here
1329    </script>
1330
1331    <script>
1332       // Your code here
1333    </script>
1334
1335    <script>
1336       // Your code here
1337    </script>
1338
1339    <script>
1340       // Your code here
1341    </script>
1342
1343    <script>
1344       // Your code here
1345    </script>
1346
1347    <script>
1348       // Your code here
1349    </script>
1350
1351    <script>
1352       // Your code here
1353    </script>
1354
1355    <script>
1356       // Your code here
1357    </script>
1358
1359    <script>
1360       // Your code here
1361    </script>
1362
1363    <script>
1364       // Your code here
1365    </script>
1366
1367    <script>
1368       // Your code here
1369    </script>
1370
1371    <script>
1372       // Your code here
1373    </script>
1374
1375    <script>
1376       // Your code here
1377    </script>
1378
1379    <script>
1380       // Your code here
1381    </script>
1382
1383    <script>
1384       // Your code here
1385    </script>
1386
1387    <script>
1388       // Your code here
1389    </script>
1390
1391    <script>
1392       // Your code here
1393    </script>
1394
1395    <script>
1396       // Your code here
1397    </script>
1398
1399    <script>
1400       // Your code here
1401    </script>
1402
1403    <script>
1404       // Your code here
1405    </script>
1406
1407    <script>
1408       // Your code here
1409    </script>
1410
1411    <script>
1412       // Your code here
1413    </script>
1414
1415    <script>
1416       // Your code here
1417    </script>
1418
1419    <script>
1420       // Your code here
1421    </script>
1422
1423    <script>
1424       // Your code here
1425    </script>
1426
1427    <script>
1428       // Your code here
1429    </script>
1430
1431    <script>
1432       // Your code here
1433    </script>
1434
1435    <script>
1436       // Your code here
1437    </script>
1438
1439    <script>
1440       // Your code here
1441    </script>
1442
1443    <script>
1444       // Your code here
1445    </script>
1446
1447    <script>
1448       // Your code here
1449    </script>
1450
1451    <script>
1452       // Your code here
1453    </script>
1454
1455    <script>
1456       // Your code here
1457    </script>
1458
1459    <script>
1460       // Your code here
1461    </script>
1462
1463    <script>
1464       // Your code here
1465    </script>
1466
1467    <script>
1468       // Your code here
1469    </script>
1470
1471    <script>
1472       // Your code here
1473    </script>
1474

```

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.

```
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://cdn.jsdelivr.net/npm/@fortawesome/fontawesome-free@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>
18  <!-- Load Bootstrap JS first -->
19  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>
20
21  <!-- Load Vue and custom cart script -->
22  <script src="https://unpkg.com/vue@3/dist/vue.global.prod.js"></script>
23  <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.



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:
27   url('images/banner1.jpeg'); background-size: cover;">
28   <h1 class="display-4">Handmade Crochet with Love</h1>
</section>
```

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

```

30      <!-- Categories Overview Section -->
31      <section class="container mt-5">
32          <div class="row mb-4">
33              <div class="col-12 text-center">
34                  <h2>Shop by Category</h2>
35                  <p class="text-muted">Explore our handcrafted collections</p>
36              </div>
37          </div>
38
39          <!-- Category Grid - Replace your existing category grid section with this -->
40          <div class="row g-4 mb-5 justify-content-center">
41              <div v-for="category in visibleCategories" :key="category.id" class=
42                  "col-lg-2 col-md-4 col-sm-6 col-6">
43                  <div class="card index_category h-100"
44                      :class="category.id"
45                      @click="showCategoryInfo(category)"
46                      tabindex="0"
47                      @keydown.enter="showCategoryInfo(category)"
48                      @keydown.space="showCategoryInfo(category)">
49                      <div class="category_card-body">
50                          <div>{{ category.icon }}</div>
51                          <h5 class="card-title">{{ category.name }}</h5>
52                          <p class="text-muted small">{{ getCategoryCount(category.id) }} items available</p>
53                      </div>
54                  </div>
55              </div>
56          </div>

```

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

```

58      <!-- Best Sellers Section -->
59      <section class="container mt-5">
60          <div class="row mb-4">
61              <div class="col-md-8">
62                  <h2><i class="fas fa-star text-warning me-2"></i>Best Sellers</h2>
63                  <p class="text-muted">Our most popular handcrafted items</p>
64              </div>
65
66          <div class="row g-4 mb-5">
67              <div
68                  v-for="(product, index) in products.slice(0, 3)"
69                  :key="product.id"
70                  class="col-lg-4 col-md-6">
71                  >
72                      <div class="card product-card h-100">
73                          <div class="position-relative">
74                              
79                              <span class="badge bg-warning position-absolute top-0 end-0 m-2"
80                                  >
81                                  #{{ index + 1 }}
82                              </span>
83                      </div>
84                      <div class="card-body">
85                          <h5 class="card-title">{{ product.name }}</h5>
86                          <p class="card-text text-muted small">{{ truncateText(product.description, 80) }}</p>
87                          <div class="d-flex justify-content-between align-items-center">
88                              <span class="price-badge">${{ formatPrice(product.price) }}</span>
89                              <small class="text-muted">{{ getProductCategory(product) }}</small>
90                          </div>
91                      </div>
92                      <div class="card-footer bg-transparent">
93                          <button class="btn btn-custom w-100" @click=
94                              "openProductModal(product)">
95                              <i class="fas fa-eye icon-spacing"></i>View Details
96                          </button>
97                      </div>
98                  </div>
99              </div>
100         </div>

```

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

```

102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144

<!-- 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">
          
          <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.

```

146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166

<!-- 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.

```

168      <!-- Enhanced Product Detail Modal -->
169      <div class="modal fade" id="productModal" tabindex="-1" aria-labelledby=
170        "productModalLabel" aria-hidden="true">
171          <div v-if="message" class="alert alert-success position-fixed top-0 end-0 m-3"
172            style="z-index: 9999;">
173              {{ message }}
174            </div>
175            <div class="modal-dialog modal-lg modal-dialog-centered">
176              <div class="modal-content custom-card" v-if="selectedProduct">
177                <div class="modal-header">
178                  <h5 class="modal-title">{{ selectedProduct.name }}</h5>
179                  <button type="button" class="btn-close" data-bs-dismiss="modal"
180                    ></button>
181                </div>
182                <div class="modal-body">
183                  <div class="row">
184                    <div class="col-md-6">
185                      
187                    </div>
188                    <div class="col-md-6">
189                      <h4 class="text-success mb-3">${{ formatPrice(selectedProduct.price) }}</h4>
190                      <p>{{ selectedProduct.description }}</p>
191                      <hr>
192                      <div class="row">
193                        <div class="col-6">
194                          <p><strong>Category:</strong><br>{{ getProductCategory(selectedProduct) }}</p>
195                        </div>
196                        <div class="col-6">
197                          <p><strong>Status:</strong><br><span class="badge
198                            bg-success">In Stock</span></p>
199                        </div>
200                      </div>
201                      <div class="modal-footer">
202                        <button type="button" class="btn profile-btn cancel-btn"
203                          data-bs-dismiss="modal">Close</button>
204                        <button type="button" class="btn profile-btn delete-btn" @click=
205                          "addToCart(selectedProduct)">
206                          <i class="fas fa-cart-plus icon-spacing"></i>Add to Cart
207                        </button>
208                      </div>
209                    </div>
210                  </div>
211                </div>
212              </div>
213            </div>
214          </div>
215        </div>
216      </div>
217    </div>
218  </div>
219
```

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.

```
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
    },  

    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=....

```
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
    },  

    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: '>All' },
          { 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
45 // Get featured products (first 3)
46 featuredProducts() {
47     return this.products.slice(0, 3);
48 },
49
50 // Get new arrivals (next 3 products)
51 newArrivals() {
52     return this.products.slice(3, 6);
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().

```
98      addToCart(product) {
99        fetch('cart_api.php', {
100          method: 'POST',
101          headers: {
102            'Content-Type': 'application/json'
103          },
104          body: JSON.stringify({
105            id: product.id,
106            name: product.name,
107            price: product.price,
108            image: product.image
109          })
110        })
111        .then(response => response.json())
112        .then(data => {
113          if (data.success) {
114            this.showMessage(data.message || 'Added to cart!', true);
115          } else {
116            this.showMessage(data.message || 'Failed to add to cart.', false);
117          }
118        })
119        .catch(error => {
120          console.error('Error adding to cart:', error);
121          this.showMessage('Error adding to cart.', false);
122        });
123      },
124
125      showMessage(text) {
126        this.message = text;
127        setTimeout(() => {
128          this.message = '';
129        }, 2000);
130      },
131    }
```

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.

```
160      // Enhanced product categorization helper methods
161      getProductCategoryId(product) {
162        const name = product.name.toLowerCase();
163        const desc = product.description.toLowerCase();
164
165        // Baby Items
166        if (name.includes('baby') || name.includes('booties') || desc.includes('nursery
167        decor') || desc.includes('baby's feet')) {
168          return 'baby';
169        }
170        // Seasonal Items (e.g., Christmas)
171        else if (name.includes('christmas') || name.includes('holiday') || name.includes
172        ('stocking')) {
173          return 'seasonal';
174        }
175      }
```

```

173     // Home Decor
174     else if (name.includes('pillow') || name.includes('plant') ||
175             name.includes('pot cover') || desc.includes('home decor')) {
176         return 'home';
177     }
178     // Clothing
179     else if (name.includes('hat') || name.includes('scarf') || name.includes(
180             'headband') ||
181             name.includes('sweater')) { // Removed 'cozy' from clothing
182                 for now
183             return 'clothing';
184         }
185     // Accessories (This is the default/fallback if no other category matches)
186     // Including "Soft Crochet Blanket" and "Crochet Coffee Cup Cozy" here
187     else {
188         return 'accessories';
189     }
190 }
191
192 getProductCategory(product) {
193     const categoryId = this.getProductCategoryId(product);
194     const category = this.categories.find(cat => cat.id === categoryId);
195     return category ? category.name : 'Accessories';
196 }
197
198 filterProductsByCategory(categoryId) {
199     if (categoryId === 'all') {
200         return this.products;
201     }
202     return this.products.filter(product =>
203         this.getProductCategoryId(product) === categoryId
204     );
}

```

7. Utility Methods

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

```

206     // Format helpers
207     formatPrice(price) {
208         return parseFloat(price).toFixed(2);
209     },

```

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

```

211     truncateText(text, maxLength) {
212         if (!text) return '';
213         if (text.length <= maxLength) return text;
214         return text.substring(0, maxLength).trim() + '...';
215     },

```

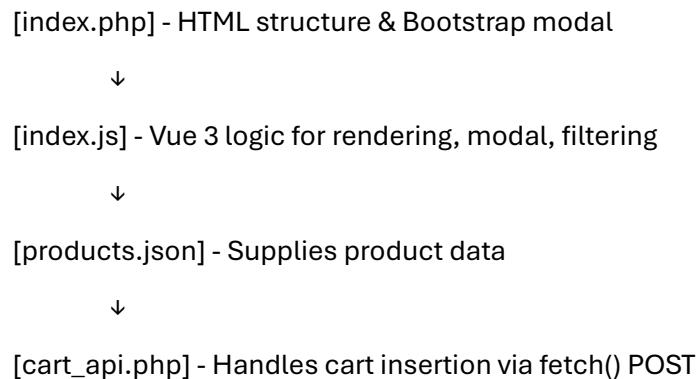
- `getCategoryStats()` → Builds a summary of product counts per category (for debugging or future UI use).

```

217     // Get category statistics for display
218     getCategoryStats() {
219         const stats = {};
220         this.categories.forEach(category => {
221             if (category.id !== 'all') {
222                 stats[category.id] = {
223                     name: category.name,
224                     count: this.getCategoryCount(category.id),
225                     icon: category.icon
226                 };
227             }
228         });
229         return stats;
230     },
231

```

Workflow Overview:



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 ===
37          category.id }]"
38        @click="selectCategory(category.id)"
39      >
40        {{ category.icon }} {{ category.name }}
41        <span class="badge bg-light text-dark ms-1">{{ category.count }}</span>
42      </button>
43    </div>

```

5. Advanced Filters

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

- **Price range** (dropdown).

```
75 <div class="col-md-2">
76     <label class="form-label">
77         <i class="fas fa-dollar-sign icon-spacing"></i>Price Range
78     </label>
79     <select class="form-select" v-model="priceRange" @change=
80         "resetPagination">
81         <option v-for="range in priceRanges" :key="range.value" :value=
82             "range.value">
83             {{ range.label }}
84         </option>
85     </select>
86 </div>
```

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

```
85 <div class="col-md-2">
86   <label class="form-label">
87     <i class="fas fa-sort icon-spacing"></i>Sort By
88   </label>
89   <select class="form-select" v-model="sortBy">
90     <option v-for="option in sortOptions" :key="option.value" :value=
91       "option.value">
92       {{ option.label }}
93     </option>
94   </select>
95 </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=
98              "resetPagination">
99              <option v-for="size in pageSizes" :key="size" :value="size">{{ size
100             }}</option>
101         </select>
102     </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=
125                  "searchQuery = ''" style="cursor: pointer;"></i>
126          </span>
127          <span v-if="selectedCategory !== 'all'" class="badge bg-success me-2">
128              Category: {{ getCategoryName(selectedCategory) }} <i class="fas
129                  fa-times ms-1" @click="selectedCategory = 'all'" style="cursor:
130                      pointer;"></i>
131          </span>
132          <span v-if="priceRange !== 'all'" class="badge bg-warning me-2">
133              Price: {{ getPriceRangeLabel(priceRange) }} <i class="fas fa-times ms-1"
134                  @click="priceRange = 'all'" style="cursor: pointer;"></i>
135          </span>
136          <button class="btn btn-sm btn-outline-danger" @click="clearAllFilters">
137              <i class="fas fa-refresh icon-spacing"></i>Clear All
138          </button>
139      </div>
140      <!-- Results Summary with Pagination Info -->
141      <div class="d-flex justify-content-between align-items-center mb-4" v-if=
142          "filteredProducts.length > 0">
143          <span class="info-bubble me-3">
144              <strong>{{ filteredProducts.length }}</strong> product<span v-if=
145                  "filteredProducts.length !== 1">s</span> found
146              <span v-if="hasActiveFilters" class="text-muted ms-2">
147                  (filtered from {{ products.length }} total)
148              </span>
149              <span class="info-bubble">
150                  <span v-if="totalPages > 1">Page {{ currentPage }} of {{ totalPages }}
151                  </span>
152                  {{ startIndex }}-{{ endIndex }} of {{ filteredProducts.length }}
153              </span>
154      </div>
155      <!-- No Products Message (Conditional Rendering Directive) -->
156      <div v-if="filteredProducts.length === 0 && !loading" class="no-products">
157          <i class="fas fa-search fa-4x text-muted mb-4"></i>
158          <h3>No Products Found</h3>
159          <p class="text-muted mb-4">
160              <span v-if="hasActiveFilters">
161                  Try adjusting your filters or search terms
162              </span>
163              <span v-else>
164                  No products are currently available
165              </span>
166          </p>
167          <button v-if="hasActiveFilters" class="btn btn-brown" @click="clearAllFilters">
168              <i class="fas fa-refresh icon-spacing"></i>Show All Products
169          </button>
170      </div>
171      <!-- Loading State -->
172      <div v-if="loading" class="text-center py-5">
173          <div class="spinner-border text-primary" role="status" style="width: 3rem;
174              height: 3rem;">
175              <span class="visually-hidden">Loading...</span>
176          </div>
177          <p class="mt-3 text-muted">Loading products...</p>
178      </div>

```

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

```

101   <div class="col-md-2">
102     <label class="form-label">View Mode</label>
103     <div class="view-mode-toggle d-flex">
104       <button
105         :class="['view-btn', 'flex-fill', { active: viewMode === 'grid' }]"
106         @click="viewMode = 'grid'"
107       >
108         <i class="fas fa-th"></i>
109       </button>
110       <button
111         :class="['view-btn', 'flex-fill', { active: viewMode === 'list' }]"
112         @click="viewMode = 'list'"
113       >
114         <i class="fas fa-list"></i>
115       </button>
116     </div>
117   </div>
118
177   <!-- Products Display (Repetition Directives with Dynamic Layout) -->
178   <div v-if="!loading && filteredProducts.length > 0">
179     <!-- Grid View -->
180     <div v-if="viewMode === 'grid'" class="row g-4">
181       <div
182         v-for="(product, index) in paginatedProducts"
183         :key="product.id"
184         :class="gridColumnClass"
185       >
186         <div class="card product-card h-100">
187           
192           <div class="card-body">
193             <h5 class="card-title">{{ product.name }}</h5>
194             <p class="card-text text-muted small">{{ truncateText(product.description, 80) }}</p>
195             <div class="d-flex justify-content-between align-items-center">
196               <span class="price-badge">${{ formatPrice(product.price) }}</span>
197               <small class="text-muted">{{ getProductCategory(product) }}</small>
198             </div>
199           </div>
200           <div class="card-footer bg-transparent">
201             <button class="btn btn-custom w-100" @click="openProductModal(product)">
202               <i class="fas fa-eye icon-spacing"></i>View Details
203             </button>
204           </div>
205         </div>
206       </div>
207     </div>

```

```

209      <!-- List View -->
210      <div v-else class="list-view">
211          <div
212              v-for="(product, index) in paginatedProducts"
213              :key="product.id"
214              class="card product-card mb-3"
215          >
216              <div class="row g-0">
217                  <div class="col-md-3">
218                      
223                  </div>
224                  <div class="col-md-9">
225                      <div class="card-body d-flex justify-content-between">
226                          <div>
227                              <h5 class="card-title">{{ product.name }}</h5>
228                              <p class="card-text">{{ product.description }}</p>
229                              <small class="text-muted">Category: {{ getProductCategory(product) }}</small>
230                          </div>
231                          <div class="text-end">
232                              <div class="price-badge mb-2">${{ formatPrice(product.price) }}</div>
233                              <button class="btn btn-custom" @click="openProductModal(product)">
234                                  <i class="fas fa-eye icon-spacing"></i>View
235                              </button>
236                          </div>
237                      </div>
238                  </div>
239              </div>
240          </div>
241      </div>
242  </div>
243

```

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

6. Pagination

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

```

245      <!-- Enhanced Pagination Controls -->
246      <div v-if="totalPages > 1" class="mt-5">
247          <div class="d-flex justify-content-center align-items-center">
248
249              <paginate
250                  v-model="currentPage"
251                  :page-count="totalPages"
252                  :click-handler="changePage"
253                  :prev-text="'Prev'"
254                  :next-text="'Next'"
255                  :container-class="`pagination justify-content-center mb-0`"
256                  :page-class="`page-item`"
257                  :page-link-class="`page-link btn-custom`"
258                  :prev-class="`page-item`"
259                  :prev-link-class="`page-link btn-custom`"
260                  :next-class="`page-item`"
261                  :next-link-class="`page-link btn-custom`"
262                  :active-class="`active`"
263                  :disabled-class="`disabled`"
264                  :page-range="5"
265              />
266
267
268              <button
269                  class="btn btn-outline-primary ms-2"
270                  :disabled="currentPage === totalPages"
271                  @click="changePage(totalPages)"
272              >
273                  <i class="fas fa-angle-double-right"></i>
274              </button>

```

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

```
276
277     <!-- Jump to Page -->
278     <div class="text-center mt-3">
279         <small class="text-muted me-2">Jump to page:</small>
280         <input
281             type="number"
282             class="form-control d-inline-block"
283             style="width: 80px;"
284             :min="1"
285             :max="totalPages"
286             v-model.number="jumpToPage"
287             @keyup.enter="goToPage"
288         >
289         <button class="btn-brown ms-2" @click="goToPage">Go</button>
290     </div>
291 </div>
```

7. Product Modal

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

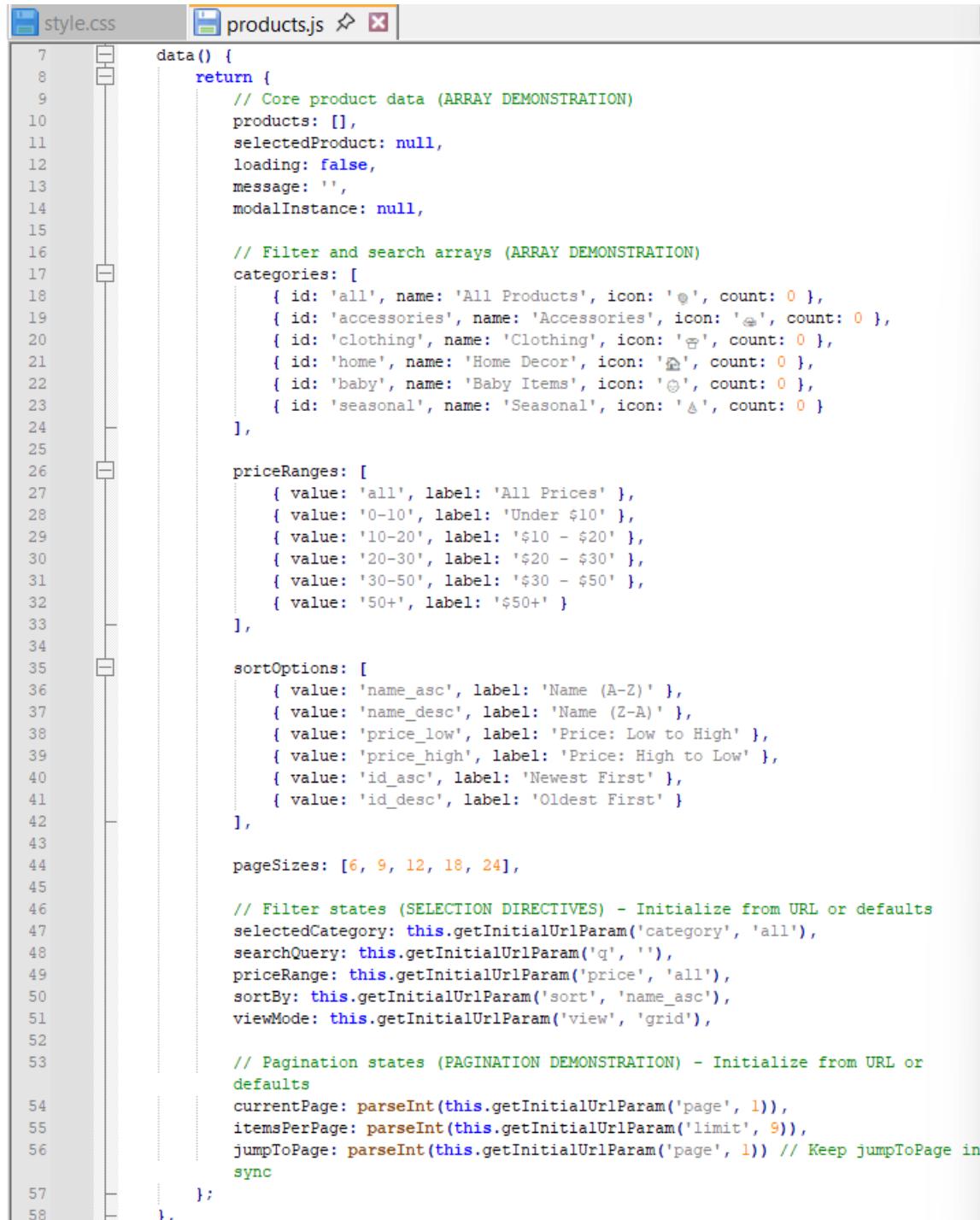
```
292 <!-- Enhanced Product Detail Modal -->
293 <div class="modal fade" id="productModal" tabindex="-1" aria-labelledby=
294 "productModalLabel" aria-hidden="true">
295     <div v-if="message" class="alert alert-success position-fixed top-0 end-0 m-3"
296         style="z-index: 9999;">
297             {{ message }}
298         </div>
299         <div class="modal-dialog modal-lg modal-dialog-centered">
300             <div class="modal-content custom-card" v-if="selectedProduct">
301                 <div class="modal-header">
302                     <h5 class="modal-title">{{ selectedProduct.name }}</h5>
303                     <button type="button" class="btn-close" data-bs-dismiss="modal">
304                         </button>
305                 </div>
306                 <div class="modal-body">
307                     <div class="row">
308                         <div class="col-md-6">
309                             
311                         </div>
312                         <div class="col-md-6">
313                             <h4 class="text-success mb-3">${{ formatPrice(selectedProduct.price) }}</h4>
314                             <p>{{ selectedProduct.description }}</p>
315                             <hr>
316                             <div class="row">
317                                 <div class="col-6">
318                                     <p><strong>Category:</strong><br>{{ getProductCategory(selectedProduct) }}</p>
319                                 </div>
320                                 <div class="col-6">
321                                     <p><strong>Status:</strong><br><span class="badge bg-success">In Stock</span></p>
322                                 </div>
323                             </div>
324                         <div class="modal-footer">
325                             <button type="button" class="btn profile-btn cancel-btn"
326                                 data-bs-dismiss="modal">Close</button>
327                             <button type="button" class="btn profile-btn delete-btn" @click=
328 "addToCart(selectedProduct)">
329                                 <i class="fas fa-cart-plus icon-spacing"></i>Add to Cart
330                             </button>
331                         </div>
332                     </div>
333                 </div>
334             </div>
335         </div>
336     </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 x |  
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  
54     // defaults  
55     currentPage: parseInt(this.getInitialUrlParam('page', 1)),  
56     itemsPerPage: parseInt(this.getInitialUrlParam('limit', 9)),  
57     jumpToPage: parseInt(this.getInitialUrlParam('page', 1)) // Keep jumpToPage in sync  
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(
288              'headband') ||
289                  name.includes('sweater')) { // Removed 'cozy' from clothing
290                  for now
291                  return 'clothing';
292          }
293          // Accessories (This is the default/fallback if no other category matches)
294          // Including "Soft Crochet Blanket" and "Crochet Coffee Cup Cozy" here
295          else {
296              return 'accessories';
297          }
298      },
```

- 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)

```

60      computed: {
61        // FILTER DEMONSTRATION - Complex filtering with multiple conditions
62        filteredProducts() {
63          let filtered = [...this.products];
64
65          // Category filter
66          if (this.selectedCategory !== 'all') {
67            filtered = filtered.filter(product => {
68              return this.getProductCategoryId(product) === this.selectedCategory;
69            });
70          }
71
72          // Search filter (name and description)
73          if (this.searchQuery.trim()) {
74            const query = this.searchQuery.toLowerCase().trim();
75            filtered = filtered.filter(product =>
76              product.name.toLowerCase().includes(query) ||
77              product.description.toLowerCase().includes(query)
78            );
79          }
80
81          // Price range filter
82          if (this.priceRange !== 'all') {
83            filtered = filtered.filter(product => {
84              const price = product.price;
85              switch (this.priceRange) {
86                case '0-10': return price < 10;
87                case '10-20': return price >= 10 && price < 20;
88                case '20-30': return price >= 20 && price < 30;
89                case '30-50': return price >= 30 && price < 50;
90                case '50+': return price >= 50;
91                default: return true;
92              }
93            });
94          }
95
96          // Sorting
97          return filtered.sort((a, b) => {
98            switch (this.sortBy) {
99              case 'name_desc':
100                return b.name.localeCompare(a.name);
101              case 'price_low':
102                return a.price - b.price;
103              case 'price_high':
104                return b.price - a.price;
105              case 'id_desc': // Newest First - higher ID is newer
106                return b.id - a.id;
107              case 'id_asc': // Oldest First - lower ID is older
108                return a.id - b.id;
109              default: // name_asc
110                return a.name.localeCompare(b.name);
111            }
112          });
113        }
114      },
115    };

```

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     ...
118     return this.filteredProducts.length;
119 },
120
121 totalPages() {
122     ...
123     return Math.ceil(this.totalProductsCount / this.itemsPerPage);
124 },
125
126 paginatedProducts() {
127     ...
128     const start = (this.currentPage - 1) * this.itemsPerPage;
129     const end = start + this.itemsPerPage;
130     ...
131     return this.filteredProducts.slice(start, end);
132 },
133
134 startIndex() {
135     ...
136     return (this.currentPage - 1) * this.itemsPerPage + 1;
137 },
138
139 endIndex() {
140     ...
141     return Math.min(this.currentPage * this.itemsPerPage, this.filteredProducts.
142         length);
143 },
144
```

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     ...
163     ...
164     ...
165     ...
166     ...
167     ...
168     ...
169     ...
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     ...
378     ...
379     ...
380     ...
381     ...
382     ...
383     ...
384 }
```

```

385           ...
386           ...
387           ...
388           ...
389           ...
390           ...
391           ...
392           ...
393           ...
394           ...
395           ...
396           ...
397           ...
398           ...
399           ...
400           ...

    console.warn('Bootstrap Modal not found. Make sure Bootstrap JS is
    loaded.');
    // Fallback if Bootstrap isn't loaded, e.g., show a simple alert
    // Corrected template literal syntax
    alert(`Product: ${product.name}\nPrice: ${product.price}\nDescription:
    ${product.description}`);
    return;
}
}

this.modalInstance.show();
},

closeProductModal() {
    if (this.modalInstance) {
        this.modalInstance.hide();
    }
    this.selectedProduct = null;
},

```

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           ...
403           ...
404           ...
405           ...
406           ...
407           ...
408           ...
409           ...
410           ...
411           ...
412           ...
413           ...
414           ...
415           ...
416           ...
417           ...
418           ...
419           ...
420           ...
421           ...
422           ...
423           ...
424           ...
425           ...
426           ...
427           ...
428           ...
429           ...
430           ...
431           ...
432           ...
433           ...
434           ...
435           ...
436           ...
437           ...

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, isSuccess = true) { // Added isSuccess parameter for styling
    this.message = text;
    // You'll need to pass 'isSuccess' / 'isError' state to the template for styling
    // For simplicity, using one 'message' here. If you have different styles,
    // you might need additional ref properties like isMessageSuccess,
    // isMessageError.
    setTimeout(() => {
        this.message = '';
    }, 1000);
},

```

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
241         // search,
242         // but pushState is also an option if you want every filter change in
243         // history.
244         // For search, replaceState is generally better for UX.
245         window.history.replaceState({}, '', newUrl);
246     },
247 }
```

- `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.

```
174      watch: {
175        // Watch for changes in filter and pagination states to update URL
176        selectedCategory: 'updateUrl',
177        searchQuery: 'updateUrl',
178        priceRange: 'updateUrl',
179        sortBy: 'updateUrl',
180        currentPage: 'updateUrl',
181        itemsPerPage: 'updateUrl',
182        viewMode: 'updateUrl',
183
184        // Auto-reset pagination when filters change (still necessary)
185        filteredProducts() {
186          // Only reset if the current page is out of bounds for the new filtered results
187          if (this.currentPage > this.totalPages && this.totalPages > 0) {
188            this.currentPage = 1;
189            this.jumpToPage = 1;
190          } else if (this.totalPages === 0) { // If no products match filters
191            this.currentPage = 1;
192            this.jumpToPage = 1;
193          }
194        },
195
196        // Update category counts when products change
197        products: {
198          handler() {
199            this.updateCategoryCounts();
200          },
201          deep: true
202        }
203      },
```

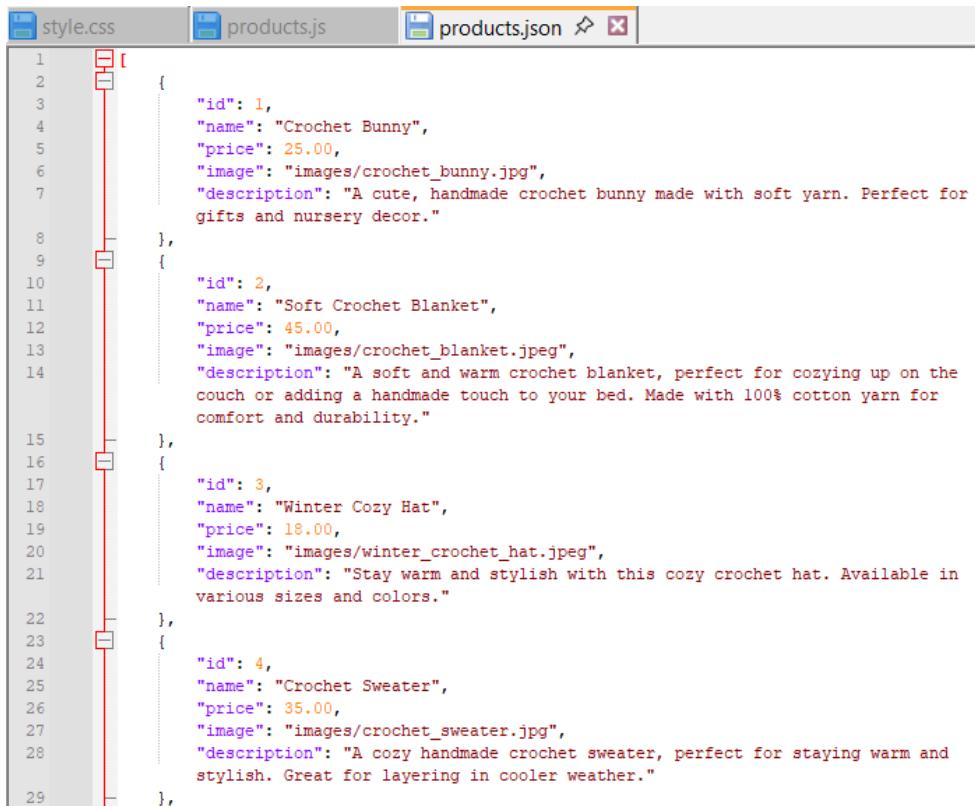
products.json:

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.



```
1  style.css [ 2  products.js [ 3  products.json ✘ ✕ ]]
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
```

```
[{"id": 1, "name": "Crochet Bunny", "price": 25.00, "image": "images/crochet_bunny.jpg", "description": "A cute, handmade crochet bunny made with soft yarn. Perfect for gifts and nursery decor."}, {"id": 2, "name": "Soft Crochet Blanket", "price": 45.00, "image": "images/crochet_blanket.jpeg", "description": "A soft and warm crochet blanket, perfect for cozying up on the couch or adding a handmade touch to your bed. Made with 100% cotton yarn for comfort and durability."}, {"id": 3, "name": "Winter Cozy Hat", "price": 18.00, "image": "images/winter_crochet_hat.jpeg", "description": "Stay warm and stylish with this cozy crochet hat. Available in various sizes and colors."}, {"id": 4, "name": "Crochet Sweater", "price": 35.00, "image": "images/crochet_sweater.jpg", "description": "A cozy handmade crochet sweater, perfect for staying warm and stylish. Great for layering in cooler weather."}]
```

```

37   [
38     {
39       "id": 6,
40       "name": "Crochet Coffee Cup Cozy",
41       "price": 8.00,
42       "image": "images/crochet_coffee_cup_cozy.jpg",
43       "description": "Keep your hands cool with this eco-friendly crochet coffee cup cozy. Fits most standard cup sizes."
44     },
45     {
46       "id": 7,
47       "name": "Crochet Baby Booties",
48       "price": 15.00,
49       "image": "images/crochet_baby_booties.jpg",
50       "description": "Cute and comfy crochet baby booties, perfect for your little one. Soft, stretchy, and gentle on baby's feet."
51     },
52     {
53       "id": 8,
54       "name": "Crochet Keychain",
55       "price": 6.00,
56       "image": "images/crochet_keychain.jpg",
57       "description": "Handmade crochet keychain. Add a touch of personality to your keys, bags, or as a gift."
58     },
59     {
60       "id": 9,
61       "name": "Crochet Plant Pot Cover",
62       "price": 18.00,
63       "image": "images/crochet_plant_pot_cover.jpg",
64       "description": "A stylish crochet cover for your plant pots. Adds charm and protection to your plants while decorating your space."
65     },
66     {
67       "id": 10,
68       "name": "Crochet Handbag",
69       "price": 22.00,
70       "image": "images/crochet_handbag.jpg",
71       "description": "Stylish and eco-friendly crochet handbag, perfect for carrying your essentials with a touch of handmade charm."
72   },
73   {
74     "id": 11,
75     "name": "Crochet Scarf",
76     "price": 20.00,
77     "image": "images/crochet_scarf.jpg",
78     "description": "Warm and cozy crochet scarf, perfect for colder weather. (Wooden house shown in image not included.)"
79   },
80   {
81     "id": 12,
82     "name": "Crochet Tote Bag",
83     "price": 28.00,
84     "image": "images/crochet_tote_bag.jpg",
85     "description": "Handmade crochet tote bag. A stylish and functional accessory for everyday use, shopping, or travel."
86   },
87   {
88     "id": 13,
89     "name": "Crochet Christmas Stocking",
90     "price": 12.00,
91     "image": "images/crochet_christmas_stocking.jpg",
92     "description": "Add a handmade touch to your holiday decorations with this crochet Christmas stocking. Perfect for filling with gifts."
93   },
94   {
95     "id": 14,
96     "name": "Crochet Baby Blanket",
97     "price": 35.00,
98     "image": "images/crochet_baby_blanket.jpeg",
99     "description": "A soft and warm crochet baby blanket, perfect for newborns. Made with hypoallergenic materials."
100  },
101  {
102    "id": 15,
103    "name": "Crochet Phone Case",
104    "price": 14.00,
105    "image": "images/crochet_phone_case.jpg",
106    "description": "Protect your phone with this cute crochet phone case. Handmade with care and available in various sizes."
107  ]

```

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:

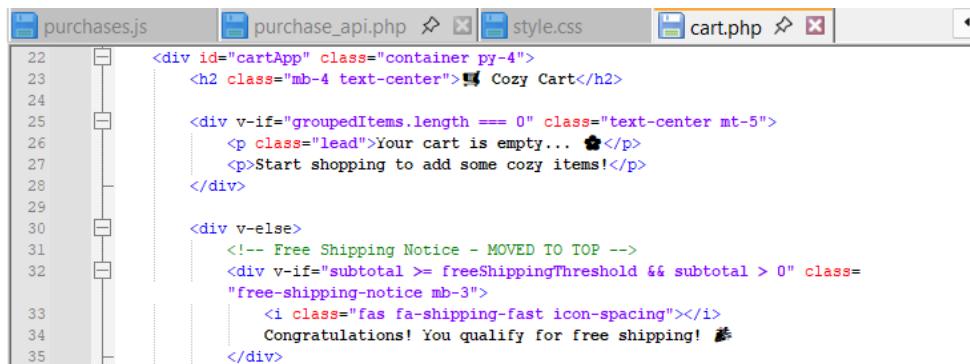
1. Shared Components

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



```
purchases.js purchase_api.php style.css cart.php
22 <div id="cartApp" class="container py-4">
23   <h2 class="mb-4 text-center">Cozy Cart</h2>
24
25   <div v-if="groupedItems.length === 0" class="text-center mt-5">
26     <p class="lead">Your cart is empty... 🛍</p>
27     <p>>Start shopping to add some cozy items!</p>
28   </div>
29
30   <div v-else>
31     <!-- Free Shipping Notice - MOVED TO TOP -->
32     <div v-if="subtotal >= freeShippingThreshold && subtotal > 0" class=
33       "free-shipping-notice mb-3">
34       <i class="fas fa-shipping-fast icon-spacing"></i>
35       Congratulations! You qualify for free shipping! 🎉
36     </div>
37   </div>
38
39   <div v-for="group in groupedItems" class="row mb-4">
40     <div v-for="item in group.items" class="col-4">
41       <div class="card h-100">
42         <img alt="Thumbnail image of the product item" class="card-img-top" data-bbox="191 512 802 674"/>
43         <div class="card-body">
44           <h3>{{ item.name }}</h3>
45           <p>{{ item.description }}</p>
46           <div>
47             <span>{{ item.price }}</span>
48             <span>{{ item.quantity }}</span>
49           </div>
50         </div>
51       </div>
52     </div>
53   </div>
54
55   <div v-if="groupedItems.length > 0" class="text-right">
56     <button class="btn btn-primary" @click="checkout">Checkout</button>
57   </div>
58
59 </div>
```

```

37
38      <!-- Shipping Progress - MOVED TO TOP -->
39      <div v-else-if="subtotal > 0 && subtotal < freeShippingThreshold" class=
40          "shipping-progress mb-3">
41          <i class="fas fa-truck icon-spacing"></i>
42          Add ${((freeShippingThreshold - subtotal).toFixed(2))} more to qualify
43          for free shipping!
44          <div class="progress mt-2" style="height: 8px;">
45              <div
46                  class="progress-bar bg-warning"
47                  :style="{ width: (subtotal / freeShippingThreshold * 100) + '%' }"
48              ></div>
49      </div>
50
51      <!-- Cart Controls -->
52      <div class="cart-container">
53          <div class="row align-items-center">
54              <div class="col-1 d-flex justify-content-center">
55                  <input
56                      type="checkbox"
57                      id="selectAll"
58                      @change="toggleSelectAll()"
59                      :checked="groupedItems.length > 0 && groupedItems.every(item =>
60                          item.selected)">
61
62                  <div class="col-7">
63                      <label for="selectAll">
64                          Select All Items {{ selectedItemsCount }} selected
65                      </label>
66
67                  <div class="col-4 text-end">
68                      <button class="remove-all-btn" @click="clearCart()">
69                          <i class="fas fa-trash icon-spacing"></i>Clear Cart
70                  </div>
71          </div>
72
73      <!-- Cart Items Container -->
74      <div class="cart-container">
75          <div class="cart-item row align-items-center" v-for="item in groupedItems"
76              :key="item.id">
77              <div class="col-1 d-flex justify-content-center">
78                  <input type="checkbox" v-model="item.selected" />
79
80              <div class="col-2">
81                  
82              </div>
83
84              <div class="col-5">
85                  <h5>{{ item.name }}</h5>
86                  <p class="mb-1">Price: ${{ item.price.toFixed(2) }}</p>
87                  <p class="mb-1">
88                      Quantity:
89                  <div class="btn-group btn-group-sm" role="group" aria-label=
90                      "Quantity controls">
91                      <button
92                          class="btn btn-outline-secondary"
93                          @click="removeOne(item)"
94                          :disabled="item.quantity === 1"
95                      >-</button>
96                      <span class="btn btn-outline-primary disabled
97                          quantity-number">
98                          {{ item.quantity }}
99                      </span>
100                     <button class="btn btn-outline-secondary" @click=
101                         "addOne(item)">+</button>
102
103                  </div>
104
105              <div class="col-4 text-end">
106                  <button class="remove-all-btn" @click="removeAll(item)">
107                      <i class="fas fa-trash"></i>
108                  </button>
109
110          </div>
111      </div>

```

```

112      <!-- Order Summary -->
113      <div class="mt-4 total-price-section">
114          <h5 class="mb-3">Order Summary</h5>
115
116          <div class="d-flex justify-content-between mb-2">
117              <span>Subtotal {{ selectedItemsCount }} items:</span>
118              <span>${{ subtotal.toFixed(2) }}</span>
119          </div>
120
121          <div class="d-flex justify-content-between mb-2">
122              <span>Shipping:</span>
123              <span v-if="postage === 0 && subtotal > 0" class="text-success">
124                  <i class="fas fa-check icon-spacing"></i>FREE
125              </span>
126              <span v-else-if="postage === 0">$0.00</span>
127              <span v-else>${{ postage.toFixed(2) }}</span>
128          </div>
129
130
131          <hr>
132
133          <div class="d-flex justify-content-between">
134              <h5>Total:</h5>
135              <h5 class="text-primary">${{ totalPrice.toFixed(2) }}</h5>
136          </div>
137
138          <div class="mt-3 text-center">
139              <button
140                  class="btn btn-brown btn-lg"
141                  :disabled="selectedItemsCount === 0"
142                  @click="proceedToCheckout"
143              >
144                  <i class="fas fa-credit-card icon-spacing"></i>Proceed to Checkout
145              </button>
146          </div>
147      </div>
148
149      <!-- Confirmation Dialog (Custom Modal) - MOVED INSIDE #cartApp -->
150      <div class="modal fade" id="confirmationModal" tabindex="-1" aria-labelledby="confirmationModalLabel" aria-hidden="true">
151          <div class="modal-dialog modal-dialog-centered">
152              <div class="modal-content custom-card">
153                  <div class="modal-header">
154                      <h5 class="modal-title section-title" id="confirmationModalLabel">
155                          {{ confirmTitle }}</h5>
156                      <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close" @click="closeConfirm"></button>
157                  </div>
158                  <div class="modal-body text-muted">
159                      {{ confirmMessage }}
160                  </div>
161                  <div class="modal-footer">
162                      <button type="button" class="btn profile-btn cancel-btn" data-bs-dismiss="modal" @click="closeConfirm">Cancel</button>
163                      <button type="button" class="btn profile-btn delete-btn" @click="executeConfirmAction">Confirm</button>
164                  </div>
165              </div>
166          </div>
167      </div>
168

```

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.

```

149
150      <!-- Confirmation Dialog (Custom Modal) - MOVED INSIDE #cartApp -->
151      <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.

```

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>

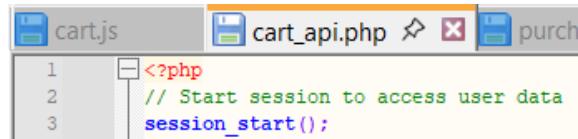
```

cart_api.php:

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.

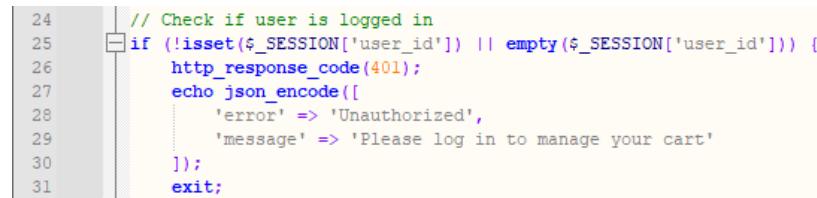
1. Session Management and Security

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



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

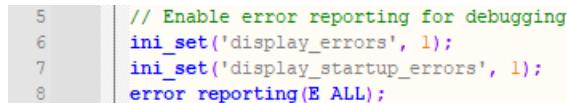
- Verifies that a user is logged in before allowing access to cart data.
- If not authenticated, the script returns an HTTP 401 Unauthorized response.



```
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 manage your cart'
30     ]);
31     exit;
```

2. Debugging and Development Helpers

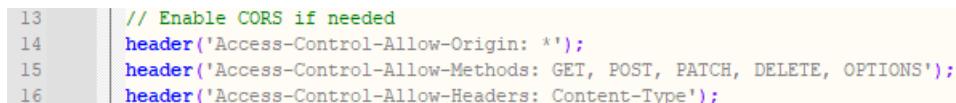
- Enables full error reporting during development for easier debugging.



```
5 // Enable error reporting for debugging
6 ini_set('display_errors', 1);
7 ini_set('display_startup_errors', 1);
8 error_reporting(E_ALL);
```

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).



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

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



```
18 // Handle preflight requests
19 if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
20     http_response_code(200);
21     exit;
22 }
```

4. Database Connection

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

```
36 // Connect to MySQL
37 $conn = mysqli_connect('localhost', 'root', '', 'project');
38 if (!$conn) {
39     http_response_code(500);
40     echo json_encode([
41         'error' => 'Failed to connect to database',
42         'details' => mysqli_connect_error()
43     ]);
44     exit;
45 }
```

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.

```
53 if ($method === 'POST') {
54     // Add item to cart
55     $input = json_decode(file_get_contents('php://input'), true);
56
57     if (!isset($input['id'], $input['name'], $input['price'], $input['image'])) {
58         http_response_code(400);
59         echo json_encode(['error' => 'Missing required fields: id, name, price, image']);
60         exit;
61     }
62
63     // Validate input
64     if (!is_numeric($input['price']) || $input['price'] < 0) {
65         http_response_code(400);
66         echo json_encode(['error' => 'Invalid price']);
67         exit;
68     }
69
70     $stmt = mysqli_prepare($conn, "INSERT INTO cart (user_id, product_id, name, price,
71     image) VALUES (?, ?, ?, ?, ?)");
72     if (!$stmt) {
73         throw new Exception('Failed to prepare statement: ' . mysqli_error($conn));
74     }
75
76     mysqli_stmt_bind_param($stmt, "iisds", $user_id, $input['id'], $input['name'],
77     $input['price'], $input['image']);
78
79     if (mysqli_stmt_execute($stmt)) {
80         echo json_encode([
81             'success' => true,
82             'message' => 'Item added to cart',
83             'item_id' => mysqli_insert_id($conn)
84         ]);
85     } else {
86         throw new Exception('Failed to execute statement: ' . mysqli_stmt_error($stmt));
87     }
88
89     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 };
93 if (!$stmt) {
94     throw new Exception('Failed to prepare statement: ' . mysqli_error($conn));
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);
202     }
```

```

202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
} 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))

```

118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
} 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);
    }
}

```

```

142 } elseif (isset($input['id'])) {
143     // Remove all instances of a specific product for current user
144     $stmt = mysqli_prepare($conn, "DELETE FROM cart WHERE user_id = ? AND
145     product_id = ?");
146     if (!$stmt) {
147         throw new Exception('Failed to prepare statement: ' . mysqli_error($conn));
148     }
149
150     mysqli_stmt_bind_param($stmt, "ii", $user_id, $input['id']);
151
152     if (mysqli_stmt_execute($stmt)) {
153         $affected_rows = mysqli_stmt_affected_rows($stmt);
154         echo json_encode([
155             'success' => true,
156             'message' => 'Items removed from cart',
157             'removed_count' => $affected_rows
158         ]);
159     } else {
160         throw new Exception('Failed to execute statement: ' . mysqli_stmt_error(
161             $stmt));
162     }
163
164     mysqli_stmt_close($stmt);
165 } else {
166     http_response_code(400);
167     echo json_encode(['error' => 'Missing product ID or clear_all flag']);
168 }

```

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.

```

1  const { createApp } = Vue;
2  createApp({
322 }).mount('#cartApp');

```

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.

```

3  data() {
4      return {
5          rawItems: [],
6          groupedItems: [],
7          postageRate: 5.00, // Fixed postage rate
8          freeShippingThreshold: 50.00, // Free shipping over $50
9          confirmDialog: false, // Controls visibility of the custom confirmation modal
10         confirmTitle: '', // Title for the confirmation modal
11         confirmMessage: '', // Message for the confirmation modal
12         confirmAction: null, // Callback function to execute on confirmation
13     };
14 }

```

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.

```
15      computed: {
16        subtotal() {
17          return this.groupedItems
18            .filter(item => item.selected)
19            .reduce((sum, item) => sum + (item.price * item.quantity), 0);
20        },
21        postage() {
22          // Free shipping if subtotal is over threshold
23          if (this.subtotal >= this.freeShippingThreshold) {
24            return 0;
25          }
26          // No postage if cart is empty
27          if (this.subtotal === 0) {
28            return 0;
29          }
30          return this.postageRate;
31        },
32        totalPrice() {
33          return this.subtotal + this.postage;
34        },
35        selectedItemsCount() {
36          return this.groupedItems.filter(item => item.selected).length;
37        }
38      },
```

4. Key Methods

a. groupItems()

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

```
39      methods: {
40        groupItems() {
41          // Store current selection state before grouping
42          const currentSelections = {};
43          this.groupedItems.forEach(item => {
44            currentSelections[item.id] = item.selected;
45          });
46
47          const grouped = {};
48          this.rawItems.forEach(item => {
49            if (!grouped[item.product_id]) {
50              grouped[item.product_id] = {
51                id: item.product_id,
52                name: item.name,
53                price: parseFloat(item.price),
54                image: item.image,
55                quantity: 1,
56                // Preserve previous selection state, default to true for new items
57                selected: currentSelections[item.product_id] !== undefined ?
58                  currentSelections[item.product_id] : true
59              };
60            } else {
61              grouped[item.product_id].quantity += 1;
62            }
63          });
64          this.groupedItems = Object.values(grouped);
65        }
66      },
```

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

e. toggleSelectAll()

- Selects all the items in the cart

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

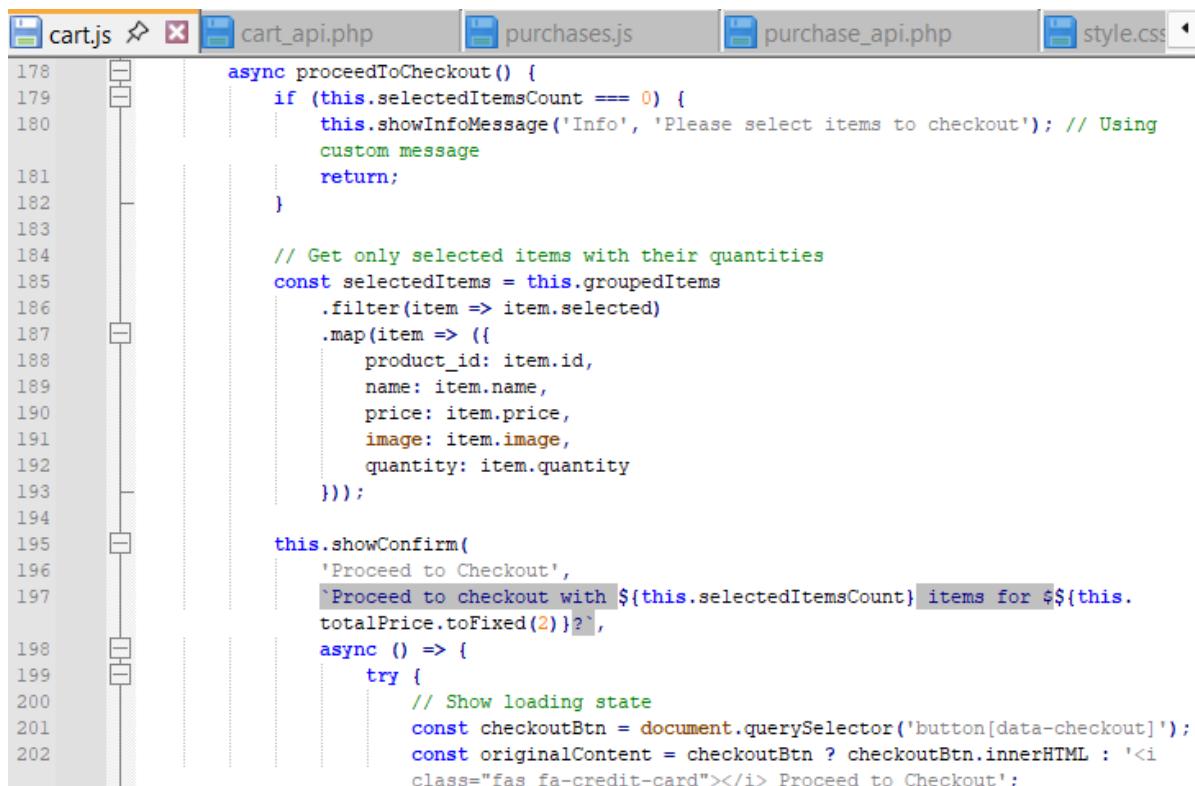
f. clearCart()

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

```

203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
    if (checkoutBtn) {
      checkoutBtn.disabled = true;
      checkoutBtn.innerHTML = '<i class="fas fa-spinner fa-spin"></i> Processing...';
    }

    const response = await fetch('purchase_api.php', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        action: 'create_from_selected_cart',
        selected_items: selectedItems,
        subtotal: this.subtotal,
        shipping: this.postage,
        total: this.totalPrice
      })
    });

    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }

    const result = await response.json();

    if (result.success) {
      this.showInfoMessage('Success', `Order created successfully!
Order #${result.order_number}`); // Using custom message

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

      // Reload cart to reflect changes
      this.reloadCart();

      // Optionally redirect to purchase history with confirmation
      // modal
      this.showConfirm(
        'View Purchases',
        'Would you like to view your purchase history?',
        () => {
          window.location.href = 'purchases.php';
        }
      );
    } else {
      throw new Error(result.message || 'Failed to create order');
    }
  } catch (error) {
    console.error('Checkout failed:', error);
    this.showInfoMessage('Error', 'Failed to process checkout. Please try again.');// Using custom message
  } finally {
    // Reset button state
    const checkoutBtn = document.querySelector('button[data-checkout]');
    if (checkoutBtn) {
      const cartData = { selectedItemsCount: this.selectedItemsCount };
      // Simplified for scope
      checkoutBtn.disabled = cartData.selectedItemsCount === 0;
      checkoutBtn.innerHTML = originalContent;
    }
  }
},
}

```

```

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(
288     'confirmationModal'));
289   modalElement.show();
290 }
291
292 closeConfirm() {
293   this.confirmDialog = false; // Reset Vue state
294   this.confirmTitle = '';
295   this.confirmMessage = '';
296   this.confirmAction = null;
297   // Use Bootstrap's JavaScript API to hide the modal
298   const modalElement = bootstrap.Modal.getInstance(document.getElementById(
299     'confirmationModal'));
300   if (modalElement) {
301     modalElement.hide();
302   }
303 }
304
305 executeConfirmAction() {
306   if (this.confirmAction) {
307     this.confirmAction();
308   }
309   this.closeConfirm(); // Close the modal after action

```

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
313   // toast/snackbar
314   // or a more sophisticated modal for general info messages.
315   console.log(`${title}: ${message}`);
316   // For now, if you want a visible (but non-native) alert, you could temporarily
317   // use:
318   // alert(`${title}: ${message}`);

```

8. Mounted Lifecycle Hook

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

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

How It Connects to Other Files:

cart.php:

- The HTML page where the cart is rendered.
- Includes <div id="cartApp"> 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

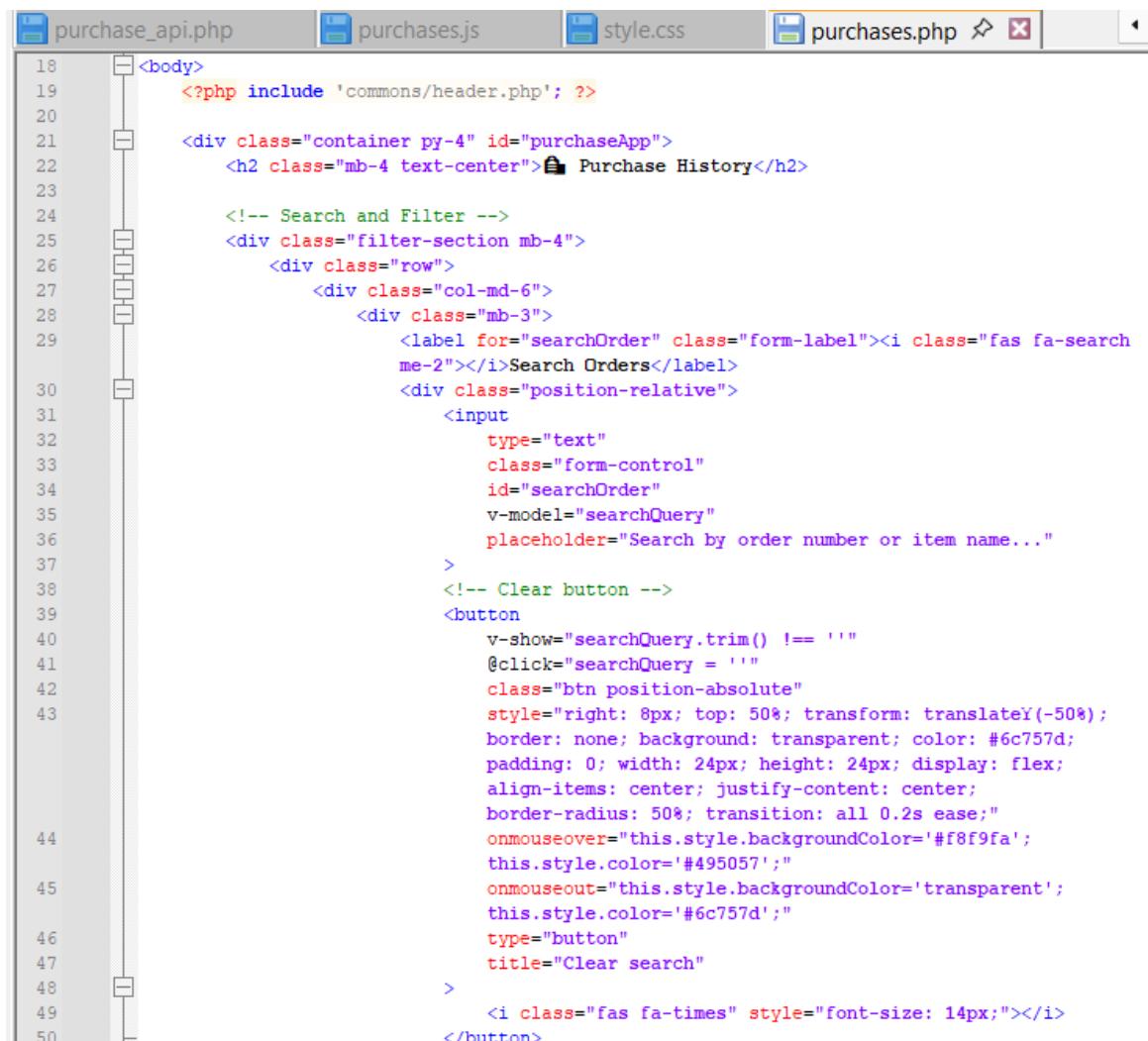
This file is responsible for rendering the purchase history page interface and mounting the Vue app.

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 <div id="purchaseApp"> container where the purchases.js Vue app is mounted.
- Renders interface of all user purchases dynamically within this section.



```
18 <body>
19     <?php include 'commons/header.php'; ?>
20
21     <div class="container py-4" id="purchaseApp">
22         <h2 class="mb-4 text-center">Purchase History</h2>
23
24         <!-- Search and Filter -->
25         <div class="filter-section mb-4">
26             <div class="row">
27                 <div class="col-md-6">
28                     <div class="mb-3">
29                         <label for="searchOrder" class="form-label"><i class="fas fa-search me-2"></i>Search Orders</label>
30                         <div class="position-relative">
31                             <input
32                                 type="text"
33                                 class="form-control"
34                                 id="searchOrder"
35                                 v-model="searchQuery"
36                                 placeholder="Search by order number or item name...">
37                         <!-- Clear button -->
38                         <button
39                             v-show="searchQuery.trim() !== ''"
40                             @click="searchQuery = ''"
41                             class="btn position-absolute"
42                             style="right: 8px; top: 50%; transform: translateY(-50%);"
43                             border: none; background: transparent; color: #6c757d;
44                             padding: 0; width: 24px; height: 24px; display: flex;
45                             align-items: center; justify-content: center;
46                             border-radius: 50%; transition: all 0.2s ease;">
47                             onmouseover="this.style.backgroundColor='#f8f9fa';
48                             this.style.color='#495057';"
49                             onmouseout="this.style.backgroundColor='transparent';
50                             this.style.color='#6c757d';"
51                             type="button"
52                             title="Clear search">
53                             <i class="fas fa-times" style="font-size: 14px;"></i>
54                         </button>
55                     </div>
56                 </div>
57             </div>
58         </div>
59     </div>
60 
```

```

51             </div>
52         </div>
53     </div>
54     <div class="col-md-3">
55         <div class="mb-3">
56             <label for="statusFilter" class="form-label"><i class="fas fa-filter me-2"></i>Filter by Status</label>
57             <select class="form-select" id="statusFilter" v-model="statusFilter">
58                 <option value="">All Orders</option>
59                 <option value="pending">Pending</option>
60                 <option value="cancelled">Cancelled</option>
61             </select>
62         </div>
63     </div>
64     <div class="col-md-3">
65         <div class="mb-3">
66             <label for="sortBy" class="form-label"><i class="fas fa-sort me-2"></i>Sort by</label>
67             <select class="form-select" id="sortBy" v-model="sortBy">
68                 <option value="newest">Newest First</option>
69                 <option value="oldest">Oldest First</option>
70                 <option value="amount_high">Highest Amount</option>
71                 <option value="amount_low">Lowest Amount</option>
72             </select>
73         </div>
74     </div>
75 </div>
76
77
78 <!-- No purchases message -->
79 <div v-if="filteredPurchases.length === 0 && !loading" class="text-center mt-5">
80     <p class="lead">No purchases found... 🚫</p>
81     <p>Your purchase history will appear here once you make some orders!</p>
82 </div>
83
84 <!-- Loading state -->
85 <div v-if="loading" class="text-center mt-5">
86     <div class="spinner-border text-primary" role="status">
87         <span class="visually-hidden">Loading...</span>
88     </div>
89 </div>
90
91 <!-- Purchase Items -->
92 <div v-for="purchase in filteredPurchases" :key="purchase.id" class="purchase-container">
93     <div class="purchase-header">
94         <div class="row align-items-center">
95             <div class="col-md-6">
96                 <h5 class="mb-1">Order #{{ purchase.order_number }}</h5>
97                 <small class="text-muted">{{ formatDate(purchase.order_date) }}</small>
98             </div>
99             <div class="col-md-3">
100                 <span :class="'status-badge status-' + purchase.order_status">
101                     {{ purchase.order_status }}
102                 </span>
103             </div>
104             <div class="col-md-3 text-end">
105                 <div class="btn-group btn-group-sm">
106                     <button
107                         class="btn btn-outline-cozy"
108                         @click="toggleEdit(purchase.id)"
109                         v-if="purchase.order_status === 'pending'"
110                     >
111                         <i class="fas fa-edit"></i> Edit
112                     </button>

```

```

113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
    <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 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>


<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">
                
            </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>
    </div>
</div>

```

```

179          <div class="col-2 text-end">
180              <span class="fw-bold">${{ item.price *
181                  item.quantity).toFixed(2) }}</span>
182          </div>
183      </div>
184  </div>
185
186  <!-- Order Summary -->
187  <div class="row mt-3 pt-3" style="border-top: 1px solid #e9ecef;">
188      <div class="col-md-8"></div>
189      <div class="col-md-4">
190          <div class="d-flex justify-content-between mb-1">
191              <span>Subtotal:</span>
192              <span>${{ (purchase.total_amount -
193                  purchase.shipping_cost).toFixed(2) }}</span>
194          </div>
195          <div class="d-flex justify-content-between mb-1">
196              <span>Shipping:</span>
197              <span v-if="purchase.shipping_cost === 0" class="text-success">FREE
198                  </span>
199              <span v-else>${{ purchase.shipping_cost.toFixed(2) }}</span>
200          </div>
201          <div class="d-flex justify-content-between fw-bold">
202              <span>Total:</span>
203              <span>${{ purchase.total_amount.toFixed(2) }}</span>
204          </div>
205      </div>
206
207  <!-- Confirmation Dialog -->
208  <div class="modal fade" id="confirmationModal" tabindex="-1" aria-labelledby=
209      "confirmationModalLabel" aria-hidden="true">
210      <div class="modal-dialog modal-dialog-centered">
211          <div class="modal-content custom-card">
212              <div class="modal-header">
213                  <h5 class="modal-title section-title" id="confirmationModalLabel">
214                      {{ confirmTitle }}</h5>
215                  <button type="button" class="btn-close" data-bs-dismiss="modal"
216                      aria-label="Close" @click="closeConfirm"></button>
217              </div>
218              <div class="modal-body text-muted">
219                  {{ confirmMessage }}<br/>
220              </div>
221              <div class="modal-footer">
222                  <button type="button" class="btn profile-btn cancel-btn"
223                      data-bs-dismiss="modal" @click="closeConfirm">Cancel</button>
224                  <button type="button" class="btn profile-btn delete-btn" @click=
225                      "executeConfirmAction">Confirm</button>
226              </div>
227          </div>
228      </div>
229  </div>
230

```

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**.

```

24  <!-- Search and Filter -->
25  <div class="filter-section mb-4">
26      <div class="row">
27          <div class="col-md-6">
28              <div class="mb-3">
29                  <label for="searchOrder" class="form-label"><i class="fas fa-search
30                      me-2"></i>Search Orders</label>
31                  <div class="position-relative">

```

```
31 <input  
32     type="text"  
33     class="form-control"  
34     id="searchOrder"  
35     v-model="searchQuery"  
36     placeholder="Search by order number or item name..."  
37 >  
38 <!-- Clear button -->  
39 <button  
40     v-show="searchQuery.trim() !== ''"  
41     @click="searchQuery = ''"  
42     class="btn position-absolute"  
43     style="right: 8px; top: 50%; transform: translateY(-50%);  
44     border: none; background: transparent; color: #6c757d;  
45     padding: 0; width: 24px; height: 24px; display: flex;  
46     align-items: center; justify-content: center;  
47     border-radius: 50%; transition: all 0.2s ease;"  
48     onmouseover="this.style.backgroundColor='#f8f9fa';  
49     this.style.color='#495057';"  
50     onmouseout="this.style.backgroundColor='transparent';  
51     this.style.color='#6c757d';"  
52     type="button"  
53     title="Clear search"  
54 >  
55     <i class="fas fa-times" style="font-size: 14px;"></i>  
56 </button>  
57 </div>  
58 </div>  
59 <div class="col-md-3">  
60     <div class="mb-3">  
61         <label for="statusFilter" class="form-label"><i class="fas fa-filter me-2"></i>Filter by Status</label>  
62         <select class="form-select" id="statusFilter" v-model="statusFilter">  
63             <option value="">All Orders</option>  
64             <option value="pending">Pending</option>  
65             <option value="cancelled">Cancelled</option>  
66         </select>  
67     </div>  
68 <div class="col-md-3">  
69     <div class="mb-3">  
70         <label for="sortBy" class="form-label"><i class="fas fa-sort me-2"></i>Sort by</label>  
71         <select class="form-select" id="sortBy" v-model="sortBy">  
72             <option value="newest">Newest First</option>  
73             <option value="oldest">Oldest First</option>  
74             <option value="amount_high">Highest Amount</option>  
75             <option value="amount_low">Lowest Amount</option>  
76         </select>  
77     </div>  
78 </div>  
79 </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**.

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

```

99      <div class="col-md-3">
100     <span :class="'status-badge status-' + purchase.order_status>
101       {{ purchase.order_status }}
102     </span>
103   </div>
104   <div class="col-md-3 text-end">
105     <div class="btn-group btn-group-sm">
106       <button
107         class="btn btn-outline-cozy"
108         @click="toggleEdit(purchase.id)"
109         v-if="purchase.order_status === 'pending'"
110       >
111         <i class="fas fa-edit"></i> Edit
112       </button>
113       <button
114         class="btn btn-danger-soft"
115         @click="cancelOrder(purchase.id)"
116         v-if="purchase.order_status === 'pending' ||"
117           purchase.order_status === 'processing'"
118       >
119         <i class="fas fa-times"></i> Cancel
120       </button>
121     </div>
122   </div>
123 </div>
162 <!-- Order Items Display -->
163 <div v-else>
164   <div v-if="purchase.items && purchase.items.length === 0" class=
165     "text-center text-muted py-4">
166     <p>This order currently has no items.</p>
167   </div>
168   <div v-for="item in purchase.items" :key="item.id" class="purchase-item">
169     <div class="row align-items-center">
170       <div class="col-2">
171         
173       </div>
174       <div class="col-6">
175         <h6 class="mb-1">{{ item.name }}</h6>
176         <small class="text-muted">Quantity: {{ item.quantity }}</small>
177       </div>
178       <div class="col-2">
179         <span class="fw-bold">${{ item.price.toFixed(2) }}</span>
180       </div>
181       <div class="col-2 text-end">
182         <span class="fw-bold">${{ (item.price *
183           item.quantity).toFixed(2) }}</span>
184       </div>
185     </div>
186   </div>
187 <!-- Order Summary -->
188 <div class="row mt-3 pt-3" style="border-top: 1px solid #e9ecf;">
189   <div class="col-md-8"></div>
190   <div class="col-md-4">
191     <div class="d-flex justify-content-between mb-1">
192       <span>Subtotal:</span>
193       <span>${{ (purchase.total_amount -
194         purchase.shipping_cost).toFixed(2) }}</span>
195     </div>
196     <div class="d-flex justify-content-between mb-1">
197       <span>Shipping:</span>
198       <span v-if="purchase.shipping_cost === 0" class="text-success">FREE
199       </span>
200       <span v-else>${{ purchase.shipping_cost.toFixed(2) }}</span>
201     </div>
202     <div class="d-flex justify-content-between fw-bold">
203       <span>Total:</span>
204       <span>${{ purchase.total_amount.toFixed(2) }}</span>
205     </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.

```

104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
      <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=
209          "confirmationModalLabel" aria-hidden="true">
210          <div class="modal-dialog modal-dialog-centered">
211              <div class="modal-content custom-card">
212                  <div class="modal-header">
213                      <h5 class="modal-title section-title" id="confirmationModalLabel">
214                          {{ confirmTitle }}</h5>
215                      <button type="button" class="btn-close" data-bs-dismiss="modal"
216                          aria-label="Close" @click="closeConfirm"></button>
217                  </div>
218                  <div class="modal-body text-muted">
219                      {{ confirmMessage }}</div>
220                  <div class="modal-footer">
221                      <button type="button" class="btn profile-btn cancel-btn"
222                          data-bs-dismiss="modal" @click="closeConfirm">Cancel</button>
223                      <button type="button" class="btn profile-btn delete-btn" @click=
224                          "executeConfirmAction">Confirm</button>
225                  </div>
226              </div>
227          </div>
228      </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']`.

```
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.

```
5 // Enable error reporting for debugging
6 ini_set('display_errors', 1);
7 ini_set('display_startup_errors', 1);
8 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.

```
13 // Enable CORS if needed
14 header('Access-Control-Allow-Origin: *');
15 header('Access-Control-Allow-Methods: GET, POST, PATCH, DELETE, OPTIONS');
16 header('Access-Control-Allow-Headers: Content-Type');
17
18 // Handle preflight requests
19 if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
20     http_response_code(200);
21     exit;
22 }
```

4. Database Connection

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

```
36 // Connect to MySQL
37 $conn = mysqli_connect('localhost', 'root', '', 'project');
38 if (!$conn) {
39     http_response_code(500);
40     echo json_encode([
41         'error' => 'Failed to connect to database',
42         'details' => mysqli_connect_error()
43     ]);
44     exit;
45 }
```

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:

```

53     if ($method === 'GET') {
54         // Get all purchases for the current user with their items
55         $purchasesQuery = "
56             SELECT p.*,
57                 COUNT(pi.id) as item_count,
58                 SUM(pi.quantity) as total_items
59             FROM purchases p
60             LEFT JOIN purchase_items pi ON p.id = pi.purchase_id
61             WHERE p.user_id = ?
62             GROUP BY p.id
63             ORDER BY p.order_date DESC
64         ";
65
66         $purchasesStmt = mysqli_prepare($conn, $purchasesQuery);
67         if (!$purchasesStmt) {
68             throw new Exception('Failed to prepare statement: ' . mysqli_error($conn));
69         }
70
71         mysqli_stmt_bind_param($purchasesStmt, "i", $user_id);
72         mysqli_stmt_execute($purchasesStmt);
73         $purchasesResult = mysqli_stmt_get_result($purchasesStmt);
74
75         if (!$purchasesResult) {
76             throw new Exception('Failed to fetch purchases: ' . mysqli_error($conn));
77         }
78
79         $purchases = [];
80         while ($row = mysqli_fetch_assoc($purchasesResult)) {
81             // Get items for this purchase
82             $itemsQuery = "SELECT * FROM purchase_items WHERE purchase_id = ? ORDER BY id ASC";
83             $itemsStmt = mysqli_prepare($conn, $itemsQuery);
84             mysqli_stmt_bind_param($itemsStmt, "i", $row['id']);
85             mysqli_stmt_execute($itemsStmt);
86             $itemsResult = mysqli_stmt_get_result($itemsStmt);
87
88             $items = [];
89             while ($itemRow = mysqli_fetch_assoc($itemsResult)) {
90                 $items[] = [
91                     'id' => (int)$itemRow['id'],
92                     'product_id' => (int)$itemRow['product_id'],
93                     'name' => $itemRow['name'],
94                     'price' => (float)$itemRow['price'],
95                     'quantity' => (int)$itemRow['quantity'],
96                     'image' => $itemRow['image']
97                 ];
98             }
99             mysqli_stmt_close($itemsStmt);
100
101            $purchases[] = [
102                'id' => (int)$row['id'],
103                'order_number' => $row['order_number'],
104                'total_amount' => (float)$row['total_amount'],
105                'shipping_cost' => (float)$row['shipping_cost'],
106                'order_status' => $row['order_status'],
107                'order_date' => $row['order_date'],
108                'updated_at' => $row['updated_at'],
109                'item_count' => (int)$row['item_count'],
110                'total_items' => (int)$row['total_items'],
111                'items' => $items
112            ];
113        }
114
115        mysqli_stmt_close($purchasesStmt);
116        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',
128                                         STR_PAD_LEFT);
129
130         // Check if order number exists and regenerate if needed
131         $checkStmt = mysqli_prepare($conn, "SELECT id FROM purchases WHERE order_number
132                                     = ?");
133         mysqli_stmt_bind_param($checkStmt, "s", $orderNumber);
134         mysqli_stmt_execute($checkStmt);
135         $checkResult = mysqli_stmt_get_result($checkStmt);
136
137         while (mysqli_num_rows($checkResult) > 0) {
138             $orderNumber = 'ORD-' . date('Y') . '-' . str_pad(rand(1, 9999), 4, '0',
139                                         STR_PAD_LEFT);
140             mysqli_stmt_bind_param($checkStmt, "s", $orderNumber);
141             mysqli_stmt_execute($checkStmt);
142             $checkResult = mysqli_stmt_get_result($checkStmt);
143         }
144         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
144     // Handle different request formats
145     if (isset($input['action']) && $input['action'] === 'create_from_selected_cart') {
146
147         // New format: selected items passed directly
148         if (!isset($input['selected_items']) || empty($input['selected_items'])) {
149             throw new Exception('No items selected for checkout');
150         }
151
152         $selectedItems = $input['selected_items'];
153         $subtotal = isset($input['subtotal']) ? (float)$input['subtotal'] : 0;
154         $shippingCost = isset($input['shipping']) ? (float)$input['shipping'] : 0;
155         $totalAmount = isset($input['total']) ? (float)$input['total'] : ($subtotal
156                                         + $shippingCost);
157
158         // Validate the calculations
159         $calculatedSubtotal = 0;
160         foreach ($selectedItems as $item) {
161             $calculatedSubtotal += (float)$item['price'] * (int)$item['quantity'];
162         }
163
164         // Allow small floating point differences
165         if (abs($calculatedSubtotal - $subtotal) > 0.01) {
166             throw new Exception('Subtotal mismatch detected');
167         }
168
169         $calculatedShipping = $calculatedSubtotal >= 50.00 ? 0.00 : 5.00;
```

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(
176             $conn));
177     }
178
179     mysqli_stmt_bind_param($cartStmt, "i", $user_id);
180     mysqli_stmt_execute($cartStmt);
181     $cartResult = mysqli_stmt_get_result($cartStmt);
182
183     $cartItems = [];
184     while ($row = mysqli_fetch_assoc($cartResult)) {
185         $cartItems[] = $row;
186     }
187     mysqli_stmt_close($cartStmt);
188
189     if (empty($cartItems)) {
190         throw new Exception('Cart is empty');
191     }
192
193     // Group cart items by product_id and calculate totals
194     $groupedItems = [];
195     $subtotal = 0;
196
197     foreach ($cartItems as $item) {
198         $productId = $item['product_id'];
199         if (!isset($groupedItems[$productId])) {
200             $groupedItems[$productId] = [
201                 'product_id' => $productId,
202                 'name' => $item['name'],
203                 'price' => (float)$item['price'],
204                 'image' => $item['image'],
205                 'quantity' => 0
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,
219             order_status)
220             VALUES (?, ?, ?, ?, 'pending')
221     ");
222
223     mysqli_stmt_bind_param($purchaseStmt, "isdd", $user_id, $orderNumber,
224     $totalAmount, $shippingCost);
225
226     if (!mysqli_stmt_execute($purchaseStmt)) {
227         throw new Exception('Failed to create purchase: ' . mysqli_stmt_error(
228             $purchaseStmt));
229     }
230
231     $purchaseId = mysqli_insert_id($conn);
232     mysqli_stmt_close($purchaseStmt);
```

```

230         // Insert purchase items
231         $itemStmt = mysqli_prepare($conn, "
232             INSERT INTO purchase_items (purchase_id, product_id, name, price, quantity,
233             image)
234             VALUES (?, ?, ?, ?, ?, ?, ?)
235         ");
236
237         foreach ($selectedItems as $item) {
238             mysqli_stmt_bind_param($itemStmt, "iisdis",
239                 $purchaseId,
240                 $item['product_id'],
241                 $item['name'],
242                 $item['price'],
243                 $item['quantity'],
244                 $item['image']
245             );
246
247             if (!mysqli_stmt_execute($itemStmt)) {
248                 throw new Exception('Failed to add purchase item: ' . mysqli_stmt_error(
249                     $itemStmt));
250             }
251         }
252
253         mysqli_stmt_close($itemStmt);
254
255         // Clear cart for current user only if using legacy format
256         if (!isset($input['action']) || $input['action'] !== 'create_from_selected_cart'
257         ) {
258             $clearCartStmt = mysqli_prepare($conn, "DELETE FROM cart WHERE user_id = ?");
259             mysqli_stmt_bind_param($clearCartStmt, "i", $user_id);
260             if (!mysqli_stmt_execute($clearCartStmt)) {
261                 throw new Exception('Failed to clear cart: ' . mysqli_stmt_error(
262                     $clearCartStmt));
263             }
264             mysqli_stmt_close($clearCartStmt);
265
266         // Commit transaction
267         mysqli_commit($conn);
268
269         echo json_encode([
270             'success' => true,
271             'message' => 'Purchase created successfully',
272             'order_number' => $orderNumber,
273             'purchase_id' => $purchaseId,
274             'total_amount' => $totalAmount
275         ]);
276
277     } catch (Exception $e) {
278         mysqli_rollback($conn);
279         throw $e;
280     }
281
282 }
```

PATCH – Update Orders

```

278     } elseif ($method === 'PATCH') {
279         // Update purchase or purchase items (only for current user)
280         $input = json_decode(file_get_contents('php://input'), true);
281
282         if (!isset($input['action'])) {
283             http_response_code(400);
284             echo json_encode(['error' => 'Missing action parameter']);
285             exit;
286         }
287
288 }
```

a. Update Quantities

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

```

288     $action = $input['action'];
289
290     if ($action === 'update_items') {
291         // Update quantities of items in a purchase
292         if (!isset($input['purchase_id'], $input['items'])) {
293             http_response_code(400);
294             echo json_encode(['error' => 'Missing purchase_id or items']);
295             exit;
296         }
297
298         // Check if purchase belongs to current user and can be edited
299         $statusStmt = mysqli_prepare($conn, "SELECT order_status FROM purchases WHERE
300         id = ? AND user_id = ?");
301         mysqli_stmt_bind_param($statusStmt, "ii", $input['purchase_id'], $user_id);
302         mysqli_stmt_execute($statusStmt);
303         $statusResult = mysqli_stmt_get_result($statusStmt);
304         $statusRow = mysqli_fetch_assoc($statusResult);
305         mysqli_stmt_close($statusStmt);
306
306         if (!$statusRow) {
307             http_response_code(404);
308             echo json_encode(['error' => 'Order not found or access denied']);
309             exit;
310         }
311
312         if ($statusRow['order_status'] !== 'pending') {
313             http_response_code(400);
314             echo json_encode(['error' => 'Order cannot be edited']);
315             exit;
316         }
317
318         mysqli_begin_transaction($conn);
319
320         try {
321             $updateStmt = mysqli_prepare($conn, "UPDATE purchase_items SET quantity = ?
322             WHERE id = ?");
323
323             foreach ($input['items'] as $item) {
324                 if ($item['quantity'] > 0) {
325                     mysqli_stmt_bind_param($updateStmt, "ii", $item['quantity'], $item[
326                     'id']);
327                     if (!mysqli_stmt_execute($updateStmt)) {
328                         throw new Exception('Failed to update item quantity: ' .
329                         mysqli_stmt_error($updateStmt));
330                     }
331                 }
332             }
333             mysqli_stmt_close($updateStmt);
334
334             // Recalculate total amount
335             $totalQuery =
336                 "SELECT SUM(price * quantity) as subtotal
337                 FROM purchase_items
338                 WHERE purchase_id = ?
339             ";
340             $totalStmt = mysqli_prepare($conn, $totalQuery);
341             mysqli_stmt_bind_param($totalStmt, "i", $input['purchase_id']);
342             mysqli_stmt_execute($totalStmt);
343             $totalResult = mysqli_stmt_get_result($totalStmt);
344             $totalRow = mysqli_fetch_assoc($totalResult);
345             mysqli_stmt_close($totalStmt);
346
346             $subtotal = (float)$totalRow['subtotal'];
347             $shippingCost = $subtotal >= 50.00 ? 0.00 : 5.00;
348             $newTotalAmount = $subtotal + $shippingCost;
349
350             // Update purchase total (verify user ownership again)
351             $updatePurchaseStmt = mysqli_prepare($conn, "
352                 UPDATE purchases
353                 SET total_amount = ?, shipping_cost = ?, updated_at = CURRENT_TIMESTAMP
354                 WHERE id = ? AND user_id = ?
355             ");

```

```

356             mysqli_stmt_bind_param($updatePurchaseStmt, "ddii", $newTotalAmount,
357                                         $shippingCost, $input['purchase_id'], $user_id);
358
359             if (!mysqli_stmt_execute($updatePurchaseStmt)) {
360                 throw new Exception('Failed to update purchase total: ' . mysqli_stmt_error($updatePurchaseStmt));
361             }
362             mysqli_stmt_close($updatePurchaseStmt);
363
364             mysqli_commit($conn);
365
366             echo json_encode([
367                 'success' => true,
368                 'message' => 'Order updated successfully',
369                 'new_total' => $newTotalAmount
370             ]);
371
372         } catch (Exception $e) {
373             mysqli_rollback($conn);
374             throw $e;
375         }

```

b. Cancel Order

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

```

376         } elseif ($action === 'cancel_order') {
377             // Cancel an order (only user's own orders)
378             if (!isset($input['purchase_id'])) {
379                 http_response_code(400);
380                 echo json_encode(['error' => 'Missing purchase_id']);
381                 exit;
382             }
383
384             // Check if order belongs to user and can be cancelled
385             $statusStmt = mysqli_prepare($conn, "SELECT order_status FROM purchases WHERE
386                                         id = ? AND user_id = ?");
387             mysqli_stmt_bind_param($statusStmt, "ii", $input['purchase_id'], $user_id);
388             mysqli_stmt_execute($statusStmt);
389             $statusResult = mysqli_stmt_get_result($statusStmt);
390             $statusRow = mysqli_fetch_assoc($statusResult);
391             mysqli_stmt_close($statusStmt);
392
393             if (!$statusRow) {
394                 http_response_code(404);
395                 echo json_encode(['error' => 'Order not found or access denied']);
396                 exit;
397             }
398
399             if (!in_array($statusRow['order_status'], ['pending', 'processing'])) {
400                 http_response_code(400);
401                 echo json_encode(['error' => 'Order cannot be cancelled']);
402                 exit;
403             }
404
405             $cancelStmt = mysqli_prepare($conn, "
406                                         UPDATE purchases
407                                         SET order_status = 'cancelled', updated_at = CURRENT_TIMESTAMP
408                                         WHERE id = ? AND user_id = ?
409                                     ");
410             mysqli_stmt_bind_param($cancelStmt, "ii", $input['purchase_id'], $user_id);
411
412             if (mysqli_stmt_execute($cancelStmt)) {
413                 echo json_encode([
414                     'success' => true,
415                     'message' => 'Order cancelled successfully'
416                 ]);
417             } else {
418                 throw new Exception('Failed to cancel order: ' . mysqli_stmt_error(
419                     $cancelStmt));
420             }
421             mysqli_stmt_close($cancelStmt);

```

DELETE – Remove Items

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

```
432     } elseif ($method === 'DELETE') {
433         // Delete purchase items or entire purchase (only user's own)
434         $input = json_decode(file_get_contents('php://input'), true);
435
436         if (!isset($input['action'])) {
437             http_response_code(400);
438             echo json_encode(['error' => 'Missing action parameter']);
439             exit;
440         }
441
442         $action = $input['action'];
443
444         if ($action === 'remove_item') {
445             // Remove specific item from purchase
446             if (!isset($input['item_id'])) {
447                 http_response_code(400);
448                 echo json_encode(['error' => 'Missing item_id']);
449                 exit;
450             }
451
452             // Get purchase_id and check if order belongs to user and can be edited
453             $itemQuery =
454                 "SELECT pi.purchase_id, p.order_status
455                 FROM purchase_items pi
456                 JOIN purchases p ON pi.purchase_id = p.id
457                 WHERE pi.id = ? AND p.user_id = ?";
458
459             $itemStmt = mysqli_prepare($conn, $itemQuery);
460             mysqli_stmt_bind_param($itemStmt, "ii", $input['item_id'], $user_id);
461             mysqli_stmt_execute($itemStmt);
462             $itemResult = mysqli_stmt_get_result($itemStmt);
463             $itemRow = mysqli_fetch_assoc($itemResult);
464             mysqli_stmt_close($itemStmt);
465
466             if (!$itemRow) {
467                 http_response_code(404);
468                 echo json_encode(['error' => 'Item not found or access denied']);
469                 exit;
470             }
471
472             if ($itemRow['order_status'] !== 'pending') {
473                 http_response_code(400);
474                 echo json_encode(['error' => 'Order cannot be edited']);
475                 exit;
476             }
477
478             mysqli_begin_transaction($conn);
479             try {
480                 // Delete the item
481                 $deleteStmt = mysqli_prepare($conn, "DELETE FROM purchase_items WHERE id = ?");
482                 mysqli_stmt_bind_param($deleteStmt, "i", $input['item_id']);
483
484                 if (!mysqli_stmt_execute($deleteStmt)) {
485                     throw new Exception('Failed to delete item: ' . mysqli_stmt_error(
486                         $deleteStmt));
487                 }
488                 mysqli_stmt_close($deleteStmt);
489
490                 // Check if any items remain
491                 $countStmt = mysqli_prepare($conn, "SELECT COUNT(*) as item_count FROM
492                     purchase_items WHERE purchase_id = ?");
493                 mysqli_stmt_bind_param($countStmt, "i", $itemRow['purchase_id']);
494                 mysqli_stmt_execute($countStmt);
495                 $countResult = mysqli_stmt_get_result($countStmt);
496                 $countRow = mysqli_fetch_assoc($countResult);
497                 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'],
506             $user_id);
507
508         if (!mysqli_stmt_execute($cancelStmt)) {
509             throw new Exception('Failed to cancel empty order: ' .
510                 mysqli_stmt_error($cancelStmt));
511         }
512         mysqli_stmt_close($cancelStmt);
513     } else {
514         // Recalculate total amount
515         $totalQuery = "
516             SELECT SUM(price * quantity) as subtotal
517             FROM purchase_items
518             WHERE purchase_id = ?
519         ";
520         $totalStmt = mysqli_prepare($conn, $totalQuery);
521         mysqli_stmt_bind_param($totalStmt, "i", $itemRow['purchase_id']);
522         mysqli_stmt_execute($totalStmt);
523         $totalResult = mysqli_stmt_get_result($totalStmt);
524         $totalRow = mysqli_fetch_assoc($totalResult);
525         mysqli_stmt_close($totalStmt);
526
527         $subtotal = (float)$totalRow['subtotal'];
528         $shippingCost = $subtotal >= 50.00 ? 0.00 : 5.00;
529         $newTotalAmount = $subtotal + $shippingCost;
530
531         // Update purchase total (verify user ownership)
532         $updatePurchaseStmt = mysqli_prepare($conn, "
533             UPDATE purchases
534             SET total_amount = ?, shipping_cost = ?, updated_at =
535             CURRENT_TIMESTAMP
536             WHERE id = ? AND user_id = ?
537         ");
538         mysqli_stmt_bind_param($updatePurchaseStmt, "ddii",
539             $newTotalAmount,
540             $shippingCost, $itemRow['purchase_id'], $user_id);
541
542         if (!mysqli_stmt_execute($updatePurchaseStmt)) {
543             throw new Exception('Failed to update purchase total: ' .
544                 mysqli_stmt_error($updatePurchaseStmt));
545         }
546         mysqli_stmt_close($updatePurchaseStmt);
547     }
548
549     mysqli_commit($conn);
550
551     echo json_encode([
552         'success' => true,
553         'message' => 'Item removed successfully'
554     ]);
555
556 } catch (Exception $e) {
557     mysqli_rollback($conn);
558     throw $e;
559 }

```

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.

```
4      data() {
5          return {
6              purchases: [],
7              loading: false,
8              editingOrder: null,
9              searchQuery: '',
10             statusFilter: '',
11             sortBy: 'newest',
12             confirmDialog: false, // Controls visibility of the custom confirmation modal
13             confirmTitle: '',    // Title for the confirmation modal
14             confirmMessage: '', // Message for the confirmation modal
15             confirmAction: null, // Callback function to execute on confirmation
16         },
17     },
```

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**.

```
19      computed: {
20          filteredPurchases() {
21              let filtered = this.purchases;
22
23              // Filter by search query
24              if (this.searchQuery) {
25                  const query = this.searchQuery.toLowerCase();
26                  filtered = filtered.filter(purchase =>
27                      purchase.order_number.toLowerCase().includes(query) ||
28                      purchase.items.some(item => item.name.toLowerCase().includes(query))
29                  );
30              }
31
32              // Filter by status
33              if (this.statusFilter) {
34                  filtered = filtered.filter(purchase => purchase.order_status === this.
35                  statusFilter);
36              }
37
38              // Sort results
39              return filtered.sort((a, b) => {
40                  switch (this.sortBy) {
41                      case 'oldest':
42                          return new Date(a.order_date) - new Date(b.order_date);
43                      case 'amount_high':
44                          return b.total_amount - a.total_amount;
45                      case 'amount_low':
46                          return a.total_amount - b.total_amount;
47                      default: // newest
48                          return new Date(b.order_date) - new Date(a.order_date);
49                  }
50              });
51      },
```

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
110        // removeItemFromOrder
111        return fetch('purchase_api.php')
112          .then(res => {
113            if (!res.ok) {
114              throw new Error(`HTTP error! status: ${res.status}`);
115            }
116            return res.json();
117          })
118          .then(data => {
119            this.purchases = data.map(purchase => ({
120              ...purchase,
121              items: purchase.items.map(item => ({
122                ...item,
123                newQuantity: item.quantity
124              }));
125            }));
126            this.loading = false;
127            return data; // Important: return data for subsequent .then() calls
128          })
129          .catch(err => {
130            console.error('Failed to load purchases:', err);
131            this.showInfoMessage('Error', 'Failed to load purchase history. Please
132            refresh the page.');
133            this.loading = false;
134            throw err; // Re-throw to propagate the error
135          });
136      },
137    
```

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        }));
158      },
159    
```

```

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 ===
209                             purchaseId);
210                         // If the purchase no longer has items (and implicitly
211                         // cancelled by backend)
212                         if (updatedPurchase && updatedPurchase.items.length === 0) {
213                             this.editingOrder = null; // Exit edit mode
214                         }
215                     });
216                     .catch(err => {
217                         console.error('Failed to remove item:', err);
218                         this.showInfoMessage('Error', 'Failed to remove item. Please try again.');// Using custom message
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
244                 change
245                 this.showInfoMessage('Success', 'Order cancelled successfully.');
246             }) // Using custom message
247             .catch(err => {
248                 console.error('Failed to cancel order:', err);
249                 this.showInfoMessage('Error', 'Failed to cancel order. Please
250                 try again.');// Using custom message
251             });
252         );
253     },
254 }
```

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(
271             'confirmationModal'));
272         modalElement.show();
273     },
274
275     closeConfirm() {
276         this.confirmDialog = false; // Reset Vue state
277         this.confirmTitle = '';
278         this.confirmMessage = '';
279         this.confirmAction = null;
280         // Use Bootstrap's JavaScript API to hide the modal
281         const modalElement = bootstrap.Modal.getInstance(document.getElementById(
282             'confirmationModal'));
283         if (modalElement) {
284             modalElement.hide();
285         }
286     },
287     executeConfirmAction() {
288         if (this.confirmAction) {
289             this.confirmAction();
290         }
291     this.closeConfirm(); // Close the modal after action
292 }
```

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
94           // corrupted
95           console.warn('Could not load filter state:', e);
96         }
97       },
98
99       // Clear saved filter state
100      clearFilterState() {
101        try {
102          sessionStorage.removeItem('purchaseFilters');
103        } catch (e) {
104          console.warn('Could not clear filter state:', e);
105        }
106      },
107    },
108  },
109}
```

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   },
263 }
```

- `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
296     // now
297     console.log(`${title}: ${message}`);
298     // For a basic visual cue without native alerts:
299     // alert(`${title}: ${message}`); // Revert to custom UI later if needed
300   },
301 }
```

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';?>.

```
style.css profile.php profile_api.php profile.js
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Profile Page</title>
6      <meta name="viewport" content="width=device-width, initial-scale=1" />
7
8      <!-- Vuetify CSS and Bootstrap-->
9      <link href="https://cdn.jsdelivr.net/npm/vuetify@3.5.0/dist/vuetify.min.css" rel="stylesheet">
10     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">
11
12     <!-- Font Awesome & Fonts -->
13     <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.0/css/all.min.css" rel="stylesheet">
14     <link href="https://fonts.googleapis.com/css2?family=Quicksand:wght@500:700&display=swap" rel="stylesheet">
15     <link href="https://cdn.jsdelivr.net/npm/@mdi/font@latest/css/materialdesignicons.min.css" rel="stylesheet">
16
17     <!-- Vue & Vuetify JS -->
18     <script src="https://unpkg.com/vue@3/dist/vue.global.prod.js"></script>
19     <script src="https://cdn.jsdelivr.net/npm/vuetify@3.5.0/dist/vuetify.min.js"></script>
20
21     <!-- Custom Styles & Script -->
22     <link rel="stylesheet" href="style/style.css">
23 </head>
24 <body>
25     <?php include 'commons/header.php'; ?>
26
27     <?php include 'commons/footer.php'; ?>
28     <!-- Load Bootstrap JS first -->
29     <script src=
30         "https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>
31
32     <!-- Load your profile Vue app last -->
33     <script src="profile.js"></script>
34 </body>
35 </html>
```

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.

```
27     <div id="profile-app"></div>
28
29     <?php include 'commons/footer.php'; ?>
30
31     <script>
32         window.currentUser = <?php echo json_encode($user); ?>;
33     </script>
```

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`.

```
1 <?php
2     session_start();
3     header('Content-Type: application/json');
4
5     // Database connection
6     $host = 'localhost';
7     $db = 'project';
8     $user = 'root';
9     $pass = '';
10    $conn = new mysqli($host, $user, $pass, $db);
11
12    if ($conn->connect_error) {
13        echo json_encode(['error' => 'Database connection failed']);
14        exit;
15    }
16
17    // Check authentication
18    if (!isset($_SESSION['user_id'])) {
19        echo json_encode(['error' => 'Unauthorized']);
20        exit;
21    }
22
23    $user_id = $_SESSION['user_id'];
24
25    // Parse request
26    $input = json_decode(file_get_contents("php://input"), true);
27    $action = $_GET['action'] ?? ($input['action'] ?? '');
28
29    // Log for debugging
30    error_log("Action: " . $action);
31    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.

```
33 // ===== USER PROFILE OPERATIONS =====
34
35 if ($action === 'update_user') {
36     $user = $input['user'] ?? [];
37     $name = trim($user['name'] ?? '');
38     $email = trim($user['email'] ?? '');
39     $password = $user['password'] ?? ''; // Keep as string, can be empty
40     $password_confirm = $user['password_confirm'] ?? ''; // New field
41
42     error_log("User update data: " . json_encode($user));
43
44     if (!$name || !$email) {
45         echo json_encode(['error' => 'Name and email are required', 'fields' => ['name' => $name, 'email' => $email]]);
46         exit;
47     }
}
```

```

49     // Check if email is valid
50     if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
51         echo json_encode(['error' => 'Invalid email format']);
52         exit;
53     }
54
55     // Check if email is already taken by another user
56     $stmt = $conn->prepare("SELECT id FROM users WHERE email = ? AND id != ?");
57     $stmt->bind_param("si", $email, $user_id);
58     $stmt->execute();
59     $result = $stmt->get_result();
60
61     if ($result->num_rows > 0) {
62         echo json_encode(['error' => 'Email is already taken by another user']);
63         exit;
64     }
65
66     // If password is provided, validate it and its confirmation
67     if (!empty($password)) {
68         if ($password !== $password_confirm) {
69             echo json_encode(['error' => 'Password and confirmation do not match']);
70             exit;
71         }
72
73         // Validate password strength
74         if (strlen($password) < 8) {
75             echo json_encode(['error' => 'Password must be at least 8 characters']);
76             exit;
77         }
78
79         if (!preg_match('/[A-Z]/', $password)) {
80             echo json_encode(['error' => 'Password must contain at least one uppercase letter']);
81             exit;
82         }
83         if (!preg_match('/[0-9]/', $password)) {
84             echo json_encode(['error' => 'Password must contain at least one number']);
85             exit;
86         }
87
88         if (!preg_match('/![#@#$%^&*()_+=\[\]{};\'":\\|\.,<>\/?]/', $password)) {
89             echo json_encode(['error' => 'Password must contain at least one special character']);
90             exit;
91         }
92
93         $hashed = password_hash($password, PASSWORD_DEFAULT);
94         $stmt = $conn->prepare("UPDATE users SET name = ?, email = ?, password = ? WHERE id = ?");
95         $stmt->bind_param("sssi", $name, $email, $hashed, $user_id);
96     } else {
97         // Update without changing password
98         $stmt = $conn->prepare("UPDATE users SET name = ?, email = ? WHERE id = ?");
99         $stmt->bind_param("ssi", $name, $email, $user_id);
100    }
101
102    if ($stmt->execute()) {
103        // Update session data
104        $_SESSION['user_name'] = $name;
105        $_SESSION['user_email'] = $email;
106
107        echo json_encode(['success' => true, 'message' => 'Profile updated successfully']);
108    } else {
109        echo json_encode(['error' => 'Failed to update profile: ' . $conn->error]);
110    }
111    exit;
112}
113

```

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.

```
115 // ===== ADDRESS OPERATIONS =====
116
117 // Get all addresses for the current user
118 if ($action === 'get_addresses') {
119     $stmt = $conn->prepare("SELECT * FROM addresses WHERE user_id = ? ORDER BY created_at
DESC");
120     $stmt->bind_param("i", $user_id);
121     $stmt->execute();
122     $result = $stmt->get_result();
123
124     $addresses = [];
125     while ($row = $result->fetch_assoc()) {
126         $addresses[] = $row;
127     }
128
129     echo json_encode(['success' => true, 'addresses' => $addresses]);
130     exit;
131 }
132
133 // Create a new address
134 if ($action === 'create_address') {
135     $address = $input['address'] ?? [];
136
137     $name = trim($address['name'] ?? '');
138     $phone = trim($address['phone'] ?? '');
139     $unit_number = trim($address['unit_number'] ?? '');
140     $street = trim($address['street'] ?? '');
141     $city = trim($address['city'] ?? '');
142     $state = trim($address['state'] ?? '');
143     $postcode = trim($address['postcode'] ?? '');
144     $country = trim($address['country'] ?? 'Malaysia');
145
146     // Validate required fields
147     if (!$name || !$phone || !$street || !$city || !$state || !$postcode || !$country) {
148         echo json_encode(['error' => 'All required fields must be filled (name, phone,
street, city, state, postcode, country)']);
149         exit;
150     }
151
152     // Validate phone number - allow international formats with country codes
153     if (!preg_match('/^[\+]?[\d\s\-\(\)]+$/', $phone) || strlen(preg_replace(
154         '/[\s\-\(\)]+/', '', $phone)) < 7) {
155         echo json_encode(['error' => 'Please enter a valid phone number']);
156         exit;
157     }
158
159     // Validate postcode based on country
160     if ($country === 'Malaysia') {
161         if (!preg_match('/^\d{5}$/', $postcode)) {
162             echo json_encode(['error' => 'Malaysian postcode must be 5 digits']);
163             exit;
164         }
165     } else {
166         // Generic postcode validation for other countries
167         if (!preg_match('/^[\w\-\s]{3,10}$/', $postcode)) {
168             echo json_encode(['error' => 'Please enter a valid postcode']);
169             exit;
170         }
171     }
172 }
```

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,
174 street, city, state, postcode, country) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)");
175 $stmt->bind_param("issssssss", $user_id, $name, $phone, $unit_number, $street, $city,
176 $state, $postcode, $country);
177
178 if ($stmt->execute()) {
179     echo json_encode(['success' => true, 'message' => 'Address created successfully',
180     'address_id' => $conn->insert_id]);
181 } else {
182     echo json_encode(['error' => 'Failed to create address: ' . $conn->error]);
183 }
184 exit;
```

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');
197
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,
206         street, city, state, postcode, country)']);
207         exit;
208     }
209
210     // Validate phone number - allow international formats with country codes
211     if (!preg_match('/^[\+]?[\d\s\-\(\)]+$/', $phone) || strlen(preg_replace(
212         '/[\s\-\(\)]+/', '', $phone)) < 7) {
213         echo json_encode(['error' => 'Please enter a valid phone number']);
214         exit;
215     }
216
217     // Validate postcode based on country
218     if ($country === 'Malaysia') {
219         if (!preg_match('/^\d{5}$/', $postcode)) {
220             echo json_encode(['error' => 'Malaysian postcode must be 5 digits']);
221             exit;
222         }
223     } else {
224         // Generic postcode validation for other countries
225         if (!preg_match('/^[\A-Za-z0-9\s\-\']{3,10}$/', $postcode)) {
226             echo json_encode(['error' => 'Please enter a valid postcode']);
227             exit;
228         }
229     }
230 }
```

```

229     // Check if address belongs to current user
230     $stmt = $conn->prepare("SELECT id FROM addresses WHERE id = ? AND user_id = ?");
231     $stmt->bind_param("ii", $address_id, $user_id);
232     $stmt->execute();
233     $result = $stmt->get_result();
234
235     if ($result->num_rows === 0) {
236         echo json_encode(['error' => 'Address not found or unauthorized']);
237         exit;
238     }
239     // Update address
240     $stmt = $conn->prepare("UPDATE addresses SET name = ?, phone = ?, unit_number = ?,
241     street = ?, city = ?, state = ?, postcode = ?, country = ? WHERE id = ? AND user_id = ?");
242     $stmt->bind_param("ssssssssii", $name, $phone, $unit_number, $street, $city, $state,
243     $postcode, $country, $address_id, $user_id);
244
245     if ($stmt->execute()) {
246         echo json_encode(['success' => true, 'message' => 'Address updated successfully']);
247     } else {
248         echo json_encode(['error' => 'Failed to update address: ' . $conn->error]);
249     }
250     exit;
251 }
```

6. Delete Address (action=delete_address)

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

```

252     // Delete an address
253     if ($action === 'delete_address') {
254         $address_id = intval($input['address_id'] ?? 0);
255
256         if (!$address_id) {
257             echo json_encode(['error' => 'Address ID is required']);
258             exit;
259         }
260
261         // Check if address belongs to current user
262         $stmt = $conn->prepare("SELECT id FROM addresses WHERE id = ? AND user_id = ?");
263         $stmt->bind_param("ii", $address_id, $user_id);
264         $stmt->execute();
265         $result = $stmt->get_result();
266
267         if ($result->num_rows === 0) {
268             echo json_encode(['error' => 'Address not found or unauthorized']);
269             exit;
270         }
271
272         // Delete address
273         $stmt = $conn->prepare("DELETE FROM addresses WHERE id = ? AND user_id = ?");
274         $stmt->bind_param("ii", $address_id, $user_id);
275
276         if ($stmt->execute()) {
277             echo json_encode(['success' => true, 'message' => 'Address deleted successfully']);
278         } else {
279             echo json_encode(['error' => 'Failed to delete address: ' . $conn->error]);
280         }
281     exit;
282 }
```

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.

```
6  createApp({
7    setup() {
8      const user = reactive({ ...window.currentUser });
9      const userForm = reactive({
10        ...user,
11        password: '', // Ensure password is empty initially for security
12        password_confirm: '' // New field for password confirmation
13      });
14
15      // Form refs for Vuetify validation
16      const userFormRef = ref(null);
17      const addressFormRef = ref(null);
18
19      // Controls dropdown visibility for profile editing
20      const showEditUser = ref(false);
21
22      // New refs for password visibility toggles
23      const showPassword = ref(false);
24      const showConfirmPassword = ref(false);
25
26      // Address management state
27      const addresses = ref([]);
28      const showAddressForm = ref(false);
29      const editingAddress = ref(null);
30      const addressForm = reactive({
31        name: '',
32        phone: '',
33        country_code: '+60', // Default to Malaysia
34        unit_number: '',
35        street: '',
36        city: '',
37        state: '',
38        postcode: '',
39        country: 'Malaysia' // Default to Malaysia
40      });
41
42      // Confirmation modal state
43      const confirmTitle = ref('');
44      const confirmMessage = ref('');
45      const confirmAction = ref(null);
```

```

47      // Country codes data
48      const countryCodes = [
49          { code: '+60', country: 'Malaysia', flag: 'MY' },
50          { code: '+65', country: 'Singapore', flag: 'SG' },
51          { code: '+62', country: 'Indonesia', flag: 'ID' },
52          { code: '+66', country: 'Thailand', flag: 'TH' },
53          { code: '+84', country: 'Vietnam', flag: 'VN' },
54          { code: '+63', country: 'Philippines', flag: 'PH' },
55          { code: '+673', country: 'Brunei', flag: 'BN' },
56          { code: '+95', country: 'Myanmar', flag: 'MM' },
57          { code: '+855', country: 'Cambodia', flag: 'KH' },
58          { code: '+856', country: 'Laos', flag: 'LA' },
59          { code: '+1', country: 'USA/Canada', flag: 'US' },
60          { code: '+44', country: 'United Kingdom', flag: 'GB' },
61          { code: '+61', country: 'Australia', flag: 'AU' },
62          { code: '+86', country: 'China', flag: 'CN' },
63          { code: '+91', country: 'India', flag: 'IN' },
64          { code: '+81', country: 'Japan', flag: 'JP' },
65          { code: '+82', country: 'South Korea', flag: 'KR' }
66      ];
67      // Countries list
68      const countries = [
69          'Malaysia', 'Singapore', 'Indonesia', 'Thailand', 'Vietnam', 'Philippines',
70          'Brunei', 'Myanmar', 'Cambodia', 'Laos', 'United States', 'Canada',
71          'United Kingdom', 'Australia', 'China', 'India', 'Japan', 'South Korea',
72          'Germany', 'France', 'Italy', 'Spain', 'Netherlands', 'Belgium', 'Switzerland',
73          'Sweden', 'Norway', 'Denmark', 'Finland', 'New Zealand', 'Brazil', 'Argentina',
74          'Chile', 'Mexico', 'South Africa', 'Egypt', 'UAE', 'Saudi Arabia', 'Turkey'
75      ];
76
77
78      // Malaysian states
79      const malaysianStates = [
80          'Johor', 'Kedah', 'Kelantan', 'Malacca', 'Negeri Sembilan', 'Pahang',
81          'Penang', 'Perak', 'Perlis', 'Sabah', 'Sarawak', 'Selangor', 'Terengganu',
82          'Federal Territory of Kuala Lumpur', 'Federal Territory of Labuan', 'Federal
83          Territory of Putrajaya'
84      ];
85      // Other states/provinces by country
86      const statesByCountry = {
87          'Malaysia': malaysianStates,
88          'Singapore': ['Central Region', 'East Region', 'North Region', 'North-East
89          Region', 'West Region'],
90          'Indonesia': ['Jakarta', 'West Java', 'Central Java', 'East Java', 'Bali',
91          'Sumatra', 'Kalimantan', 'Sulawesi'],
92          'Thailand': ['Bangkok', 'Chiang Mai', 'Phuket', 'Pattaya', 'Krabi', 'Samui',
93          'Hua Hin'],
94          'Philippines': ['Metro Manila', 'Cebu', 'Davao', 'Iloilo', 'Cagayan de Oro',
95          'Zamboanga'],
96          'United States': ['Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California',
97          'Colorado', 'Connecticut', 'Delaware', 'Florida', 'Georgia'],
98          'Canada': ['Ontario', 'Quebec', 'British Columbia', 'Alberta', 'Manitoba',
99          'Saskatchewan', 'Nova Scotia', 'New Brunswick'],
100         'Australia': ['New South Wales', 'Victoria', 'Queensland', 'Western Australia',
101          'South Australia', 'Tasmania', 'Northern Territory', 'Australian Capital
102          Territory']
103     };
104
105     // Get states for selected country
106     const getStatesForCountry = (country) => {
107         return statesByCountry[country] || [];
108     };
109
110     // Watch for country changes to update state options
111     const onCountryChange = () => {
112         const states = getStatesForCountry(addressForm.country);
113         if (states.length > 0 && !states.includes(addressForm.state)) {
114             addressForm.state = '';
115         }
116     };

```

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.

```
// ===== USER PROFILE FUNCTIONS =====
// Actual save user function (called after confirmation)
const performSaveUser = async () => {
    try {
        // Only include password and password_confirm if password field is not empty
        const payloadUser = {
            name: userForm.name,
            email: userForm.email
        };

        if (userForm.password) {
            payloadUser.password = userForm.password;
            payloadUser.password_confirm = userForm.password_confirm;
        }

        const payload = {
            action: 'update_user',
            user: payloadUser
        };

        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) {
            Object.assign(user, {
                name: userForm.name,
                email: userForm.email
            });

            userForm.password = '';
            userForm.password_confirm = ''; // Clear confirmation field
            showEditUser.value = false;

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

            showConfirm('Success!', 'Your profile has been updated successfully!', null);
        } else {
            showConfirm('Error', data.error || "Failed to update profile. Please try again.", null);
        }
    } catch (err) {
        console.error("Error saving user info:", err);
        showConfirm('Error', 'An error occurred while saving your profile. Please try again.', null);
    }
}
```

```

270     );
271
272     // Save user with validation and confirmation
273     const saveUser = async () => {
274       const { valid } = await userFormRef.value.validate();
275       if (!valid) return;
276
277       showConfirm(
278         'Confirm Changes',
279         'Are you sure you want to save these changes to your profile?',
280         performSaveUser
281       );
282     };
283
284     // Cancel editing and reset form
285     const cancelEdit = () => {
286       Object.assign(userForm, user);
287       userForm.password = '';
288       userForm.password_confirm = ''; // Clear confirmation field
289       showEditUser.value = false;
290
291       nextTick(() => {
292         userFormRef.value?.resetValidation();
293       });
294     };

```

3. Address Management

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

```

296     // ===== ADDRESS MANAGEMENT FUNCTIONS =====
297
298
299     // Load addresses from server
300     const loadAddresses = async () => {
301       try {
302         const res = await fetch('profile_api.php', {
303           method: 'POST',
304           headers: { 'Content-Type': 'application/json' },
305           body: JSON.stringify({ action: 'get_addresses' })
306         });
307
308         const data = await res.json();
309         if (data.success) {
310           addresses.value = data.addresses || [];
311         } else {
312           console.error('Failed to load addresses:', data.error);
313         }
314       } catch (err) {
315         console.error('Error loading addresses:', err);
316       }
317     };

```

- 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.

```

318     // Reset address form
319     const resetAddressForm = () => {
320       Object.assign(addressForm, {
321         name: '',
322         phone: '',
323         country_code: '+60',
324         unit_number: '',
325         street: '',
326         city: '',
327         state: '',
328         postcode: '',
329         country: 'Malaysia'
330       });
331       editingAddress.value = null;
332     };

```

```

334     // Show add address form
335     const showAddAddress = () => {
336         resetAddressForm();
337         showAddressForm.value = true;
338     };
339
340     // Show edit address form
341     const showEditAddress = (address) => {
342         // Parse the phone number to separate country code and number
343         let countryCode = '+60';
344         let phoneNumber = address.phone;
345
346         // Try to extract country code from phone number
347         for (const cc of countryCodes) {
348             if (address.phone.startsWith(cc.code)) {
349                 countryCode = cc.code;
350                 phoneNumber = address.phone.substring(cc.code.length).trim();
351                 break;
352             }
353         }
354
355         Object.assign(addressForm, {
356             ...address,
357             phone: phoneNumber,
358             country_code: countryCode,
359             state: address.state || '',
360             country: address.country || 'Malaysia'
361         });
362         editingAddress.value = address.id;
363         showAddressForm.value = true;
364     };
365
366     // Cancel address form
367     const cancelAddressForm = () => {
368         resetAddressForm();
369         showAddressForm.value = false;
370
371         nextTick(() => {
372             addressFormRef.value?.resetValidation();
373         });
374     };

```

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

```

376     // Save address (create or update)
377     const saveAddress = async () => {
378         const { valid } = await addressFormRef.value.validate();
379         if (!valid) return;
380
381         try {
382             // Combine country code and phone number
383             const fullPhoneNumber = addressForm.country_code + addressForm.phone;
384
385             const payload = {
386                 action: editingAddress.value ? 'update_address' : 'create_address',
387                 address: {
388                     ...addressForm,
389                     phone: fullPhoneNumber // Send combined phone number
390                 }
391             };
392
393             if (editingAddress.value) {
394                 payload.address.id = editingAddress.value;
395             }
396         }
397     };

```

```

397         const res = await fetch('profile_api.php', {
398             method: 'POST',
399             headers: { 'Content-Type': 'application/json' },
400             body: JSON.stringify(payload)
401         });
402
403         const data = await res.json();
404
405         if (data.success) {
406             await loadAddresses(); // Reload addresses
407             cancelAddressForm();
408             showConfirm(
409                 'Success!',
410                 editingAddress.value ? 'Address updated successfully!' : 'Address
411                 added successfully!',
412                 null
413             );
414         } else {
415             showConfirm('Error', data.error || 'Failed to save address. Please try
416             again.', null);
417         }
418     } catch (err) {
419         console.error('Error saving address:', err);
420         showConfirm('Error', 'An error occurred while saving the address. Please
421             try again.', null);
422     }
423 };

```

- **Delete Address:**

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

```

422         // Delete address with confirmation
423         const confirmDeleteAddress = (address) => {
424             showConfirm(
425                 'Delete Address',
426                 'Are you sure you want to delete the address for ${address.name}?',
427                 () => deleteAddress(address.id)
428             );
429
430
431         // Actually delete the address
432         const deleteAddress = async (addressId) => {
433             try {
434                 const res = await fetch('profile_api.php', {
435                     method: 'POST',
436                     headers: { 'Content-Type': 'application/json' },
437                     body: JSON.stringify({
438                         action: 'delete_address',
439                         address_id: addressId
440                     })
441                 });
442
443                 const data = await res.json();
444
445                 if (data.success) {
446                     await loadAddresses(); // Reload addresses
447                     showConfirm('Success!', 'Address deleted successfully!', null);
448                 } else {
449                     showConfirm('Error', data.error || 'Failed to delete address. Please
450                     try again.', null);
451                 }
452             } catch (err) {
453                 console.error('Error deleting address:', err);
454                 showConfirm('Error', 'An error occurred while deleting the address. Please
455                     try again.', null);
456             }
457         };

```

4. Validation Systems

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

```
110 // ===== USER PROFILE VALIDATION =====
111
112 // Rules for Vuetify's validation system
113 const userRules = {
114   name: [
115     v => !!v.trim() || 'Name is required',
116     v => v.trim().length >= 2 || 'Name must be at least 2 characters',
117     v => /^[a-zA-Z\s]+$/ test(v.trim()) || 'Name must contain only alphabetic
118     characters'
119   ],
120   email: [
121     v => !!v.trim() || 'Email is required',
122     v => /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\$/ test(v.trim()) || 'Please enter a valid email address'
123   ],
124   password: [
125     // Password is optional on profile update, but if entered, must meet
126     // criteria
127     v => !v || v.length >= 8 || 'Password must be at least 8 characters',
128     v => !v || /[A-Z]/.test(v) || 'Password must contain at least one uppercase
129     letter',
130     v => !v || /[0-9]/.test(v) || 'Password must contain at least one number',
131     v => !v || /[^#\$%^&*()_+=\{\}!":\\\|,.<>\?]/.test(v) || 'Password
132     must contain at least one special character'
133   ],
134   password_confirm: [
135     v => {
136       // Only require confirmation if a new password is being set
137       if (userForm.password) {
138         return !!v || 'Password confirmation is required';
139       }
140       return true; // Not required if password field is empty
141     },
142     v => {
143       // Only validate match if a new password is being set
144       if (userForm.password) {
145         return v === userForm.password || 'Passwords do not match';
146       }
147       return true; // No need to match if password field is empty
148     }
149   ],
150 };
151 
```

- Address Details Validation

```
148 // ===== ADDRESS VALIDATION =====
149
150 const addressRules = {
151   name: [
152     v => !!v.trim() || 'Name is required',
153     v => v.trim().length >= 2 || 'Name must be at least 2 characters'
154   ],
155   phone: [
156     v => !!v.trim() || 'Phone number is required',
157     v => /^[\\d\\s\\-\\+\\(\\)]+$/ test(v.trim()) || 'Please enter a valid phone
158     number',
159     v => v.trim().length >= 7 || 'Phone number must be at least 7 digits',
160     v => v.trim().length <= 15 || 'Phone number must not exceed 15 digits'
161   ],
162   street: [
163     v => !!v.trim() || 'Street address is required',
164     v => v.trim().length >= 5 || 'Please enter a complete street address'
165   ],
166 };
```

```

165     city: [
166       v => !!v.trim() || 'City is required',
167       v => v.trim().length >= 2 || 'City must be at least 2 characters'
168     ],
169     state: [
170       v => !!v.trim() || 'State/Province is required'
171     ],
172     postcode: [
173       v => !!v.trim() || 'Postcode is required',
174       v => {
175         const code = v.trim();
176         // Malaysian postcode validation
177         if (addressForm.country === 'Malaysia') {
178           return /\^\d{5}\$/.test(code) || 'Malaysian postcode must be 5 digits'
179           ;
180         }
181         // Generic postcode validation for other countries
182         return /^[A-Za-z0-9\s\-\-]{3,10}\$/.test(code) || 'Please enter a valid
183         postcode';
184       }
185     ],
186     country: [
187       v => !!v.trim() || 'Country is required'
188     ]
189   };

```

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()

```

189 // ===== CONFIRMATION MODAL =====
190
191 // Show confirmation dialog
192 const showConfirm = (title, message, action) => {
193   confirmTitle.value = title;
194   confirmMessage.value = message;
195   confirmAction.value = action;
196
197   // Use Bootstrap modal
198   const modal = new bootstrap.Modal(document.getElementById('confirmationModal'));
199   modal.show();
200 };
201
202 // Execute the confirmed action
203 const executeConfirmAction = () => {
204   if (confirmAction.value) {
205     confirmAction.value();
206   }
207   closeConfirm();
208 };
209
210 // Close confirmation modal
211 const closeConfirm = () => {
212   const modal = bootstrap.Modal.getInstance(document.getElementById(
213     'confirmationModal'));
214   if (modal) {
215     modal.hide();
216   }
217   confirmAction.value = null;
218 };

```

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 v-if="!showEditUser" class="profile-btn btn-brown" prepend-icon="mdi-square-edit-outline" @click="showEditUser = true"> 506             Edit Info 507           </v-btn> 508         </div> 509       <div v-if="!showEditUser" class="user-info-display"> 510         <p><strong>Name:</strong> {{ user.name }}</p> 511         <p><strong>Email:</strong> {{ user.email }}</p> 512       </div> 513     </v-card> 514   </v-container> 515 </v-main> 516 </v-app> 517 </div> 518 </div> 519 </div> 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 label="Name" v-model="userForm.name" :rules="userRules.name" variant="outlined" class="mb-3"> 524       </v-text-field> 525     <v-text-field label="Email" v-model="userForm.email" :rules="userRules.email" variant="outlined" class="mb-3"> 526       </v-text-field> 527     <v-text-field label="New Password (leave empty to keep current)" v-model="userForm.password" :rules="userRules.password" :type="showPassword ? 'text' : 'password'" variant="outlined" class="mb-3"> 528       hint="Leave empty if you don't want to change your password" 529       persistent-hint 530       :append-inner-icon="showPassword ? 'mdi-eye' : 'mdi-eye-off'" 531       @click:append-inner="showPassword = !showPassword" 532     </v-text-field> 533   </v-form> 534 </div> 535 </div> 536 </div> 537 </div> 538 </div> 539 </div> 540 </div> 541 </div> 542 </div> 543 </div> 544 </div> 545 </div> 546 </div> 547 </div> 548 </div>
```

```

549 <v-text-field
550   label="Confirm New Password"
551   v-model="userForm.password_confirm"
552   :rules="userRules.password_confirm"
553   :type="showConfirmPassword ? 'text' : 'password'"
554   variant="outlined"
555   class="mb-4"
556   :disabled="!userForm.password"
557   :append-inner-icon="showConfirmPassword ? 'mdi-eye' :
558   'mdi-eye-off'"
559   @click:append-inner="showConfirmPassword =
560   !showConfirmPassword"
561 ></v-text-field>
562 <div class="d-flex gap-2">
563   <v-btn class="profile-btn success-btn"
564     prepend-icon="mdi-content-save"
565     @click="saveUser">
566     Save Changes
567   </v-btn>
568   <v-btn class="profile-btn cancel-btn"
569     @click="cancelEdit">
570     Cancel
571   </v-btn>
572 </div>
573 </v-form>
574 </div>
575 </v-card>
576
577 <!-- Shipping Addresses Section -->
578 <v-card class="pa-6 custom-card">
579   <div class="d-flex justify-space-between align-center mb-4">
580     <h2 class="section-title">Shipping Addresses</h2>
581     <v-btn
582       v-if="!showAddressForm"
583       class="profile-btn btn-brown"
584       @click="showAddAddress"
585       prepend-icon="mdi-plus"
586     >
587       Add Address
588     </v-btn>
589   </div>
590 <!-- Address Form -->
591 <div v-if="showAddressForm" class="mb-6">
592   <h4 class="mb-4">{{ editingAddress ? 'Edit Address' : 'Add New
593   Address' }}</h4>
594   <v-form ref="addressFormRef" lazy-validation>
595     <v-row>
596       <v-col cols="12" md="6">
597         <v-text-field
598           label="Full Name"
599           v-model="addressForm.name"
600           :rules="addressRules.name"
601           variant="outlined"
602         ></v-text-field>
603       </v-col>
604       <v-col cols="12" md="6">
605         <div class="d-flex gap-2">
606           <v-select
607             label="Country Code"
608             v-model="addressForm.country_code"
609             :items="countryCodes"
610             item-title="country"
611             item-value="code"
612             variant="outlined"
613             style="flex: 0 0 140px; max-height: 10px;"
614             :menu-props="{ maxHeight: 300 }"
615           >
616             <template v-slot:selection="{ item }">
617               <span>{{ item.raw.flag }} {{ item.raw.code }}</span>
618             </template>
619             <template v-slot:item="{ item, props }">
620               <v-list-item v-bind="props">
621                 <template v-slot:prepend>
622                   <span class="me-2">{{ item.raw.flag }}</span>
623                 </template>
624               </v-list-item>
625             </template>
626           </v-select>
627         </div>
628       </v-col>
629     </v-row>
630   </v-form>
631 </div>

```

```

621           <v-list-item-title>{{ item.raw.code }} - {{ item.raw.country }}</v-list-item-title>
622           </v-list-item>
623       </template>
624   </v-select>
625   <v-text-field
626     label="Phone Number"
627     v-model="addressForm.phone"
628     :rules="addressRules.phone"
629     variant="outlined"
630     style="flex: 1;">
631   </v-text-field>
632 </div>
633 </v-col>
634 </v-row>
635 <v-row>
636   <v-col cols="12" md="4">
637     <v-text-field
638       label="Unit Number (Optional)"
639       v-model="addressForm.unit_number"
640       variant="outlined"
641       hint="House, Apartment, suite, unit, etc."
642       persistent-hint>
643     </v-text-field>
644   </v-col>
645   <v-col cols="12" md="8">
646     <v-text-field
647       label="Street Address"
648       v-model="addressForm.street"
649       :rules="addressRules.street"
650       variant="outlined">
651     </v-text-field>
652   </v-col>
653 </v-row>
654 <v-row>
655   <v-col cols="12" md="4">
656     <v-text-field
657       label="City"
658       v-model="addressForm.city"
659       :rules="addressRules.city"
660       variant="outlined">
661     </v-text-field>
662   </v-col>
663   <v-col cols="12" md="8">
664     <v-text-field
665       label="Postcode"
666       v-model="addressForm.postcode"
667       :rules="addressRules.postcode"
668       variant="outlined">
669     </v-text-field>
670   </v-col>
671 </v-row>
672 <v-row>
673   <v-col cols="12" md="4">
674     <v-select
675       label="State/Province"
676       v-model="addressForm.state"
677       :items="getStatesForCountry(addressForm.country)">
678     </v-select>
679   </v-col>
680   <v-col cols="12" md="8">
681     <v-select
682       label="Country"
683       v-model="addressForm.country"
684       :items="countries"
685       :rules="addressRules.country"
686       variant="outlined"
687       @update:modelValue="onCountryChange"
688       :menu-props="{ maxHeight: 300 }">
689     </v-select>
690   </v-col>
691 </v-row>
692
693
694
695

```

```

696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756

```

```

        <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"
                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>

```

```

758      <!-- Confirmation Modal -->
759      <div class="modal fade" id="confirmationModal" tabindex="-1"
760          aria-labelledby="confirmationModalLabel" aria-hidden="true">
761          <div class="modal-dialog modal-dialog-centered">
762              <div class="modal-content custom-card">
763                  <div class="modal-header">
764                      <h5 class="modal-title section-title"
765                          id="confirmationModalLabel">{{ confirmTitle }}</h5>
766                      <button type="button" class="btn-close" data-bs-dismiss="modal"
767                          aria-label="Close" @click="closeConfirm"></button>
768                  </div>
769                  <div class="modal-body text-muted">
770                      {{ confirmMessage }}
771                  </div>
772                  <div class="modal-footer">
773                      <button type="button" class="btn profile-btn cancel-btn"
774                          data-bs-dismiss="modal" @click="closeConfirm">
775                          {{ confirmTitle === 'Success!' || confirmTitle === 'Error'
776                          ? 'OK' : 'Cancel' }}</button>
777                      <button
778                          v-if="confirmTitle !== 'Success!' & confirmTitle !==
779                          'Error'"
780                          type="button"
781                          class="btn profile-btn success-btn"
782                          @click="executeConfirmAction">
783                          Confirm
784                      </button>
785                  </div>
786              </div>
787          </div>
788      </v-app>
789
790  }).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.

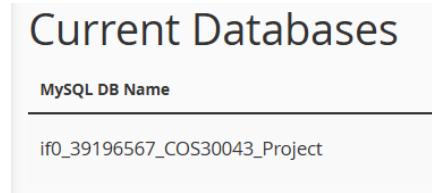
3. User can update name/email/password → submitted to action=update_user.
4. User can manage addresses (add/edit/delete) → all use POST requests with specific action flags.
5. Backend validates, updates DB, and returns JSON responses.
6. Vue displays toast/snackbar messages or validation hints accordingly.

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

```
SQLs > users.sql
1 -- Create the database
2 CREATE DATABASE IF NOT EXISTS if0_39196567_COS30043_Project;
3 USE if0_39196567_COS30043_Project;
4
5 -- Drop and recreate users table
6 DROP TABLE IF EXISTS users;
7
8 CREATE TABLE users (
9     id INT AUTO_INCREMENT PRIMARY KEY,
10    name VARCHAR(100) NOT NULL,
11    email VARCHAR(100) NOT NULL UNIQUE,
12    password VARCHAR(255) NOT NULL,
13    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
14 );
15
```

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.

MySQL DB Name	MySQL User Name	MySQL Password	MySQL Host Name
if0_39196567_COS30043_Project	if0_39196567	(Your vPanel Password)	sql204.infinityfree.com

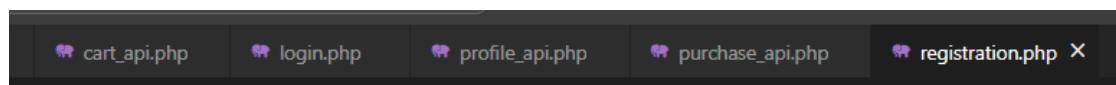
The password for the database can be obtained from the domain homepage. This password was automatically generated when the domain is created.

The screenshot shows the vPanel interface. On the left, under 'Domains', there is a list with one item: 'cozyloop.infinityfreeapp.com'. To the right of this list is a purple button labeled '+ Add Domain'. Below the list is a table with columns 'DOMAIN' and 'ACTIONS', containing the entry 'cozyloop.infinityfreeapp.com' with a 'Manage' button next to it. On the right side of the interface, under 'Account Details', there are fields for 'USERNAME' (set to 'if0_39196567') and 'PASSWORD' (set to 'G19kSBoo5L'). There are also icons for copy and paste.

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
13 // Enable CORS if needed
14 header('Access-Control-Allow-Origin: *');
15 header('Access-Control-Allow-Methods: GET, POST, PATCH, DELETE, OPTIONS');
16 header('Access-Control-Allow-Headers: Content-Type');
17
18 // Handle preflight requests
19 if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
20     http_response_code(200);
21     exit;
22 }
23
24 // Check if user is logged in
25 if (!isset($_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 manage your cart'
30     ]);
31     exit;
32 }
33
34 $user_id = $_SESSION['user_id'];
35
36 // Connect to MySQL
37 $conn = mysqli_connect('sql204.infinityfree.com', 'if0_39196567', 'G19kSBoo5L', 'if0_39196567_C0530043_Project');
38 if (!$conn) {
39     http_response_code(500);
40 }
```

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.

The screenshot shows the InfinityFree website homepage. At the top, there is a navigation bar with links for Home, Profile, Accounts, Free SSL Certificates, Website Builder, Domain Checker, Knowledge Base, and Community Forum. A purple 'Go Premium' button is visible, along with a user icon and the email address shotakuki1202@gmail.com. Below the navigation, there is a section titled 'ACCOUNTS' with 'Hosting Accounts'. It displays three existing accounts: 'if0_39186206' (Website for cozyloop.wuaze.com), 'if0_39196567' (Website for cozyloop.infinityfreeapp.com), and 'if0_39211330' (Website for cozyloop1.infinityfreeapp.com). A purple 'Create Account' button is located in the top right corner of this section. The bottom left shows 'Active Accounts: 2 / 3'.

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

The screenshot shows the 'Choose Hosting Plan' page. It features four hosting plan options: INFINITYFREE (\$0 forever), STARTER PREMIUM (\$2.50/month, billed yearly), SUPER PREMIUM (\$3.65/month, billed yearly), and ULTIMATE PREMIUM (\$6.02/month, billed yearly). Each plan has a list of included features. Below each plan is a 'Create Now' or 'Order Now' button. A message at the top states: 'InfinityFree is sponsored by iFastNet, a provider of powerful, low cost web hosting. Are you looking for more features, more server power, personal support and more? Then iFastNet is the service for you!'

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

The screenshot shows the 'Step 2: Choose a Domain Name' form. It asks for the initial domain name and provides dropdown menus for Subdomain and Domain Extension. The Subdomain field contains 'cozyloop' and the Domain Extension field contains 'infinityfreeapp.com'. At the bottom, there are 'Back' and 'Check Availability' buttons.

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

Step 3: Additional Information

Account Label
Website for cozyloop1.infinityfreeapp.com

A short description to help you identify the account.

Account Username
(generated automatically)

Used to login to FTP, MySQL, etc.

Account Password
.....

A unique password, between 8 and 15 characters, letters and numbers only.

Email Consent
(please select)

Is our supplier allowed to contact you about your hosting account?

← Back + Create Account

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

Manage if0_39196567

Account Options

- Home
- Upgrade to Premium
- Statistics
- FTP Details
- MySQL Databases
- Deactivation History
- Account Settings

Control Panel File Manager Website Builder Script Installer

Domains + Add Domain

DOMAIN	ACTIONS
cozyloop.infinityfreeapp.com	→ Manage

Account Details

USERNAME: if0_39196567

PASSWORD:

STATUS: Active

LABEL: Website for cozyloop.infinityfreeapp.com

WEBSITE IP: (unknown)

HOSTING VOLUME: (unknown)

HOME DIRECTORY: (unknown)

CREATION DATE: 2025-06-10

Domains + Add Domain

DOMAIN	ACTIONS
cozyloop.infinityfreeapp.com	→ Manage

Account Details

USERNAME: if0_39196567

PASSWORD: G19kSBoo5L

The screenshot displays the main account management interface for a user named 'if0_39196567'. On the left, there's a sidebar with various account options like Home, Upgrade to Premium, Statistics, and MySQL Databases. The main area has four tabs at the top: Control Panel (green), File Manager (orange), Website Builder (teal), and Script Installer (purple). Below these tabs, there's a 'Domains' section with a table showing one domain entry: 'cozyloop.infinityfreeapp.com' with a 'Manage' button. To the right of the domains is a large 'Account Details' panel containing fields for Username (set to 'if0_39196567'), Password (a masked string), Status (Active), Label (Website for cozyloop.infinityfreeapp.com), and several status indicators for Website IP, Hosting Volume, Home Directory, and Creation Date (2025-06-10). At the bottom, there's another 'Domains' section and an 'Account Details' panel, both showing the same information as the main panel above. The overall layout is clean and organized, typical of a web-based hosting control panel.

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

The screenshot displays two separate sessions of a file manager, likely from a web-based control panel. Both sessions are navigating through the 'htdocs' directory.

Session 1 (Top):

- Name: commons (Folder)
- Name: images (Folder)
- Name: style (Folder)
- Name: cart.js (File)
- Name: cart.php (File)
- Name: cart_api.php (File)
- Name: index.js (File)
- Name: index.php (File)

Session 2 (Bottom):

- Name: login.js (File)
- Name: login.php (File)
- Name: logout.php (File)
- Name: products.js (File)
- Name: products.json (File)
- Name: products.php (File)
- Name: profile.js (File)
- Name: profile.php (File)
- Name: profile_api.php (File)
- Name: purchase_api.php (File)

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

The screenshot shows the VistaPanel control panel interface. At the top, there are four main navigation buttons: 'Control Panel' (green), 'File Manager' (orange), 'Website Builder' (teal), and 'Script Installer' (purple). Below these, the main dashboard is divided into several sections:

- FILES**: Includes 'Online File Manager', 'Directory Privacy', 'FTP Accounts', and 'Free FTP Software'.
- DATABASES**: Includes 'phpMyAdmin', 'MySQL Databases', and 'Remote MySQL'.
- DOMAINS**: Includes 'Addon Domains', 'Sub Domains', 'Aliases (Parked Domains)', and 'Redirects'.
- EMAIL**: Includes 'MX Entry' and 'SPF Records'.

On the right side, there is a sidebar titled 'ACCOUNT DETAILS' containing account-specific information:

- Main Domain: if0_39196567.infinityfree.com
- FTP hostname: ftpupload.net
- FTP username: if0_39196567
- MySQL hostname: sql204.infinityfree.com
- MySQL username: if0_39196567
- Hosting Volume: vol13_3

Below the account details, it shows the 'Home Directory' as /home/vol13_3/public_html/if0_39196567 and a link to 'More account settings'.

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

The screenshot shows the 'MySQL Databases' page within the VistaPanel control panel. The title bar says 'MySQL Databases'. The page contains the following sections:

- Create New Database**: A yellow banner indicates 'Currently using 0 of 400 available databases.' Below this, a form field shows 'if0_39196567_ COS30043 Project' and a blue 'Create Database' button.
- Current Databases**: A table header with columns: MySQL DB Name, MySQL User Name, MySQL Password, MySQL Host Name, and PHPMyAdmin.
- Footer**: Text about experiencing issues with phpMyAdmin, a note about premium hosting allowing individual MySQL users and privileges, and a 'Click here' link.

New Database has been successfully created.

The screenshot shows the VistaPanel MySQL management interface. At the top, it says "Create New Database" and "Currently using 1 of 400 available databases." A "New Database:" input field contains "if0_39196567_". Below it is a "Create Database" button. Under "Delete a database", there is a dropdown menu set to "if0_39196567_COS30043_Project" and a "Remove Database" button. The "Current Databases" section lists one database: "if0_39196567_COS30043_Project" with MySQL User Name "if0_39196567", MySQL Password "(Your vPanel Password)", MySQL Host Name "sql204.infinityfree.com", and a blue "Admin" button. A note at the bottom says: "Experiencing issues with phpMyAdmin? Log out of your control panel, clear cookies and sessions [not available in all browsers] and log back into your control panel and try again.."

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

The screenshot shows the phpMyAdmin interface for the database "if0_39196567_COS30043_Project". The "Structure" tab is selected. It displays five tables: "addresses", "cart", "purchases", "purchase_items", and "users". Each table has columns for "Action", "Rows", "Type", and "Collation". All tables are defined as MyISAM with latin1_swedish_ci collation. Below the table list, there is a "Create table" form with fields for "Name:" and "Number of columns: 4".

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.

The screenshot shows the "Domains" section of the website's control panel. It lists a single domain: "cozyloop.infinityfreeapp.com" with an "Manage" button. To the right, under "Account Details", are fields for "USERNAME" (set to "if0_39196567"), "PASSWORD" (represented by a series of asterisks), and "STATUS" (set to "Active").

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

```

83 } elseif ($method === 'POST') {
84     // Handle different POST actions: add item, clear cart, remove all of a specific item, add one item, remove one
85     // item
86
87     // Determine the action. If 'action' is not set, assume it's 'add_item' for backward compatibility.
88     $action = $input['action'] ?? 'add_item';
89
90     // Case 1: Add item to cart (initial add from product page or explicit 'add_item' action)
91     if ($action === 'add_item' && isset($input['id'], $input['name'], $input['price'], $input['image'])) {
92         // Validate input
93         if (!is_numeric($input['price']) || $input['price'] < 0) {
94             http_response_code(400);
95             echo json_encode(['error' => 'Invalid price']);
96             exit;
97         }
98
99         // Case 2: Clear entire cart
100        elseif ($action === 'clear_cart') {
101            $stmt = mysqli_prepare($conn, "DELETE FROM cart WHERE user_id = ?");
102            if (!$stmt) {
103                throw new Exception('Failed to prepare statement: ' . mysqli_error($conn));
104            }
105
106         // Case 3: Remove all instances of a specific product
107         elseif ($action === 'remove_all_items' && isset($input['id'])) {
108             $productId = intval($input['id']); // This ID is the product_id
109
110             if (!$productId) {
111                 http_response_code(400);
112                 echo json_encode(['error' => 'Product ID is required for remove_all_items action']);
113                 exit;
114             }
115
116         // Case 4: Add one instance of a product
117         elseif ($action === 'add_one_item' && isset($input['id'], $input['name'], $input['price'], $input['image'])) {
118             $productId = $input['id'];
119             // Validate input
120             if (!is_numeric($input['price']) || $input['price'] < 0) {
121                 http_response_code(400);
122                 echo json_encode(['error' => 'Invalid price']);
123                 exit;
124             }
125
126         // Case 5: Remove one instance of a product
127         elseif ($action === 'remove_one_item' && isset($input['id'])) {
128             $productId = $input['id'];
129
130             $stmt = mysqli_prepare($conn, "DELETE FROM cart WHERE user_id = ? AND product_id = ? LIMIT 1");
131             if (!$stmt) {
132                 throw new Exception('Failed to prepare statement: ' . mysqli_error($conn));
133             }
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206

```

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

```

228 } elseif ($method === 'DELETE') {
229     // All specific DELETE actions (clear_all, remove_all_items)
230     // have been moved to the POST method with an 'action' parameter.
231     // Therefore, any direct DELETE request at this point is unexpected.
232     http_response_code(405); // Method Not Allowed for specific DELETE operations now.
233     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)

```

235 } elseif ($method === 'PATCH') {
236     // All PATCH actions are now deprecated in favor of POST with 'action' parameter.
237     // The original 'remove_one' is now 'remove_one_item' in POST.
238     // The original 'add_one' is now 'add_one_item' in POST.
239     $input = json_decode(file_get_contents('php://input'), true); // Re-decode for safety
240     if (isset($input['action'])) {
241         http_response_code(405);
242         echo json_encode(['error' => "PATCH method for action '$input['action']' is no longer supported. Please
use POST instead."]);
243     } else {
244         http_response_code(405);
245         echo json_encode(['error' => 'PATCH method is not supported for this endpoint. Please use POST with an
action parameter.']);
246     }

```

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                     })
92                 }
93             )
94         }
95
96         clearCart() {
97             this.showConfirm(
98                 'Clear Cart',
99                 'Are you sure you want to clear the entire cart?',
100                () => {
101                    fetch('cart_api.php', {
102                        method: 'DELETE',
103                        headers: { 'Content-Type': 'application/json' },
104                        body: JSON.stringify({ clear_all: true })
105                    })
106                }
107            )
108        }
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
```

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             })
112
113         }
114
115         addOne(item) {
116             fetch('cart_api.php', {
117                 method: 'PATCH',
118                 headers: { 'Content-Type': 'application/json' },
119                 body: JSON.stringify({
120                     id: item.id,
121                     action: 'add_one',
122                     name: item.name,
123                     price: item.price,
124                     image: item.image
125                 })
126             })
127
128
129
130
131
132
133
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                 })
93             }
94         )
95     }
96
97     clearCart() {
98         this.showConfirm(
99             'Clear Cart',
100            'Are you sure you want to clear the entire cart?',
101            () => {
102                // Changed method to POST and adjusted body to send 'action: clear_cart'
103                fetch('cart_api.php', {
104                    method: 'POST', // Changed to POST
105                    headers: { 'Content-Type': 'application/json' },
106                    body: JSON.stringify({ action: 'clear_cart' }) // Changed payload
107                })
108            }
109        )
110    }
111 }
```

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         })
121
122     addOne(item) {
123         // Changed method to POST and adjusted body to send 'action: add_one_item'
124         fetch('cart_api.php', {
125             method: 'POST', // Changed to POST
126             headers: { 'Content-Type': 'application/json' },
127             body: JSON.stringify({
128                 action: 'add_one_item', // New action name
129                 id: item.id, // item.id here refers to product_id
130                 name: item.name,
131                 price: item.price,
132                 image: item.image
133             })
134         })
135     }
136 }
```

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:

```
195      .then((data) => {
196        if (data.success) {
197          this.showInfoMessage('Success', data.message || 'Cart cleared successfully!');
198          this.reloadCart();
199        } else {
200          this.showInfoMessage('Error', data.error || 'Failed to clear cart. Please try again.');
201        }
202      })
203    )
204  )
205}
```

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

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

// Helper method to remove selected items

```
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      },
```

Production version simplified this:

// Simplified - backend handles cart clearing after successful purchase

```
239      const response = await fetch('purchase_api.php', {
240        method: 'POST',
241        headers: {
242          'Content-Type': 'application/json'
243        },
244        body: JSON.stringify({
245          action: 'create_from_cart',
246          selected_items: selectedItems // Send only selected items
247        })
248      });
249
```

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

```
52     try {
53         if ($method === 'GET') {
54             // Get all purchases for the current user with their items
55             $purchasesQuery = "
56                 SELECT p.*,
57                     COUNT(pi.id) as item_count,
58                     SUM(pi.quantity) as total_items
59             FROM purchases p
60             LEFT JOIN purchase_items pi ON p.id = pi.purchase_id
61             WHERE p.user_id = ?
62             GROUP BY p.id
63             ORDER BY p.order_date DESC
64         ";
65     }
```

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

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

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

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

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

```
432 } elseif ($method === 'DELETE') {
433     // Delete purchase items or entire purchase (only user's own)
434     $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

```
119 } elseif ($method === 'POST') {
120     // Handle different POST actions: create purchase, remove item, cancel order, update items
121
122     if (!isset($input['action'])) {
123         http_response_code(400);
124         echo json_encode(['error' => 'Missing action parameter for POST request']);
125         exit;
126     }
127
128     $action = $input['action'];
129
130     if ($action === 'create_from_cart') {
131         // Create new purchase from selected cart items for current user
132         mysqli_begin_transaction($conn);
133
134         // NEW CASE: Remove specific item from order (moved from DELETE method)
135         elseif ($action === 'remove_item_from_order') {
136             if (!isset($input['item_id']) || !isset($input['purchase_id'])) {
137                 http_response_code(400);
138                 echo json_encode(['error' => 'Missing item_id or purchase_id']);
139                 exit;
140             }
141         }
142     }
143 }
```

```

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     }
372 }
373 // NEW CASE: Update items in order (moved from PATCH method)
374 elseif ($action === 'update_items_in_order') {
375     if (!isset($input['purchase_id'], $input['items'])) {
376         http_response_code(400);
377         echo json_encode(['error' => 'Missing purchase_id or items']);
378         exit;
379     }
380 }

```

- **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

```

204 // Remove only the selected items from the cart
205 foreach ($selectedItems as $item) {
206     $productId = $item['id']; // This is the product_id from the grouped item
207     $quantityToRemove = $item['quantity'];
208
209     // Fetch specific cart entry IDs to delete
210     $selectCartItemsStmt = mysqli_prepare($conn, "SELECT id FROM cart WHERE user_id = ? AND product_id = ? LIMIT ?");
211     if (!$selectCartItemsStmt) {
212         throw new Exception('Failed to prepare select cart items statement: ' . mysqli_error($conn));
213     }
214     mysqli_stmt_bind_param($selectCartItemsStmt, "ii", $user_id, $productId, $quantityToRemove);
215     mysqli_stmt_execute($selectCartItemsStmt);
216     $cartItemIdsResult = mysqli_stmt_get_result($selectCartItemsStmt);
217
218     $idsToDelete = [];
219     while ($row = mysqli_fetch_assoc($cartItemIdsResult)) {
220         $idsToDelete[] = $row['id'];
221     }
222     mysqli_stmt_close($selectCartItemsStmt);
223
224     if (!empty($idsToDelete)) {
225         $idList = implode(',', $idsToDelete);
226         // Prepare and execute statement to delete specific cart items by ID
227         $deleteSpecificCartItemsStmt = mysqli_prepare($conn, "DELETE FROM cart WHERE id IN ($idList)");
228         if (!$deleteSpecificCartItemsStmt) {
229             throw new Exception('Failed to prepare delete specific cart items statement: ' . mysqli_error($conn));
230         }
231         if (!mysqli_stmt_execute($deleteSpecificCartItemsStmt)) {
232             throw new Exception('Failed to clear specific cart items: ' . mysqli_stmt_error($deleteSpecificCartItemsStmt));
233         }
234         mysqli_stmt_close($deleteSpecificCartItemsStmt);
235     }
236 }

```

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
207                 // it's empty.
208                 this.loadPurchases().then(() => {
209                     const updatedPurchase = this.purchases.find(p => p.id ===
210                         purchaseId);
211                     // If the purchase no longer has items (and implicitly
212                     // cancelled by backend)
213                     if (updatedPurchase && updatedPurchase.items.length === 0)
214                     {
215                         this.editingOrder = null; // Exit edit mode
216                     }
217                 });
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
166             })
167         });
168     },
169 }
```

```

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  },
184
185  cancelOrder(orderId) {
186    this.showConfirm(
187      'Cancel Order',
188      'Are you sure you want to cancel this order?',
189      () => {
190        fetch('purchase_api.php', {
191          method: 'PATCH',
192          headers: { 'Content-Type': 'application/json' },
193          body: JSON.stringify({
194            action: 'cancel_order',
195            purchase_id: orderId
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          this.loadPurchases(); // Reload purchases to reflect status change
206          this.showInfoMessage('Success', 'Order cancelled successfully.');
207        }); // Using custom message
208      }
209      .catch(err => {
210        console.error('Failed to cancel order:', err);
211        this.showInfoMessage('Error', 'Failed to cancel order. Please try again.');
212      });
213    }
214  );
215}

```

Production (InfinityFree):

DELETE Methods using POST instead:

```
190      removeItemFromOrder(purchaseId, itemId) {
191        this.showConfirm(
192          'Remove Item',
193          'Are you sure you want to remove this item from the order?',
194          () => {
195            // Changed method to POST and adjusted body
196            fetch('purchase_api.php', {
197              method: 'POST', // Changed to POST
198              headers: { 'Content-Type': 'application/json' },
199              body: JSON.stringify({
200                action: 'remove_item_from_order', // New action name
201                item_id: itemId,
202                purchase_id: purchaseId // Added purchase_id for backend context
203              })
204            })
205            .then(res => {
206              if (!res.ok) {
207                throw new Error(`HTTP error! status: ${res.status}`);
208              }
209              return res.json();
210            })
211            .then((data) => {
212              if (data.success) {
213                this.showInfoMessage('Success', data.message || 'Item removed successfully!');
214                // After successful item removal, reload purchases.
215                // Then, check the state of the purchase and exit edit mode if it's empty.
216                this.loadPurchases().then(() => {
217                  const updatedPurchase = this.purchases.find(p => p.id === purchaseId);
218                  // If the purchase no longer has items (and implicitly cancelled by backend)
219                  if (updatedPurchase && updatedPurchase.items.length === 0) {
220                    this.editingOrder = null; // Exit edit mode
221                  }
222                });
223              } else {
224                this.showInfoMessage('Error', data.error || 'Failed to remove item. Please try again.');
225              }
226            })
227            .catch(err => {
228              console.error('Failed to remove item:', err);
229              this.showInfoMessage('Error', 'An unexpected error occurred while removing item. Please try again.'); // Using custom message
230            });
231          }
232        );
233      },
```

PATCH Methods using POST instead:

```
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        // Changed method to POST and adjusted body to send 'action: update_items_in_order'
160        fetch('purchase_api.php', {
161          method: 'POST', // Changed to POST
162          headers: { 'Content-Type': 'application/json' },
163          body: JSON.stringify({
164            action: 'update_items_in_order', // New action name
165            purchase_id: orderId,
166            items: updates
167          })
168        })
169        .then(res => {
170          if (!res.ok) {
171            throw new Error(`HTTP error! status: ${res.status}`);
172          }
173          return res.json();
174        })
175        .then((data) => {
176          if (data.success) {
177            this.editingOrder = null;
178            this.loadPurchases();
179            this.showInfoMessage('Success', data.message || 'Order updated successfully!');
180          }
181        });
182      },
```

```

180         } else {
181             this.showInfoMessage('Error', data.error || 'Failed to update order. Please try again.');
182         }
183     })
184     .catch(err => {
185         console.error('Failed to update order:', err);
186         this.showInfoMessage('Error', 'An unexpected error occurred while updating order. Please try again.');
187     });
188 },

```

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:

```

175         .then((data) => {
176             if (data.success) {
177                 this.editingOrder = null;
178                 this.loadPurchases();
179                 this.showInfoMessage('Success', data.message || 'Order updated successfully!');
180             } else {
181                 this.showInfoMessage('Error', data.error || 'Failed to update order. Please try again.');
182             }
183         })
184         .catch(err => {
185             console.error('Failed to update order:', err);
186             this.showInfoMessage('Error', 'An unexpected error occurred while updating order. Please try again.');
187         });
188 },

```

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:

```
216
217
218
219
220
221
222
```



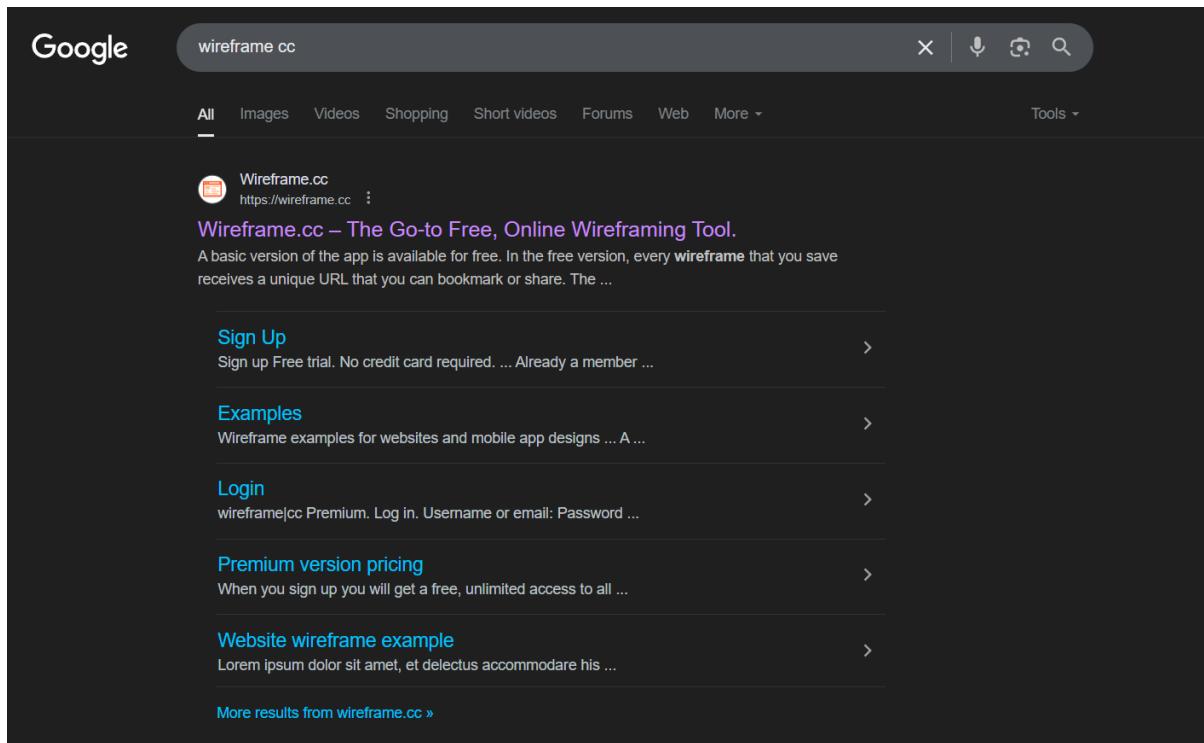
```
this.loadPurchases().then(() => {
  const updatedPurchase = this.purchases.find(p => p.id === purchaseId);
  // If the purchase no longer has items (and implicitly cancelled by backend)
  if (updatedPurchase && updatedPurchase.items.length === 0) {
    this.editingOrder = null; // Exit edit mode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

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



The screenshot shows a Google search results page for the query "wireframe cc". The top result is a link to the Wireframe.cc website, which is described as "The Go-to Free, Online Wireframing Tool". The page includes links for "Sign Up", "Examples", "Login", and "Premium version pricing". Below the main content, there is a link to "More results from wireframe.cc »".

Google search results for "wireframe cc":

- Wireframe.cc – The Go-to Free, Online Wireframing Tool.
A basic version of the app is available for free. In the free version, every **wireframe** that you save receives a unique URL that you can bookmark or share. The ...
- Sign Up >
- Examples >
- Login >
- Premium version pricing >
- Website wireframe example >
- More results from wireframe.cc »

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

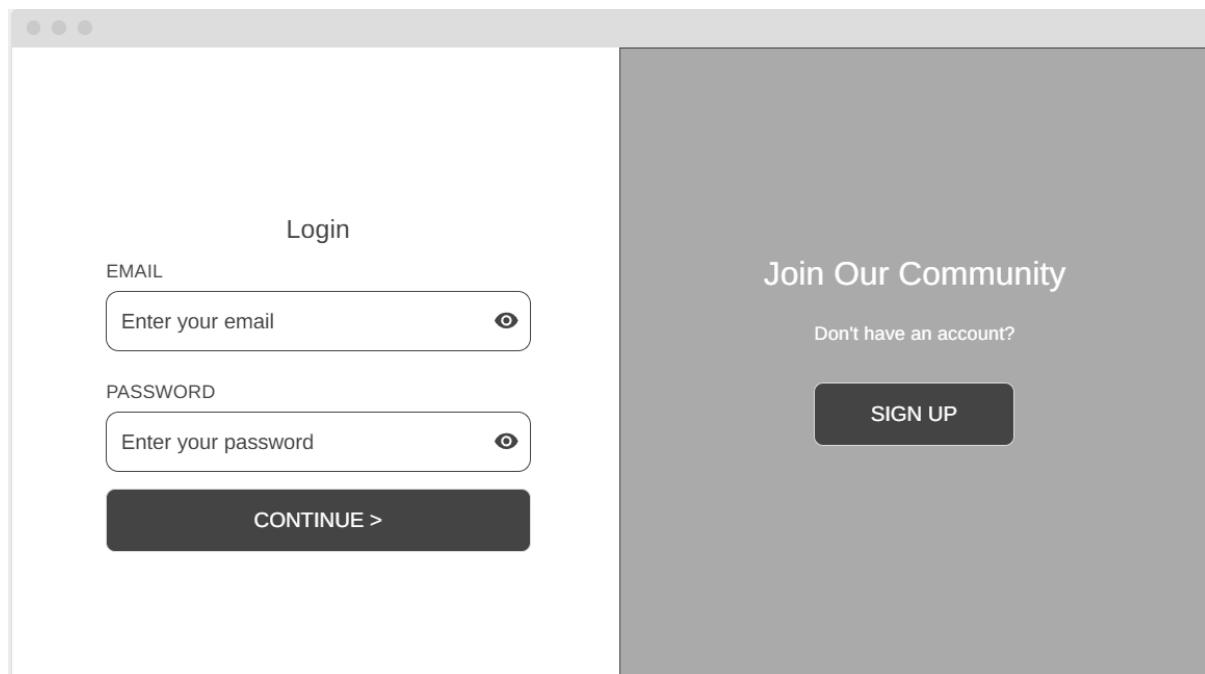
User Experience

- **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

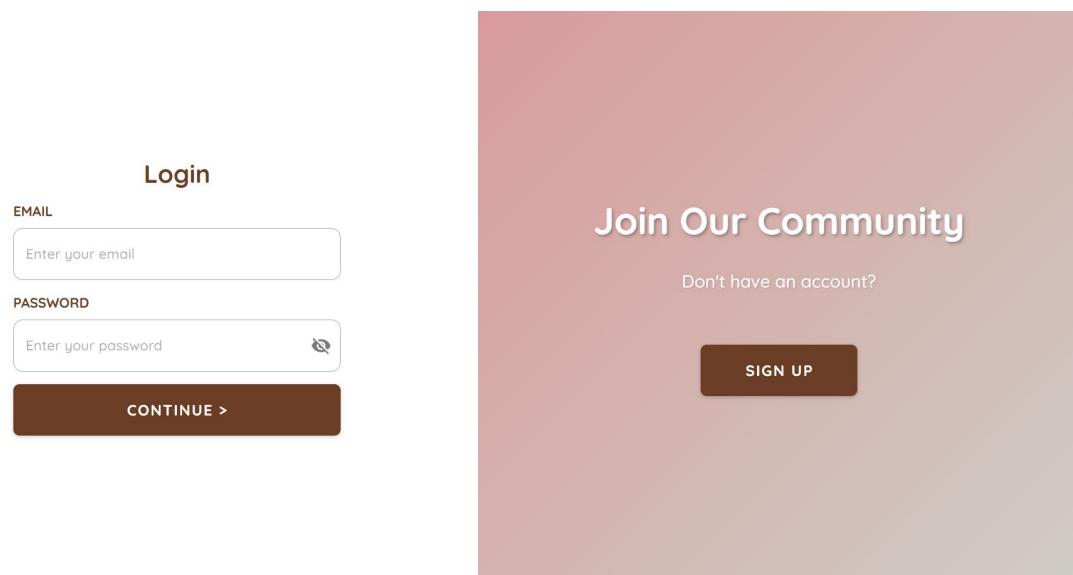
Design Goals

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.

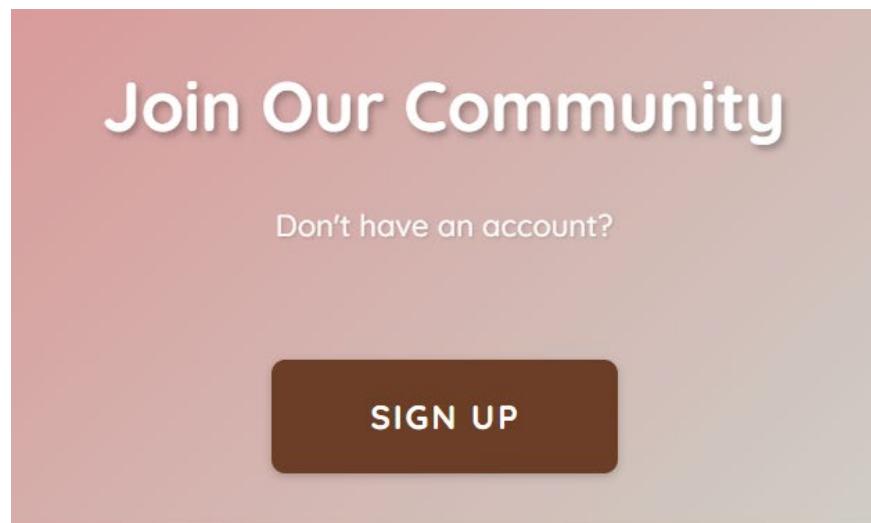
Login Wireframe:



Login Screenshot (Fullscreen):



Login Screenshot (Minimized Screen - Responsive):



Login

EMAIL

Enter your email

PASSWORD

Enter your password

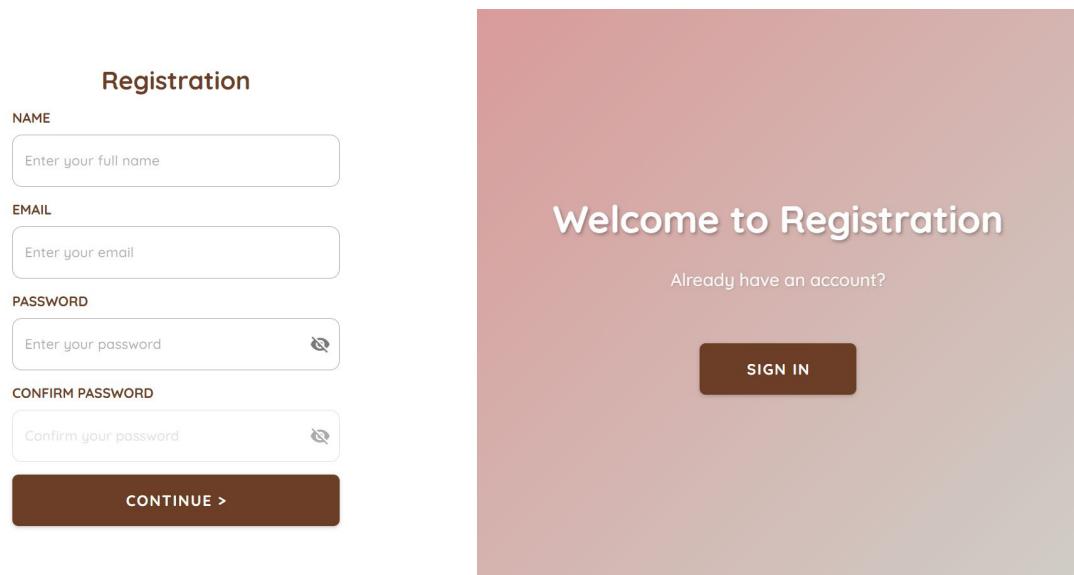


CONTINUE >

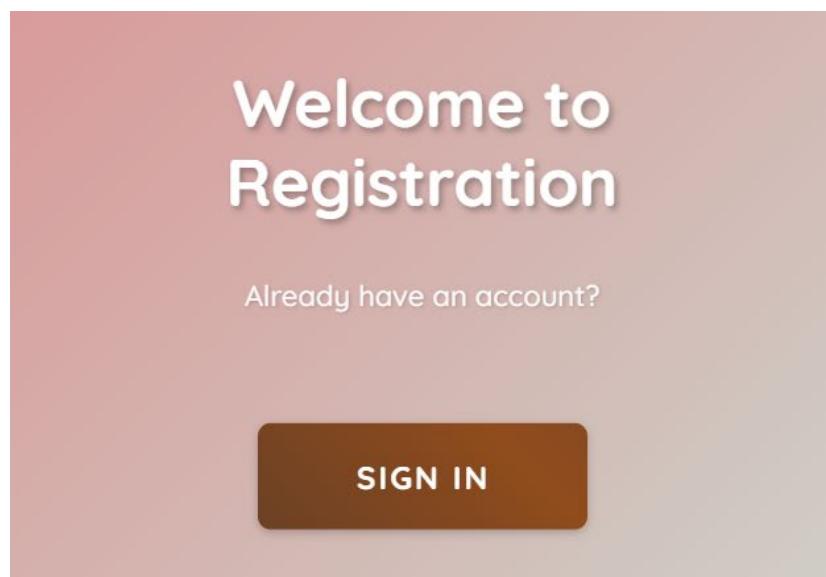
Registration Wireframe:

The wireframe consists of two side-by-side panels. The left panel is titled "Registration" and contains four input fields: "NAME" (placeholder: "Enter your full name"), "EMAIL" (placeholder: "Enter your email"), "PASSWORD" (placeholder: "Enter your password" with an eye icon), and "CONFIRM PASSWORD" (placeholder: "Confirm your password" with an eye icon). Below these fields is a dark grey button labeled "CONTINUE >". The right panel is titled "Welcome to Registration" and features a dark grey "SIGN IN" button. There is also a link "Already have an account?".

Registration Screenshot (Fullscreen):



Registration Screenshot (Minimized Screen - Responsive):



Registration

NAME

Enter your full name

EMAIL

Enter your email

PASSWORD

Enter your password



CONFIRM PASSWORD

Confirm your password



CONTINUE >

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

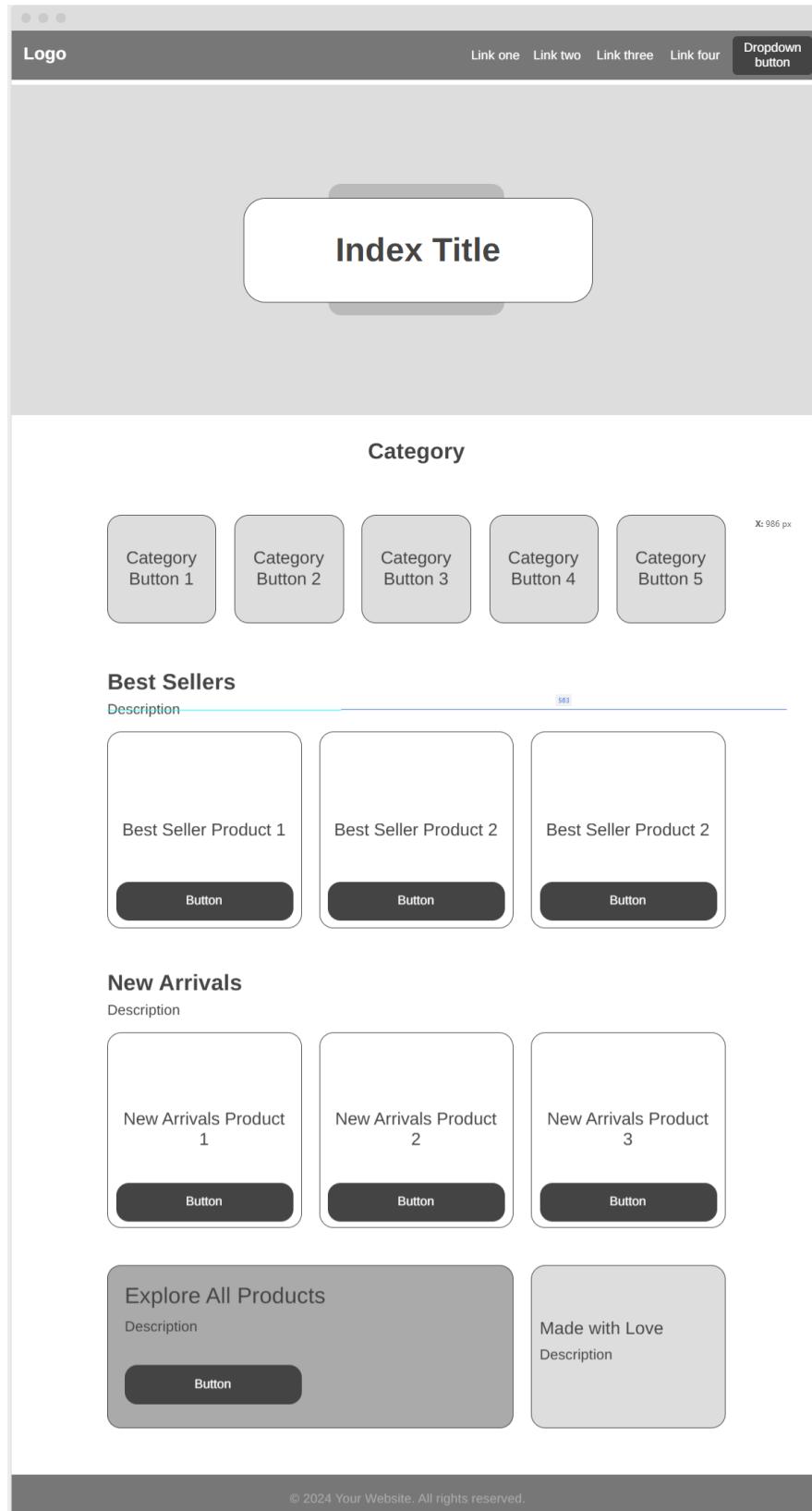
- **Intuitive navigation:** Category-based shopping with clear visual icons (Accessories, Clothing, Home Decor, etc.)
- **Product discovery:** Featured sections highlight popular items and new arrivals with pricing and quick actions
- **Call-to-action optimization:** Green buttons and "View Details" 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):

CozyLoops

Home Products Cart Purchases Bobby ▾

Handmade Crochet with Love

Shop by Category

Explore our handcrafted collections

- Accessories (6 items available)
- Clothing (3 items available)
- Home Decor (2 items available)
- Baby Items (3 items available)
- Seasonal (1 items available)

★ Best Sellers

Our most popular handcrafted items

Crochet Bunny
A cute, handmade crochet bunny made with soft yarn. Perfect for gifts and nurser..

\$25.00 Baby Items

[View Details](#)

Soft Crochet Blanket
A soft and warm crochet blanket, perfect for cozying up on the couch or adding a..

\$45.00 Accessories

[View Details](#)

Winter Cozy Hat
Stay warm and stylish with this cozy crochet hat. Available in various sizes and..

\$18.00 Clothing

[View Details](#)

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](#)

Every item crafted by hand with care and precision

Index Screenshot (Minimized Screen - Responsive):

CozyLoops

Home

Products

Cart

Purchases

Bobby ▾

Handmade Crochet with Love

Shop by Category

Explore our handcrafted collections

- Accessories**
6 items available
- Clothing**
3 items available
- Home Decor**
2 items available
- Baby Items**
3 items available
- Seasonal**
1 items available

★ Best Sellers

Our most popular handcrafted items

Crochet Bunny

A cute, handmade crochet bunny made with soft yarn. Perfect for gifts and nurser...

\$25.00

Baby Items

[View Details](#)

Soft Crochet Blanket

A soft and warm crochet blanket, perfect for cozying up on the couch or adding a...

\$45.00

Accessories

[View Details](#)

Winter Cozy Hat

Stay warm and stylish with this cozy crochet hat. Available in various sizes and...

\$18.00

Clothing

[View Details](#)

New Arrivals

Fresh additions to our collection



New

Crochet Sweater

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

\$35.00

Clothing

[View Details](#)



New

Handmade Crochet Pillow

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

\$30.00

Home Decor

[View Details](#)



New

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

© 2025 CozyLoops. All rights reserved.

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:

Logo

Link one Link two Link three Link four Dropdown button

Product Showcase

Browse by Category

All Products All Products All Products All Products All Products All Products

Search Products Price Range Sort By Search Products View Mode

Product Product Product

Product Product Product

Product Product Product

Prev 1 2 Prev

Jump to page: 1 Go

Products Screenshot (Fullscreen):

CozyLoops

Home Products Cart Purchases Bobby ▾

Product Showcase

Browse by Category

- All Products 15
- Accessories 6
- Clothing 3
- Home Decor 2
- Baby Items 3
- Seasonal 1

Search Products

Price Range Sort By Items Per Page View Mode

15 products found Page 1 of 2 (1-9 of 15)

	Crochet Baby Blanket A soft and warm crochet baby blanket, perfect for newborns. Made with hypoallerg... \$35.00		Crochet Baby Booties Cute and comfy crochet baby booties, perfect for your little one. Soft, stretchy... \$15.00		Crochet Bunny A cute, handmade crochet bunny made with soft yarn. Perfect for gifts and nurser... \$25.00
	Crochet Christmas Stocking Add a handmade touch to your holiday decorations with this crochet Christmas sto... \$12.00		Crochet Coffee Cup Cozy Keep your hands cool with this eco-friendly crochet coffee cup cozy. Fits most s... \$8.00		Crochet Handbag Stylish and eco-friendly crochet handbag, perfect for carrying your essentials w... \$22.00
	Crochet Keychain Handmade crochet keychain. Add a touch of personality to your keys, bags, or a... \$6.00		Crochet Phone Case Protect your phone with this cute crochet phone case. Handmade with care and ava... \$14.00		Crochet Plant Pot Cover A stylish crochet cover for your plant pots. Adds charm and protection to your p... \$18.00
View Details	View Details	View Details	View Details	View Details	View Details

Prev 1 2 Next

Jump to page: 1 GO

© 2025 CozyLoops. All rights reserved.

Products Screenshot (Minimized Screen - Responsive):

CozyLoops

- Home
- Products**
- Cart
- Purchases
- Bobby ▾

Product Showcase

Browse by Category

- All Products 15**
- Accessories 6**
- Clothing 3
- Home Decor 2
- Baby Items 3
- Seasonal 1

Search Products

Price Range

All Prices

Sort By

Name (A-Z)

Items Per Page

9

View Mode

grid icon

list icon

15 products found

Page 1 of 2 (1-9 of 15)

Crochet Baby Blanket

A soft and warm crochet baby blanket, perfect for newborns. Made with hypoallerg...

\$35.00

Baby Items

View Details

Crochet Baby Booties

Cute and comfy crochet baby booties, perfect for your little one. Soft, stretchy...

\$15.00

Baby Items

View Details

Crochet Bunny

A cute, handmade crochet bunny made with soft yarn. Perfect for gifts and nurser...

\$25.00

Baby Items

View Details

Crochet Christmas Stocking

Add a handmade touch to your holiday decorations with this crochet Christma...

\$12.00

Seasonal

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](#)



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: [GO](#)

© 2025 CozyLoops. All rights reserved.

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

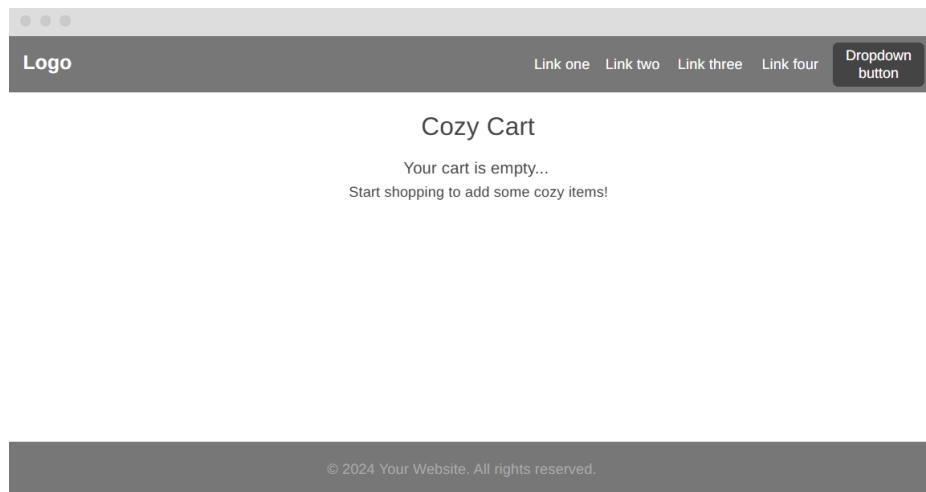
- **Emotional engagement:** Empty cart messaging maintains brand personality while encouraging action
- **Bulk operations:** "Select all items" functionality with clear cart management options
- **Purchase motivation:** Free shipping threshold prominently displayed to increase average order value
- **Conversion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

Design Goals

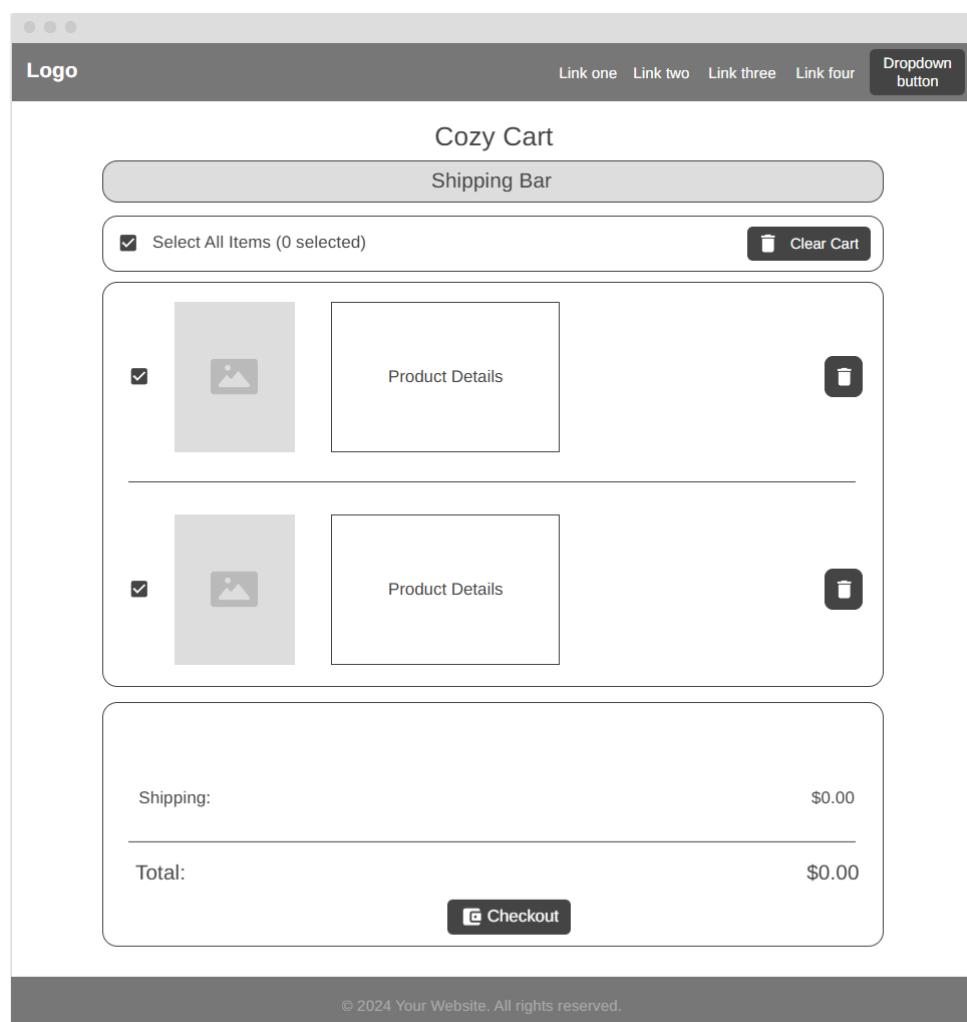
The design creates a seamless transition from browsing to purchasing while maintaining the warm, community-focused brand identity. The cart experience reduces abandonment through clear pricing, shipping incentives, and intuitive management tools, while the playful "cozy"

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):

The screenshot shows the CozyLoops website with a pink header bar. The header includes the brand name "CozyLoops" on the left and navigation links "Home", "Products", "Cart" (which is highlighted in red), "Purchases", and "Bobby" on the right. Below the header, the main content area has a light pink background. At the top of this area, there's a section titled "Cozy Cart" with a shopping cart icon. It displays the message "Your cart is empty..." with a small heart icon and encourages users to "Start shopping to add some cozy items!". At the bottom of the main content area, a thin footer bar contains the copyright notice "© 2025 CozyLoops. All rights reserved."

Cart Screenshot (With Products Fullscreen):

The screenshot shows the CozyLoops website with a pink header bar. The header includes the brand name "CozyLoops" on the left and navigation links "Home", "Products", "Cart" (which is highlighted in red), "Purchases", and "Bobby" on the right. Below the header, the main content area has a light pink background. At the top of this area, there's a yellow banner with a delivery truck icon and the text "Add \$10.00 more to qualify for free shipping!". Below the banner, there's a section titled "Cozy Cart" with a shopping cart icon. It shows two items selected: "Crochet Bunny" and "Crochet Baby Booties". Each item has a checkbox checked, a small image, the product name, price (\$25.00 and \$15.00 respectively), quantity (set to 1), and a "Remove" button. Below the item list is an "Order Summary" table:

Order Summary	
Subtotal (2 items):	\$40.00
Shipping:	\$5.00
Total:	\$45.00

At the bottom of the summary table is a dark blue "PROCEED TO CHECKOUT" button. At the very bottom of the page, a thin footer bar contains the copyright notice "© 2025 CozyLoops. All rights reserved."

Cart Screenshot (Minimized Screen - Responsive):

The image displays two screenshots of the CozyLoops mobile application, illustrating the cart screen across different screen sizes.

Top Screenshot (Large Screen):

- Header:** "CozyLoops" logo, navigation menu icon (three horizontal lines), "Home", "Products", **"Cart"** (highlighted in red), "Purchases", and "Bobby" (with a dropdown arrow).
- Title:** "Cozy Cart" with a shopping cart icon.
- Notification Bar:** "Add \$10.00 more to qualify for free shipping!" with a progress bar.
- Cart Item 1:** "Crochet Bunny" (checkbox checked, image of a bunny), Price: \$25.00, Quantity: 1 (selected), Total: \$25.00.
- Cart Item 2:** "Crochet Baby Booties" (checkbox checked, image of baby booties), Price: \$15.00, Quantity: 1 (selected), Total: \$15.00.
- Order Summary:** Subtotal (2 items): \$40.00, Shipping: \$5.00, Total: \$45.00.
- Checkout Button:** "PROCEED TO CHECKOUT" with a shopping cart icon.

Bottom Screenshot (Small Screen):

- Header:** "CozyLoops" logo, navigation menu icon, "Home", "Products", **"Cart"** (highlighted in red), **"Purchases"** (highlighted in red), and "Bobby" (with a dropdown arrow).
- Title:** "Cozy Cart" with a shopping cart icon.
- Message:** "Your cart is empty... 🌸".
- Text:** "Start shopping to add some cozy items!"
- Footer:** "© 2025 CozyLoops. All rights reserved."

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:

The wireframe illustrates a 'Purchases History' section with a header bar containing a logo, four links, and a dropdown button. Below the header is a search bar with fields for 'Search Products', 'Price Range', and 'Sort By'. Two order cards are displayed, each with an 'Order ID', 'Date/TimeStamp', 'Product Details' (with a placeholder image), and a summary table showing Subtotal, Shipping, and Total. The footer contains a copyright notice.

Purchases History

Logo Link one Link two Link three Link four Dropdown button

Search Products Price Range Sort By

Order ID Order Status Edit/Cancel Order

Date/TimeStamp

Product Details

Subtotal:	\$0.00
Shipping:	\$0.00/FREE
Total:	\$0.00

Subtotal: \$0.00
Shipping: \$0.00/FREE
Total: \$0.00

Order ID Order Status Edit/Cancel Order

Date/TimeStamp

Product Details

Subtotal:	\$0.00
Shipping:	\$0.00/FREE
Total:	\$0.00

© 2024 Your Website. All rights reserved.

Purchases Screenshot (Fullscreen):

The screenshot shows the 'Purchase History' section of the CozyLoops website. At the top, there are search and filter options: 'Search Orders' (with a placeholder 'Search by order number or item name...'), 'Filter by Status' (set to 'All Orders'), and 'Sort by' (set to 'Newest First'). Below this, two orders are listed:

Order #ORD-2025-0323
 June 11, 2025 at 08:20 PM
 PENDING

Crochet Baby Booties	Quantity: 5	\$15.00	\$75.00
		Subtotal:	\$75.00
		Shipping:	FREE
		Total:	\$75.00

Order #ORD-2025-2427
 June 11, 2025 at 08:18 PM
 CANCELLED

Crochet Coffee Cup Cozy	Quantity: 1	\$8.00	\$8.00
		Subtotal:	\$8.00
		Shipping:	\$5.00
		Total:	\$13.00

At the bottom, a copyright notice reads: © 2025 CozyLoops. All rights reserved.

Purchases Screenshot (Minimized Screen - Responsive):

The screenshot shows the 'Purchase History' section on a mobile-like screen. The navigation bar includes 'Home', 'Products', 'Cart', 'Purchases' (which is highlighted), and 'Bobby ▾'. The main content area displays the same two orders as the fullscreen version:

Order #ORD-2025-0323
 June 11, 2025 at 08:20 PM
 PENDING

Crochet Baby Booties	Quantity: 5	\$15.00	\$75.00
		Subtotal:	\$75.00
		Shipping:	FREE
		Total:	\$75.00

Order #ORD-2025-2427
 June 11, 2025 at 08:18 PM
 CANCELLED

Crochet Coffee Cup Cozy	Quantity: 1	\$8.00	\$8.00
		Subtotal:	\$8.00
		Shipping:	\$5.00
		Total:	\$13.00

At the bottom, a copyright notice reads: © 2025 CozyLoops. All rights reserved.

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

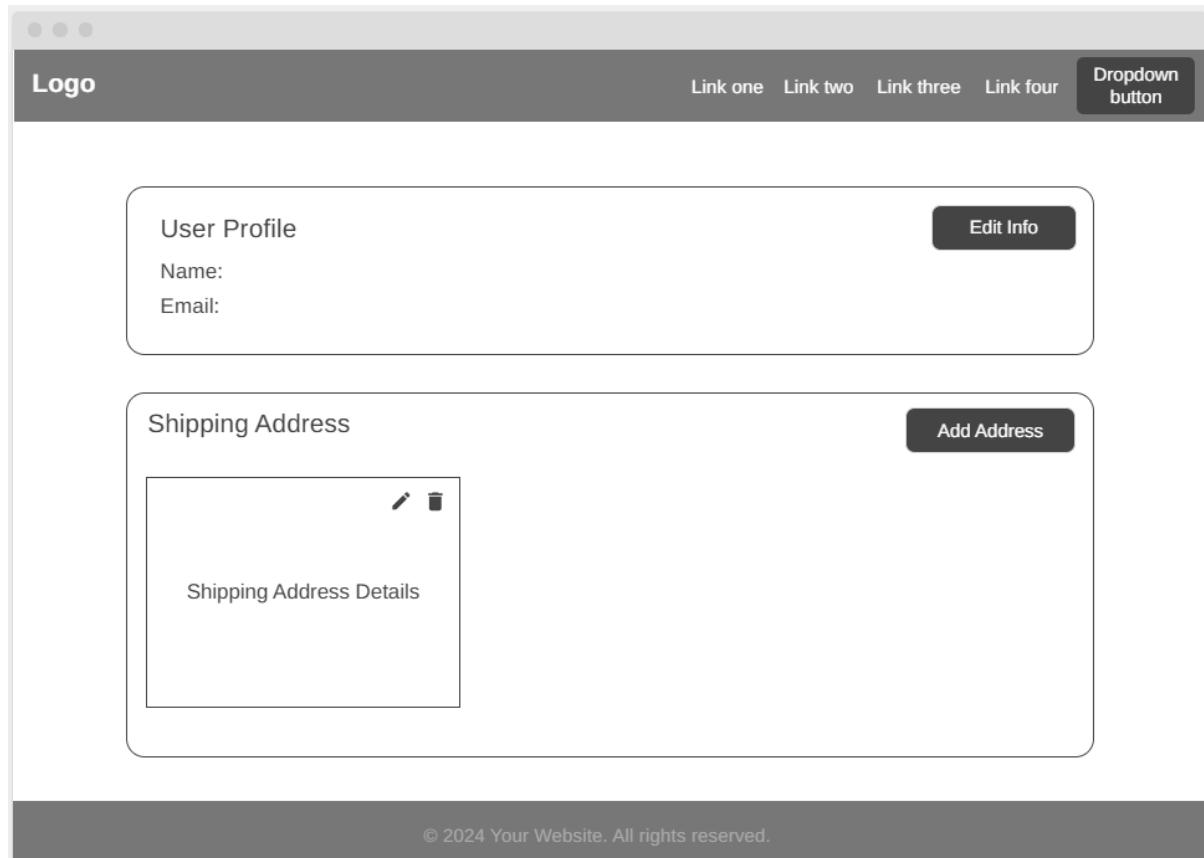
- **Context switching:** Smooth transition between view and edit modes without page refreshes
- **Data organization:** Logical grouping of personal information separate from shipping logistics
- **Security features:** Password confirmation fields and clear save/cancel options for secure updates
- **Address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

The wireframe displays two stacked edit forms on a web page. At the top, a dark header bar contains a 'Logo' icon, four navigation links ('Link one' through 'Link four'), and a 'Dropdown button'. The first form, titled 'User Profile', is for 'Edit Profile Information' and includes fields for 'Name', 'Email', 'New Password', and 'Confirm New Password', along with 'Save Changes' and 'Cancel' buttons. The second form, titled 'Shipping Address', is for 'Add New Address/Edit Address' and includes fields for 'Full Name' (with a '+60' country code button), 'Unit Number (Optional)', 'Street Address', 'State/Province', 'Country', and 'Save/Update Changes' buttons. A copyright notice at the bottom reads '© 2024 Your Website. All rights reserved.'

User Profile
Edit Profile Information

Name

Email

New Password

Confirm New Password

Save Changes Cancel

Shipping Address
Add New Address/Edit Address

Full Name +60 Phone no.

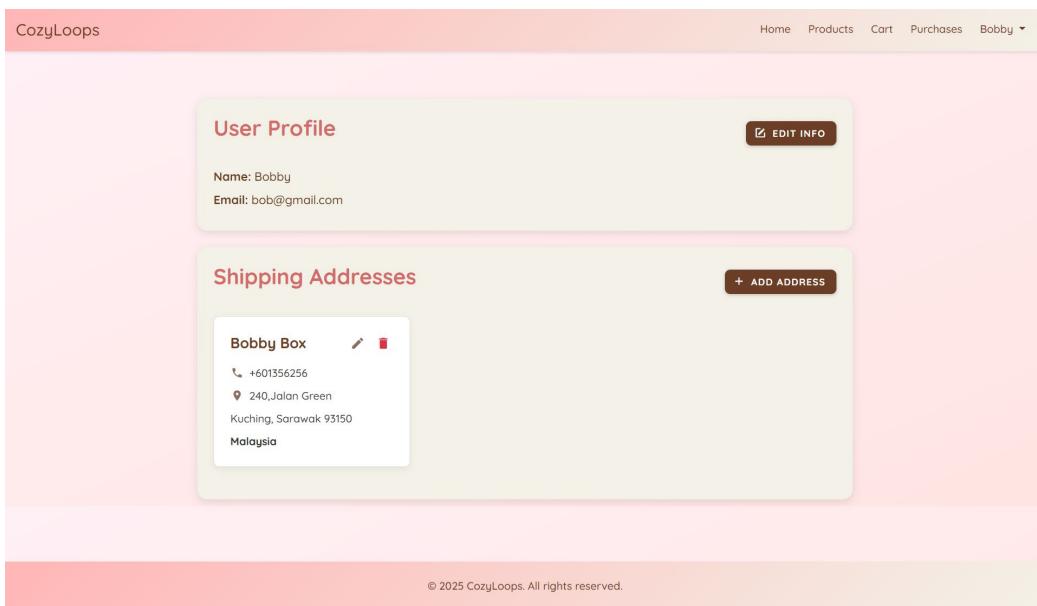
Unit Number (Optional) Street Address

State/Province Country

Save/Update Changes Cancel

© 2024 Your Website. All rights reserved.

Profile Screenshot (Main View Fullscreen):



Profile Screenshot (Main View Fullscreen):

