IE4727 Web Application Design - Project Report

Design project group number: *F31-DG06*

Team members: *Tan Yin Yu, Liw Jun Le*

Project Title: *Lecker Haus – German Food Delivery*

Summary of Project:

Lecker Haus *is a web portal for ordering authentic German cuisine online. The application allows customers to browse for German dishes organised into Starters, Mains, Sides and Desserts.*

Customers can create accounts, add items to their shopping cart, and place orders by providing delivery information, including name, address, and phone number. The system generates order confirmations and order tracking through five status stages: Order Placed, Order Accepted, Preparing Order, Out for Delivery, and Order Delivered.

The application features a customer feedback system where users can rate their experience and provide comments linked to specific order numbers. Additionally, Lecker Haus offers catering services for events, allowing customers to plan larger gatherings with customised German cuisine menus.

Architecturally implemented in ***two parallel versions*** *to demonstrate comprehensive full-stack development proficiency:*

1. ***Traditional stack:*** *Server-side rendered multi-page, built with PHP, MYSQL and vanilla JavaScript. Following conventional web development patterns.*

2. ***Modern stack:*** *Single-page application built with React, utilising RESTful APIs (PHP backend), state management and client-side rendering without page refreshes.*

1. **Application Requirements and Specifications**

2. **Functional Requirements and Specifications**

3. **Design of the web application**
   - 3.1 Site Map
   - 3.2 Story Board
   - 3.3 Wireframe
   - 3.4 Database

4. **Implementation**

5. **Testing of web application**

6. **Summary of Modern Enhancements with Justification**

7. **Conclusion**

Appendices:

- Appendix 1: Source Codes
- Appendix 2: Contribution to the project by each team member
- Appendix 3: Contribution to the report by each team member

1. Application Requirements and Specifications

This section outlines the overall expectations for Lecker Haus. The homepage should set the tone with a clear top navigation and welcoming hero. The interface should have a bright accent that suits a food platform hinting at a warm and appetising atmosphere, paired with soft neutrals for readability. The hero should feature a zoomed-in food image, so the first thing visitors see is the cuisine itself. Showing an immediate and appetising cue that anchors LeckerHaus brand and invites users to browse.

R1. Usability:
The website should be easy to understand on the first visit. Key actions like logging in, browsing dishes, adding to cart, checking out and viewing order history must be obvious and take only a few straightforward steps. The navigation bar and buttons use plain language. Call to actions such as "Browse Menu" should be highlighted to stand out and cart is always accessible to users.

R2. Security:
User authentication can be handled with password hashing using PHP's password_hash( ) to store passwords and password_verify( ) for login checks. Sesssions ($_SESSION) can be implemented to track logged-in users and restrict actions such as ordering food without login. Database queries can use PHP Data Objects prepared statements to prevent SQL injection. The output from the server-side generated webpage should be sanitised with perhaps with htmlspecialcharacters( ) to block cross-site scripting. On the client side, React can implement validation checks for input data before sending data to the server.

R3. Scalability:
This project should separates frontend (React) and backend (PHP APIs), allowing independent updates to each layer. The API endpoints should follow RESTful design using HTPP methods and are stateless. The modular folder structure eg. (pages/, components/, services/ and backend/api/) can make adding features strightforward. Currently, if the application runs locally on XAMPP, this architecture supports future cloud deployment where scaling would become relevant.

R4: Maintainability:
Code should be organised into logical folders (pages/, components/, services/ and backend/api/) for clear separation of functionality. The centralised api.js consolidates all backend communication in one location, making API URL changes simple. Naming conventions must also be consistent with camelCase for JavaScript like handleSubmit()) and snake_case for database fields like user_id. CSS stylesheets should also be separated for each page and component, preventing style conflicts and making design updates modular and easy.

R5: Accessibility:

Form inputs should include proper <label> elements with htmlFor attributes linking to input IDs like <label> htmlFor=”username”> throughout login, feedback and checkout forms, helping assistive technologies identify field purposes. Images must also include alt attributes describing content like alt=”Plate of Tomato Pasta”. These indications are to aid those who require further assistance in identifying elements in the webpage.

2. Functional Requirements and Specifications

F1. User Authentication

The system should provide secure user registration and login functionality to track the orders of each user. New users can create accounts by providing their name, email address, and password. Email validation can be performed via PHP's filter_var( ) function while password validation is done via regex pattern (8 char long, at least 1 letter, 1 number and 1 special character). Users who log out should terminate their session and clear authentication cookies.

F2. Menu Browsing and Ordering

The system should enable users to browse the menu organised by categories like Mains, Starters, Sides and Desserts. Each menu item should display a product's image, name, description and price. Users can filter menu items by selecting a specific category. The menu interface should include a search functionality that allows users to search for specific dishes by name. Users should be able to sort menu items by price. The menu should dynamically load product information from the database and after the user clicks on "Add to cart" there should be a confirmation message informing of the event.

F3. Shopping Cart

The system should provide a shopping cart where users can view all menu items they have added for purchase. The cart should display each item's name, image, individual price, selected quantity and subtotal. Users should be able to modify item quantities directly in the cart using increment/decrement buttons. Items can be removed from cart entirely with a delete button. The cart automatically calculates and displays the total price for all items, updating in real time as the user makes changes. Cart contents should persist in the database and linked to the user's session allowing users to leave and return without losing their cart items. Location and phone number details can be filled up before placing order. When the cart is empty, it will be clearly indicated and direct the user to the menu.

F4. Checkout and Order Placement

Before checkout, users must enter delivery information including their phone number and delivery address. There should be a validation logic to ensure complete information is given before order submission. User can see a order summary showing cart items, quantities, individual prices and the total order amount. Upon clicking the "Place Order" button, the system processes the order by creating a new order record in the database with a unique order ID, storing all order items with their details, and recording the customer information and timestamp of order. After successful order placement, the cart should be cleared and users redirected to another page to view order status. If order creation fails, an error message will prompt the user to try again.

F5. Order Status Tracking

The system should allow users to have a page to track the current status of their orders. Users can view their active and completed orders in a chronological list. Each order displays its current status (e.g., "Order Placed", "Preparing", "Out for Delivery" and "Delivered"). Order statuses are stored in the database and should be updated by restaurant staff, with changes immediately reflected. (Button to update statuses are shown as a demo within the webpage. Theoretically, users themselves would not be able to update statuses themselves. This part was done solely to demonstrate examples of status updates.)

F6. Order History Viewing
The system should have an order history page implemented with server-side generated PHP page that displays past orders for the logged-in user. Each order shows the order ID, order date and time, total amount and current status. Orders should be sorted by descending date/time. If a user is not logged in, attempting to access this page automatically redirects them to the login page. This page should dynamically generate HTML content based on database queries.

F7. Customer Feedback Submission
The system should provide a feedback form where users can submit their reviews and comments about LeckerHaus food ordering experience. The feedback form includes fields for the user's name, email address, rating and additional comments. All fields should be validated on the client-side (JavaScript) and server-side (PHP) to ensure complete information is submitted. Upon submission, the feedback is stored in the database with a timestamp and an "approved" status flag defaulting to unapproved which can be toggled to control what feedback appears on the feedback page. Users will also receive a confirmation message thanking them for their feedback.

F8. Catering Package
For orders of a larger scale, the system will have a page to present information about available catering packages for large gatherings. This page will display a table with three package tiers to compare features of each package in a side-by-side format. This tabular format enables users to quickly compare packages and identify which features are included in each tier. Below the table, there should be a contact section with the catering team's email address, directing interested users to reach out for bookings or custom requests. This serves both as an informational resource and marketing tool to expand Lecker Haus' business beyond regular delivery orders.

F9. Email for catering (Localhost)
The system should provide an inquiry form on the catering page where users can submit custom catering requests directly to the restaurant team. The form collects essential event details including customer name, contact information (email and phone number), event date, expected number of guests and additional message or special requests. All form fields should undergo validation both on the client side (JavaScript) and server-side (PHP) to ensure complete and correctly formatted information. Phone number must be exactly 8 digits, email addresses should follow "user@example.com" format.

Upon successful form submission, the system uses PHP's built-in mail() function to send inquiry details to the catering team's email address. The email includes all submitted information formatted clearly for the restaurant staff to review and respond to the inquiry. The email will be interfaced locally via Thunderbird with the Mercury/32 mail server.

3. Design of the web application

Site Map

The LeckerHaus web application follows a Single Page Application architecture. The application is organised around a hierarchical navigation structure with Home as the root page branching into other pages. User authentication creates a protected area containing Cart, Checkout, Order Status and Order History Page.

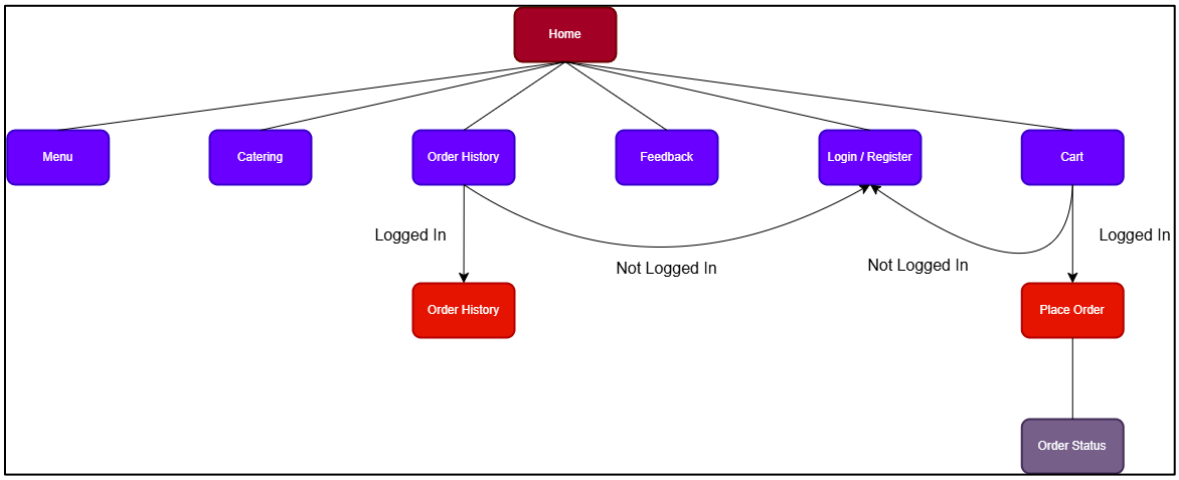


Figure 1. Sitemap

Wireframe

Figure 2. Overview of general wireframing

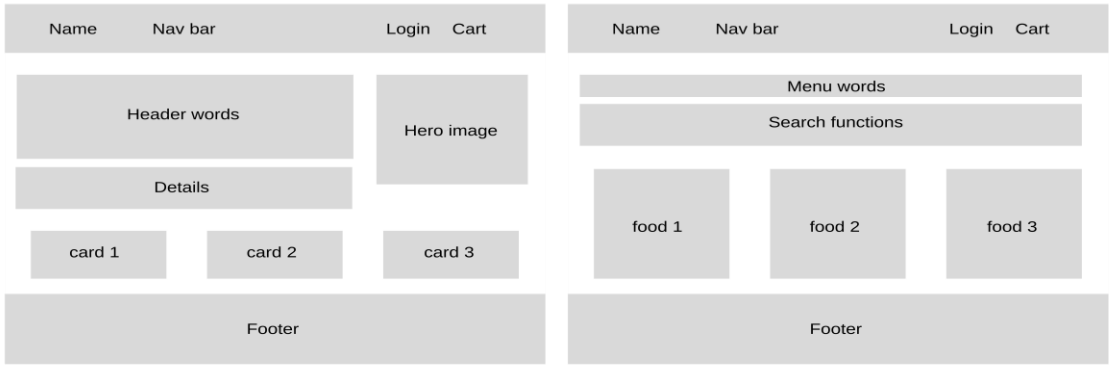


Figure 3. Left: Main page, right: Menu page

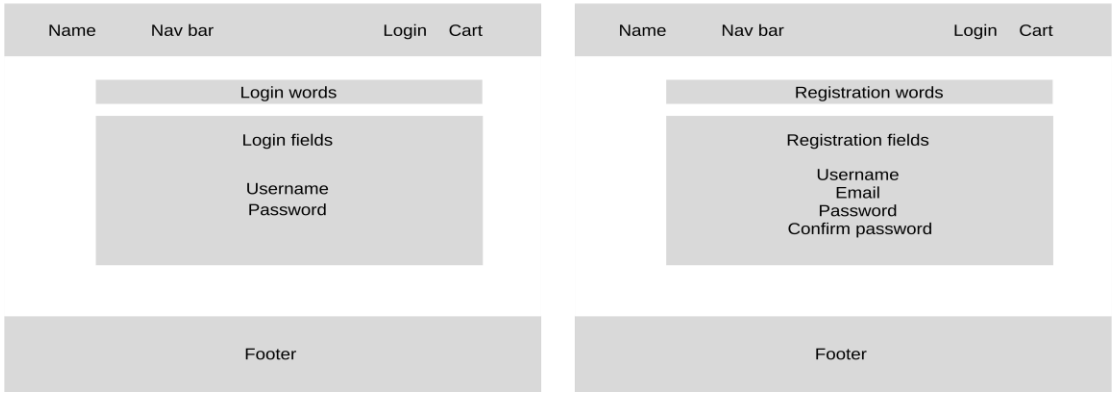


Figure 4. Left: Login page, right: Registration page

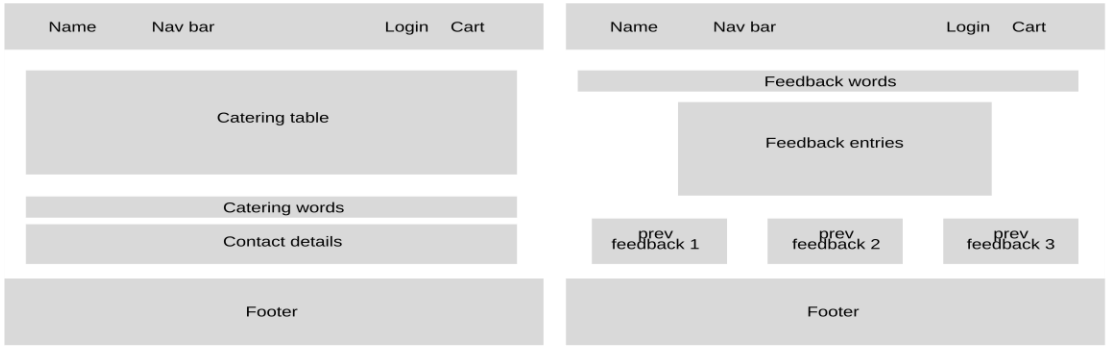


Figure 5. Left: Catering page, right: Feedback page

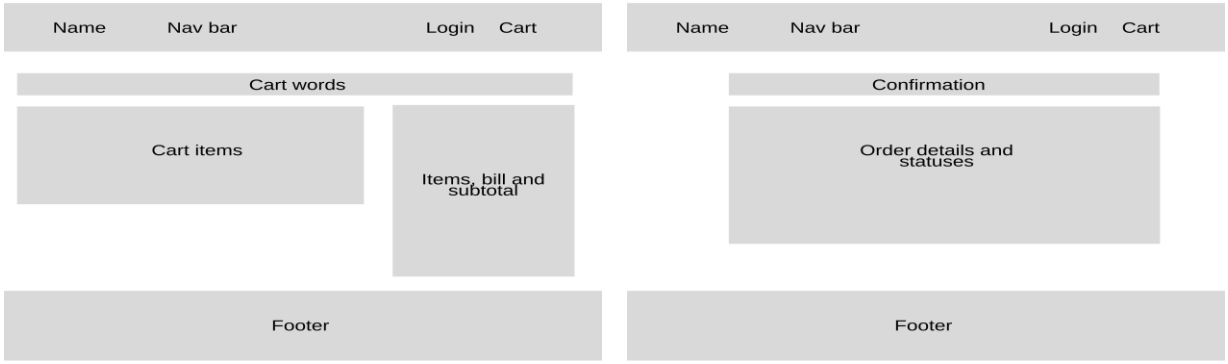


Figure 6. Left: Cart page, right: Confirmation page

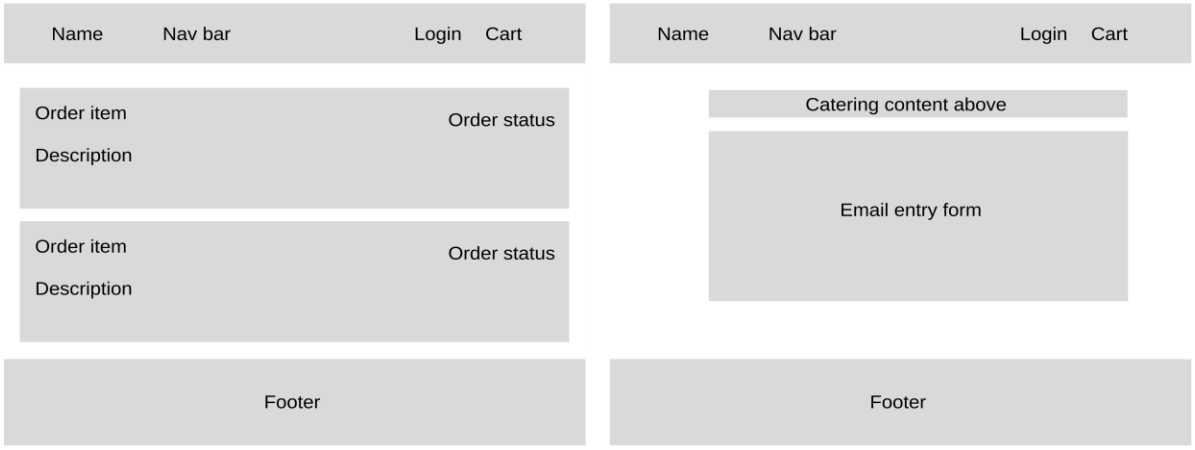


Figure 7. Left: Order history page, right: Email functionality within catering page

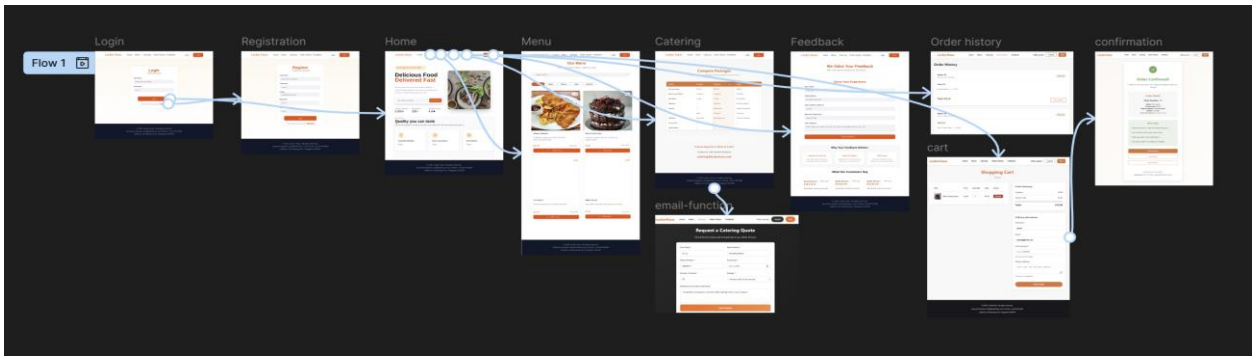


Figure 8. Overview of in-depth wireframing

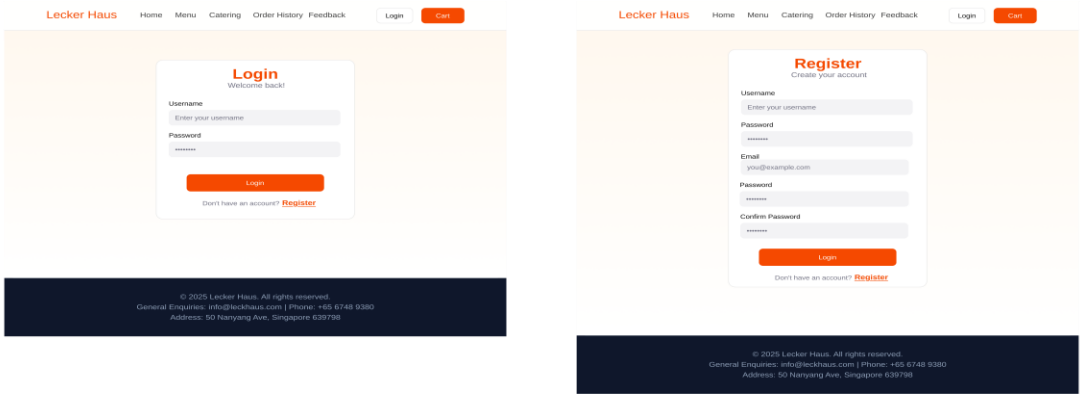


Figure 9. Left: Login page, right: Registration page

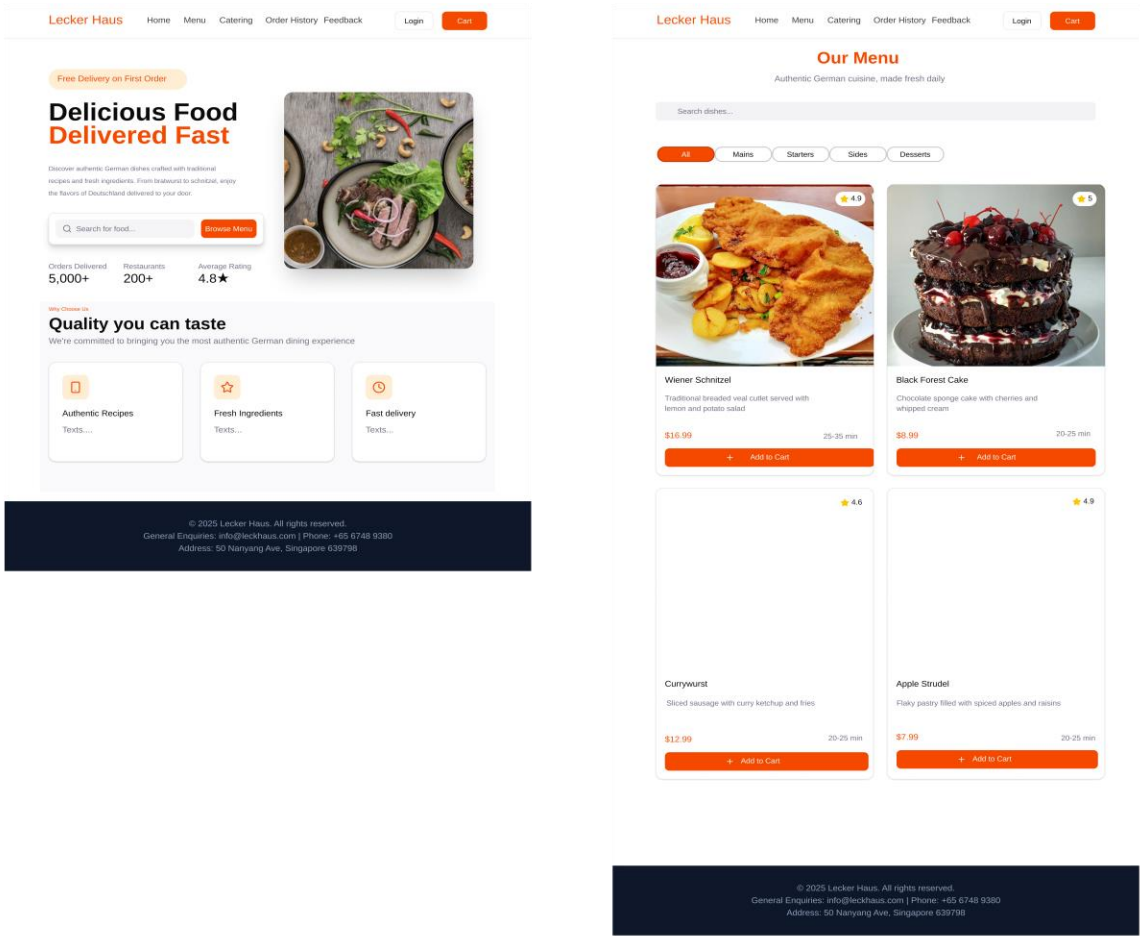


Figure 10. Left: Main page, right: Menu page

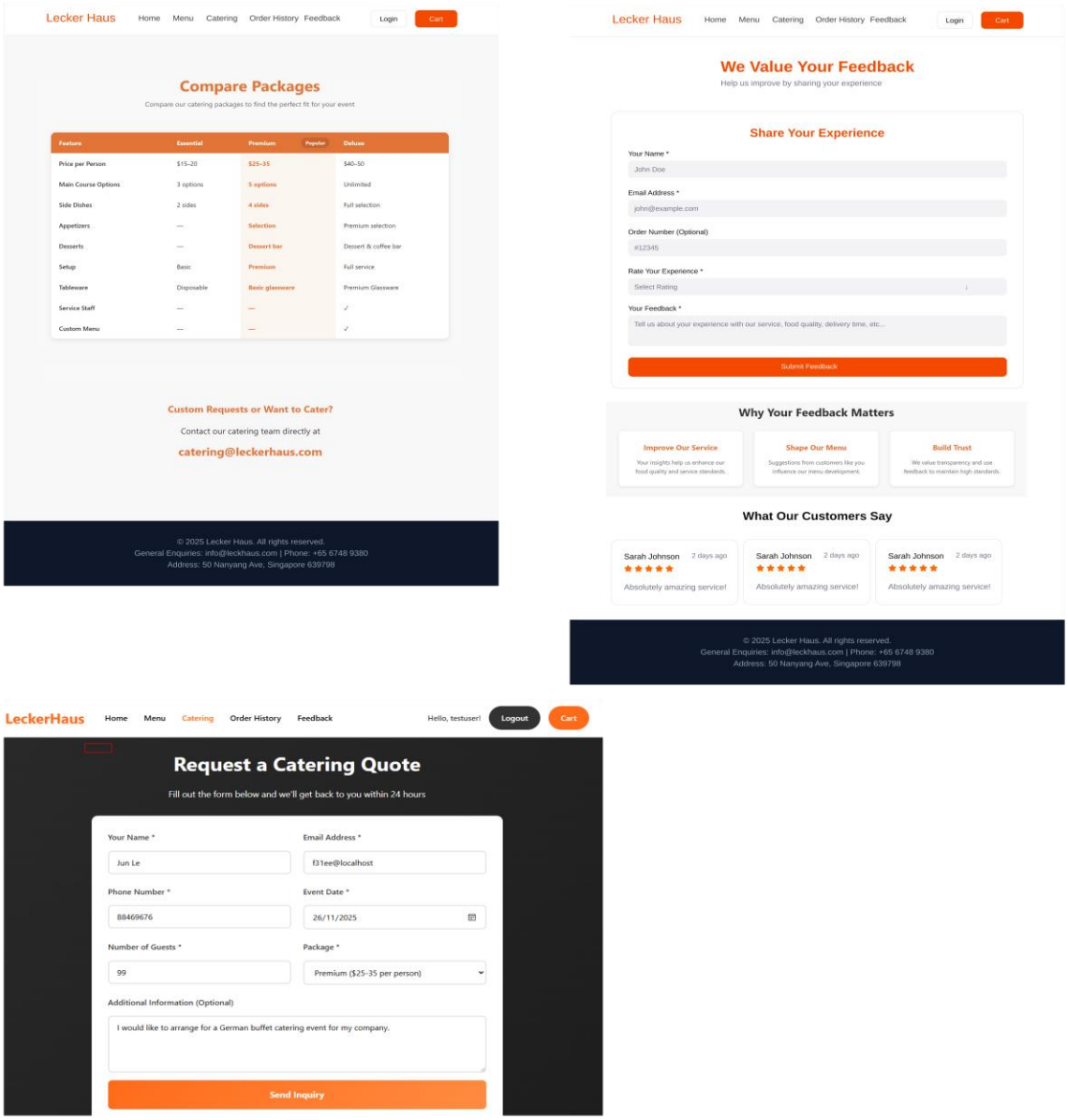


Figure 11. Top left: Catering page, top right: Feedback page, bottom: email functionality within catering page

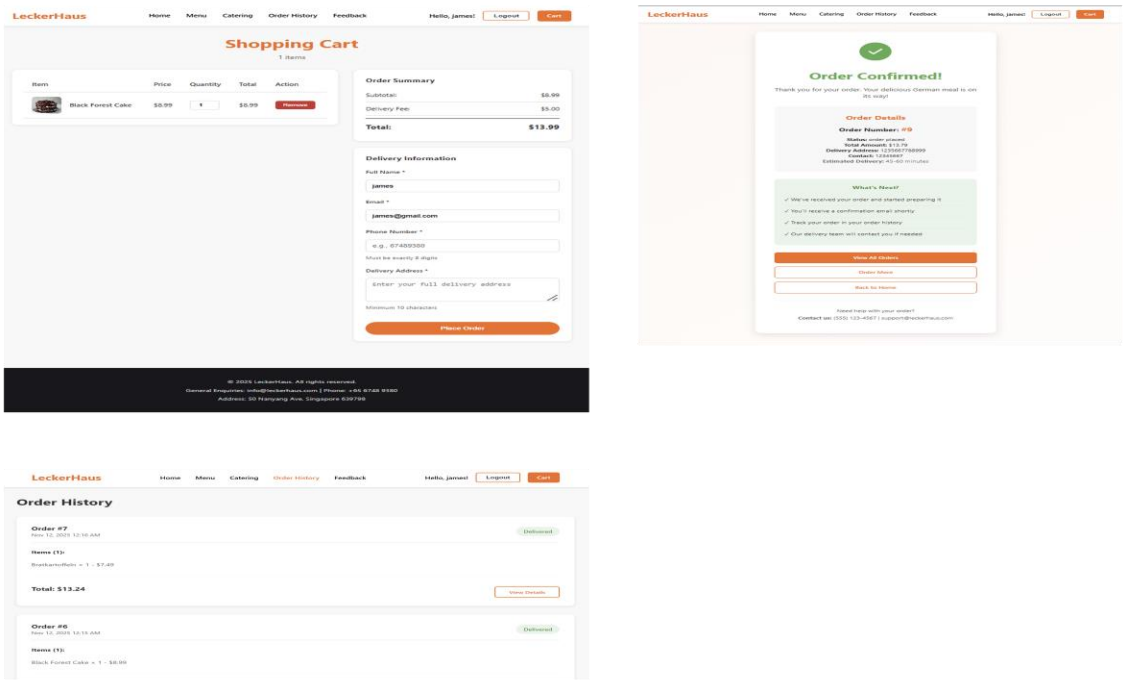


Figure 12. Left: Cart page, right: Confirmation page

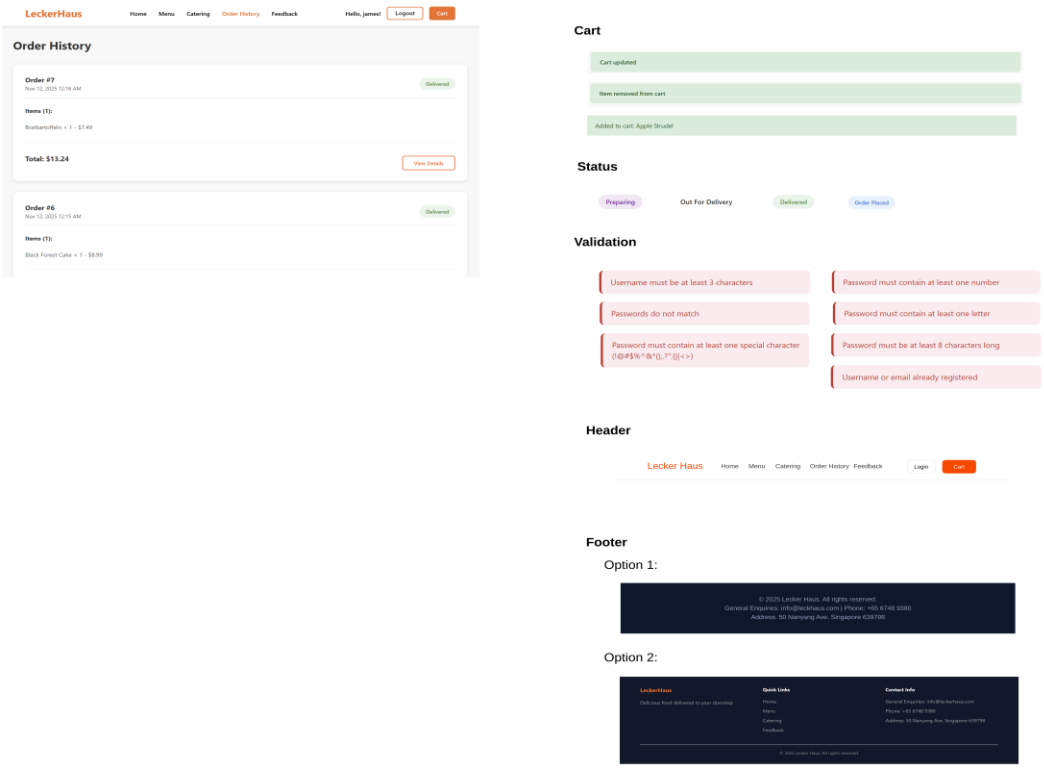


Figure 13. Left: Order history page, right: components design

Story Board

This storyboard illustrates the complete user journey through a food ordering web application, beginning with the homepage where users can explore featured items and navigate key sections.

Users authenticate through a streamlined login page before accessing the menu, where they can browse, search, filter by category, and sort items to find their desired dishes. Selected dishes are added to the shopping cart, where quantities can be adjusted before proceeding to checkout with delivery details. After order confirmation, users can track their order status through the order history page, watching as it progresses from “Order Placed” to “Delivered”. The application also includes a feedback system for users to share their dining experience and a catering inquiry form for special event requests. Through the journey, the design maintains consistent navigation with an orange accent colour (#ff6b1a) highlighting active pages and call-to-action buttons.

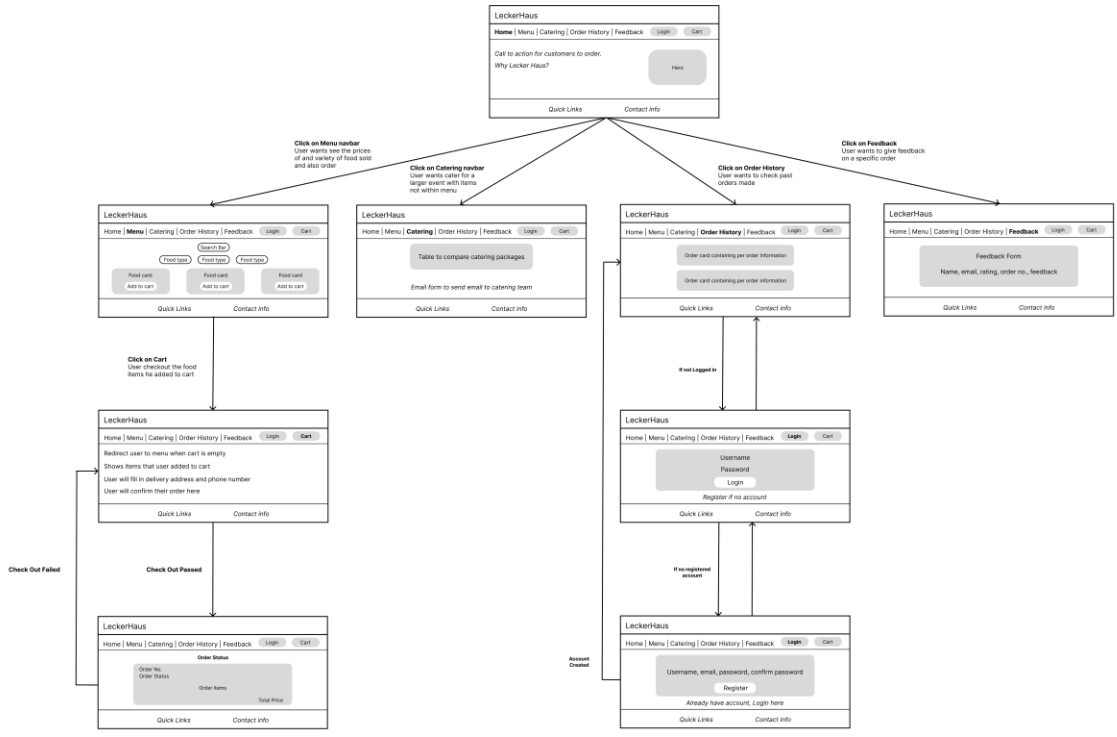


Figure 14. Story Board design

Database

The web application uses a MYSQL relational database with six tables: users, menu_items, cart_items, orders, orders_items and feedbacls. Foreign key relationships ensure data integrity: cart_items references users and menu_items, order_items references orders and menu_items, while feedback links to users. This normalised design supports core e-commerce functionality including authentication, browsing, cart management, checkout, order tracking, and customer feedback collection.

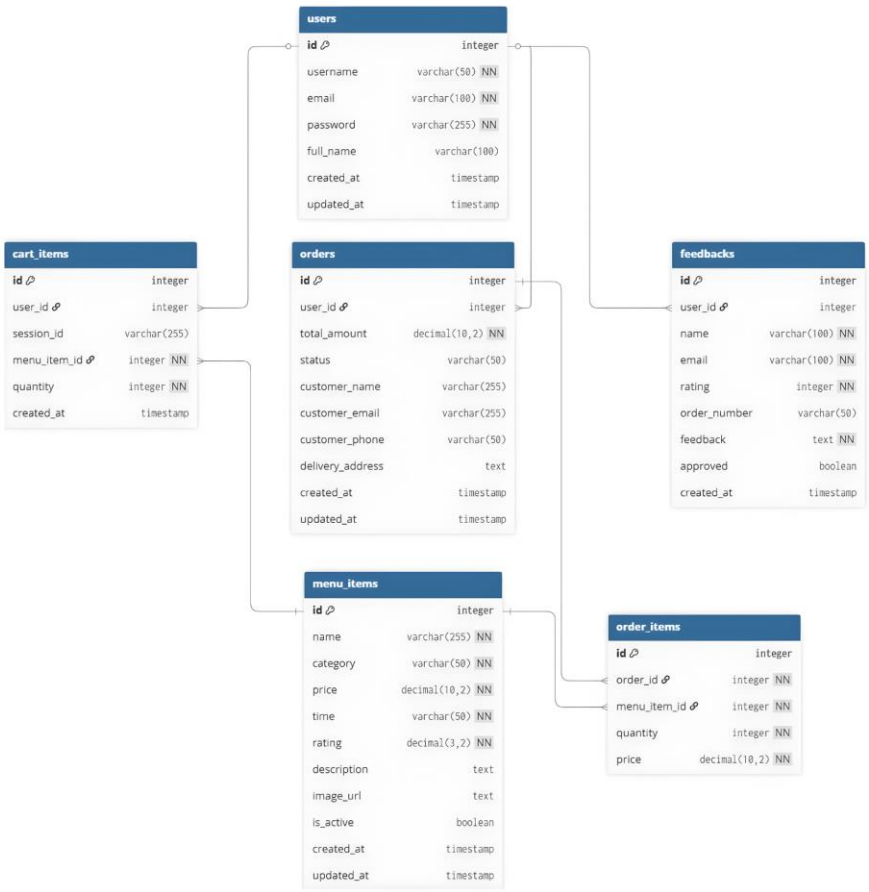


Figure 15. Database design

4. Implementation:

Description of the coding of TWO Functional Requirements

We would like to highlight the **search functionality** within the menu and home page as well as the **order history page** which showcases the order status tracking.

Search Functionality (F2)

Firstly, the search function. We wanted our landing page to be able to direct users to our website's main purpose, the ordering of food. Hence, we included the search bar on both the menu page and the home page, which is the landing page and the first thing that greets users.

We designed it to be placed at the top, so that it is easily accessible for users to find food items that they crave. Considering this, as our webpage sells German cuisine, the German dishes may not be named intuitively to English native speakers which is the main demographics of our target audience. Hence, we also **widened the search to food descriptions** as well. For example, users can search "meat" for meat dishes and food items with "meat" within its description would be returned to the user as well.

Order History Page (F6)

Secondly, the order history page. After placing an order, users are likely to check their order status for updates on when their food would arrive. Furthermore, they might want to review their order history, to find out what dishes they have ordered.

We hence decided to combine these user stories into a single page which allows users to obtain this information from one glance. We found that visually, the different **status should be of different colours** so that their change is apparent. Information like their total bill, phone number and location entered would also be able to be seen in their order history, per order.

5. Testing of web application

Describe the tests performed and the results for each Functional Requirement. A test case is a specific scenario or condition used to verify that the application behaves as expected. Test cases include: Input data; Expected Output; Test Method; Test Results. Make use of the table below. For each row, put in the description and result of each test case pertaining to a Functional Requirement.

Functional Requirement	Description of Test Case	Result
F1. User Authentication	Username, Email, and password validations	Refer to **figure 16** below. Error messages shown when requirements specific to each field are not met.
F2. Menu browsing and ordering	Filter menu items by selecting a specific category, search functionality, and confirmation message when adding to cart.	Refer to **figure 17** below. Filter and search logic and functioning. A confirmation message pops up per item added to the cart.
F3. Shopping Cart	Redirection to the menu when the cart is empty. Cart displays item information which can be edited or removed with a button. Total cost is calculated automatically. Placeholders for additional phone and location details before placing orders.	Refer to **figure 18** below. '+' and '-' buttons change quantity of items in cart and total amount will be calculated upon any change. Name and email fields are filled with database' information per user. Phone number field accepts exactly 8 digits only, and delivery address accepts minimum 10 characters.
F4. Checkout and order placement	User unable to proceed unless all fields match validation. Once conditions are met, order confirmation page is shown and cart is emptied. 3 buttons to lead user to menu, order history and home page.	Refer to **figure 19** below. If the fields' validations are not met, the order cannot be placed. Upon placing order, the order confirmation page will appear.
F5. Order status tracking	Status shown in order history. Button is solely for demonstration purpose, users would not be able to advance status themselves in a real setting.	Refer to **figure 20** below. Status shown, demo button to advance status and status update should be reflected. Advances from "Order Placed" to "Preparing", "Out for Delivery" and "Delivered". Going back to "View Details" order confirmation page reflects the status as well.
F6. Order History Viewing	Order history page shows order history with details for each user.	Refer to **figure 21** below. Order history only shows logged in user's history, not other users'.

F7. Customer Feedback Submission	User unable to proceed unless all fields match validation. Dropdown for review rating selection. Once conditions are met, thank you and review message will be shown. Feedback not shown immediately, only after backend approval.	Refer to **figure 22** below. Name, email, rating and feedback input is required. Email needs to be valid, rating must be selected from the dropdown. Review submitted will be set as "0" under the database "approved" column first. Changing this value from "0" to "1" shows the feedback in the feedback page.
F8. Catering Package	Table shows details, email shown to interested users for contact details.	Refer to **figure 23** below. **No test case** required for catering table. Catering from under F9.
F9. Email for catering (Localhost)	Order history page shows order history with details for each user.	Refer to **figure 24** below.

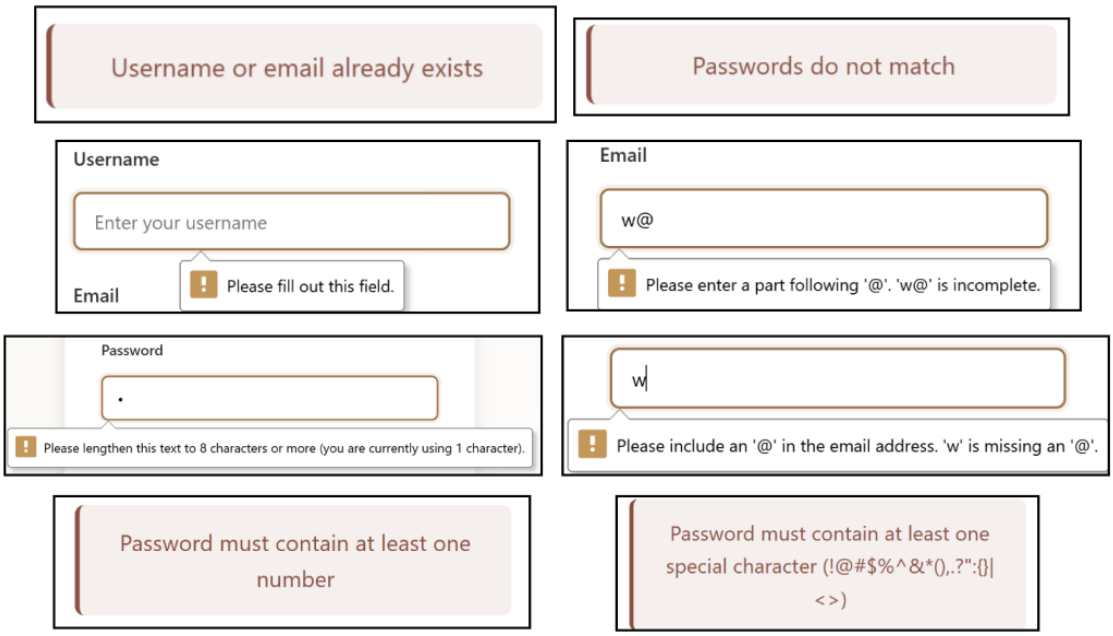


Figure 16. F1 - Possible cases of failed authentications

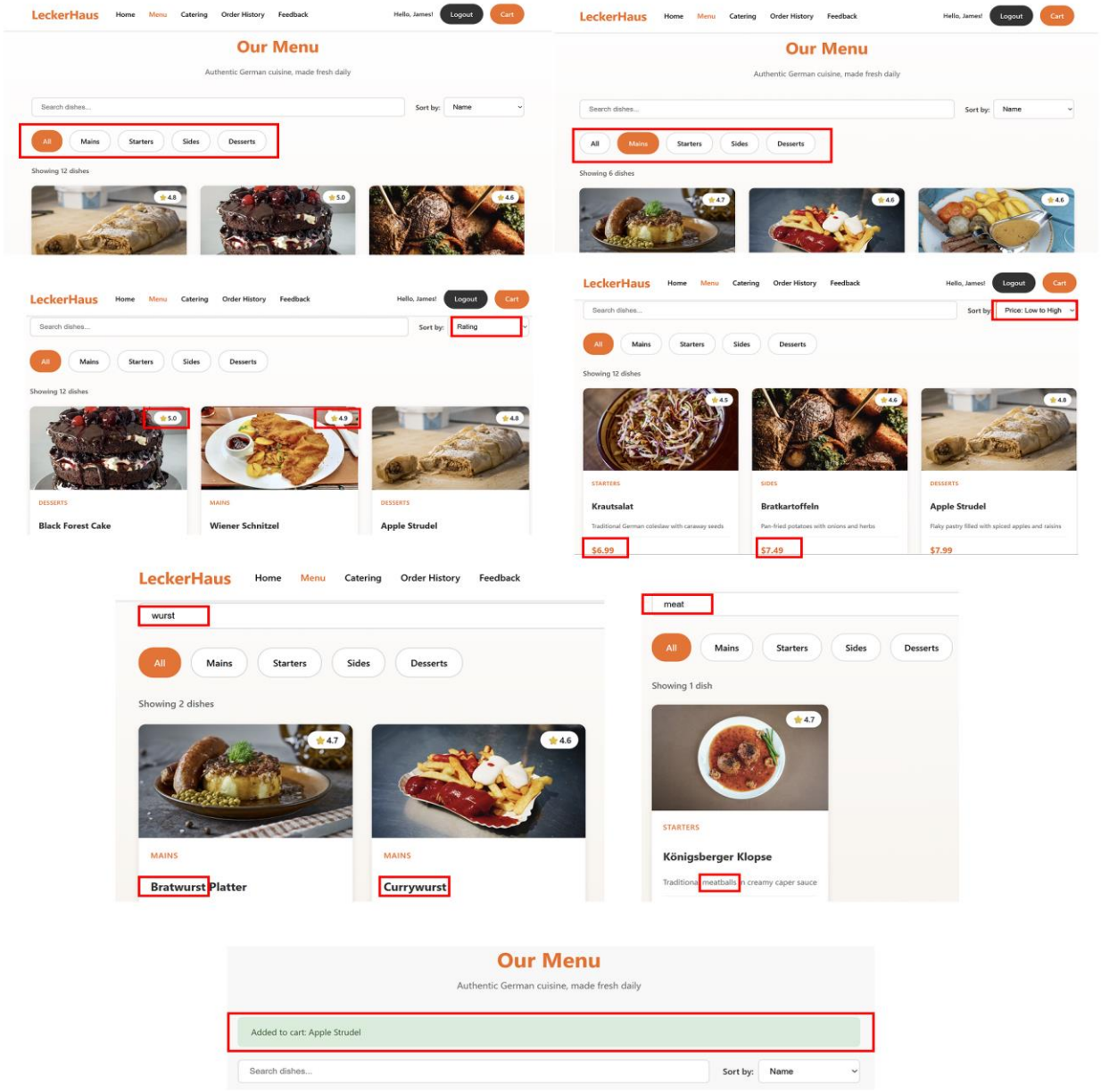


Figure 17. F2 – Filters by category, price and ratings and search bar

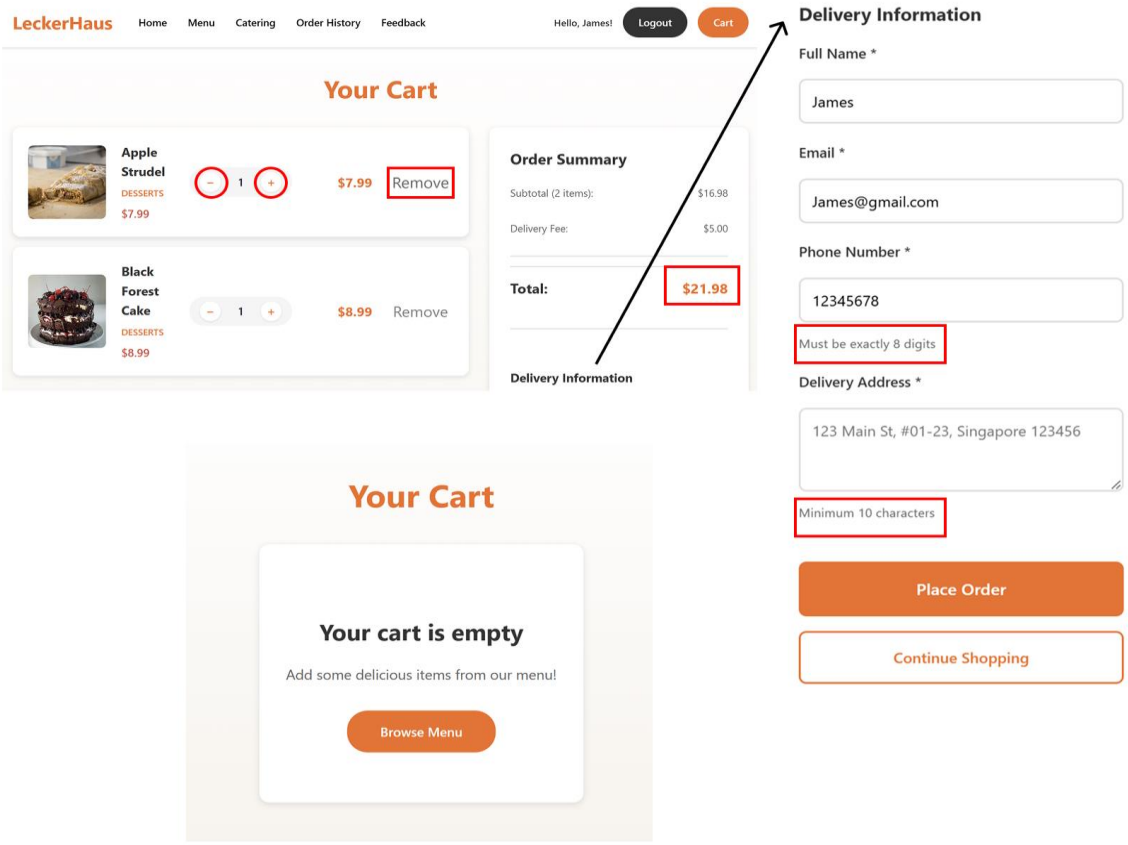


Figure 18. F3 – Cart page scenarios and functions

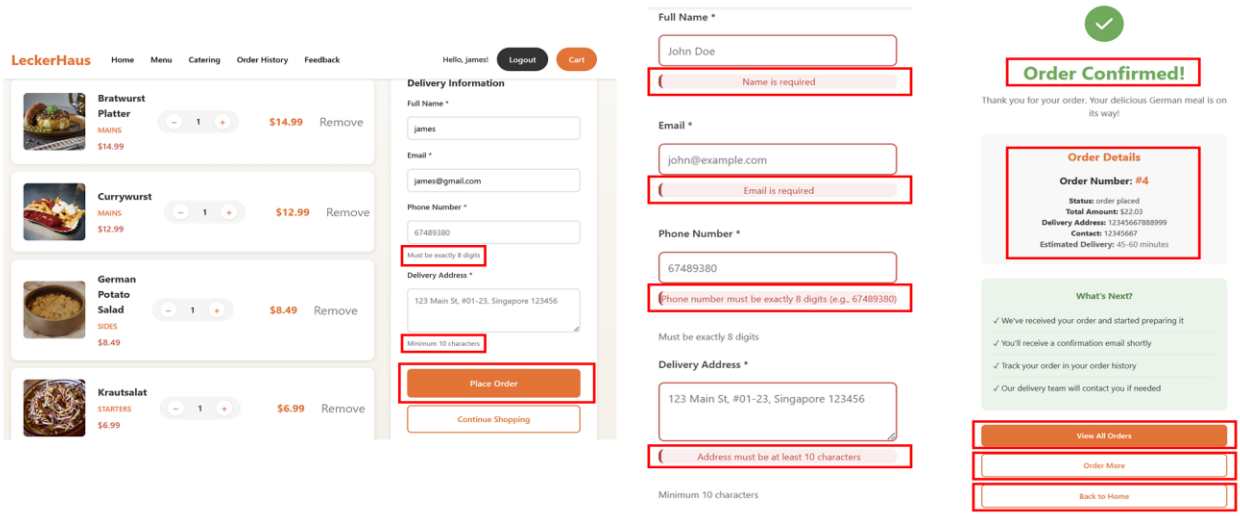


Figure 19. F4 – Checkout and order placement

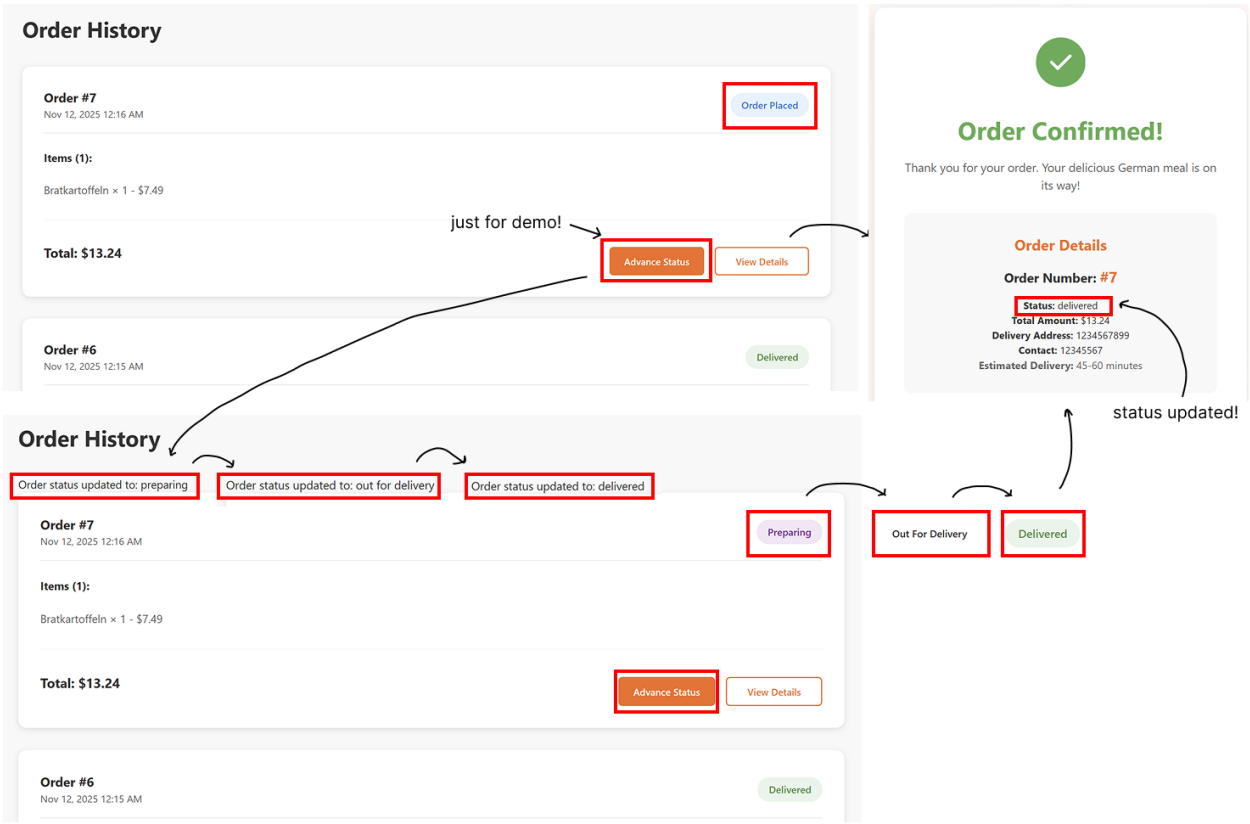


Figure 20. F5 – Order status tracking

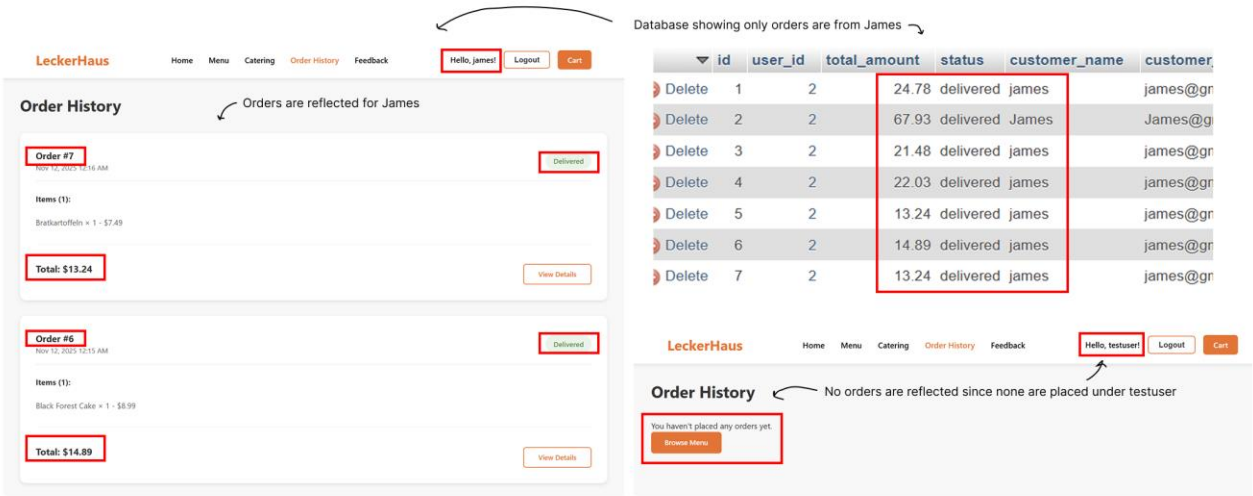


Figure 21. F6 – Order History Viewing

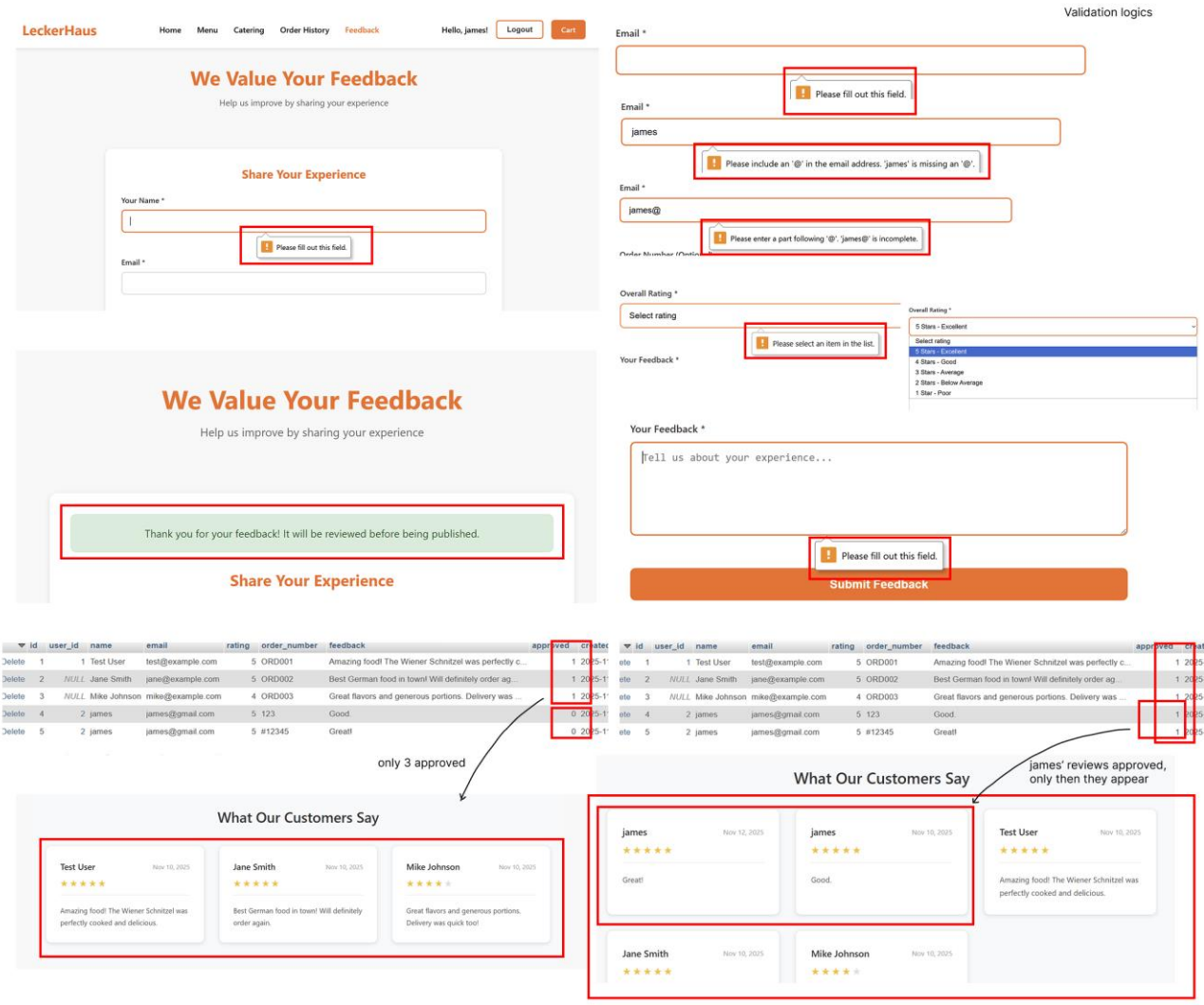


Figure 22. F7 – Customer Feedback Submission

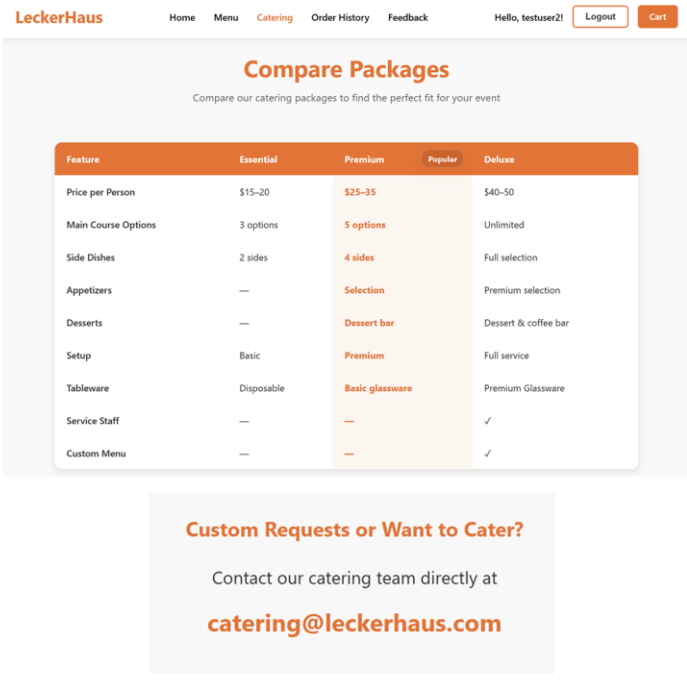


Figure 23. F8 – Catering Page Table

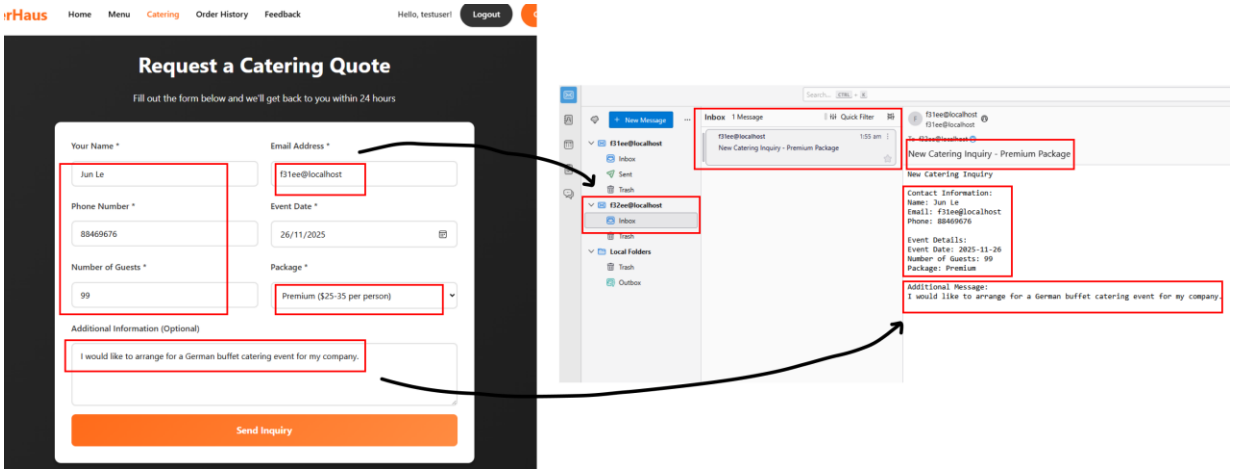


Figure 24. F9 – Catering Form Submission and receiving of email

6. Summary of Modern Enhancements

The modern implementation leverages React.js with Vite, introducing several contemporary web development techniques that significantly enhance user experience, performance, and maintainability beyond the traditional server-rendered PHP approach.

A) React Framework & Component Architecture

The modern implementation is built using React 18 with functional components and hooks (useState, useEffect, useContext, useNavigate). React's component-based architecture promotes code reusability and modularity.

Each feature (Header, Footer, Menu, Cart) is an isolated, self-contained component that can be independently developed, tested and maintained. This reduces code duplication compared to the traditional approach where header.php and footer.php are included in every page and any change to the header or footer requires verification across all pages. React components are imported once and automatically updated everywhere they are used.

Hence, changes to navigation, styling or user display logic only require updating Header.jsx once. The component uses React hooks (useAuth) to access global state without prop drilling. This reduces the maintenance burden and ensures UI consistency.

B) RESTful API architecture with JSON Communication & Client-Side Filtering

We implemented backend PHP scripts in /backend/api/ to serve as REST endpoints returning JSON responses, consumed by frontend via fetch API.

Traditional PHP tightly couples frontend (HTML) with backend (PHP logic) in the same files, making it impossible to adapt to mobile apps or third-party applications in the future. Every search or filter action triggers a form submission, database query, page reload, and re-rendering of the entire page.

On the other hand, RESTful APIs provide clean separation of concerns, backend focuses on data, frontend on presentation. The decoupled architecture enables multiple clients (web app, mobile app, admin dashboard) to consume the same backend APIs without code duplication. APIs return pure JSON data that can be cached, manipulated, and displayed differently by various clients. APIs can be documented, versioned, tested independently with tools like Postman, and maintained by other teams.

C) Single Page Application (SPA) with Client-Side Routing

The traditional PHP web application requires full reloads on every navigation, re-downloading HTML, CSS and JavaScript, re-executing scripts, and re-rendering the entire layout including unchanged header and footer. This can be observed with visible flashing of website when the menu search is performed in the Traditional version.

The React Router DOM (v6) handles all navigation without full page reloads. The application loads once, then dynamically swaps page content based on the URL route. The SPA loads the application shell (Header, Footer, CSS, JavaScript) once during intial page load. Subsequent navigation only fetches and renders the specific page content, leaving the header and footer mounted in the DOM. When users click Home -> Menu -> Cart -> Checkout, React Router intercepts the click, prevents the default browser navigation then updates the URL using the History API, and renders only the new page component in milliseconds. Navigation is instantaneous with smooth transitions, providing a native app-like experience with zero flashing, eliminating the jarring “click-wait-reload" cycle familiar to the traditional version.

7. Conclusion

Summarize what has been achieved and what has been deferred in the current stage.

We developed a food web application for German cuisine by taking ourselves through the typical user journey. We considered the next steps as well as ideal functionalities for the smooth transition of each process. This meant quick redirections or backtracking through buttons on each page, and clear directions and information for simple navigation. Furthermore, we considered security risks and implemented password hashing. SQL injection prevention via prepared statements, and XSS protection through output escaping.

The modern approach takes this to the next level with faster loading through react routers, and pages do not need to keep refreshing, reducing wait time. In a real-world setting, this is one of the important and more highly regarded features that users note.

We can further improve our application with transitions in the future, such as the use of the GSAP library for a nicer touch to our web application. Further features such as map API integration can also be considered to show the route that the delivery is taking. We also note that we could implement a forget password feature, where users can retrieve their account through email or message.

Appendices:

Appendix 1: Source Codes **with proper comments for evaluation**
Appendix 2: Contribution to the project by each team member shown in a table (see Table A2.1 and A2.2).
Appendix 3: Contribution to the report by each team member shown in a table (see Table A3).

Appendix 1 Source Codes

Database Schema

1. Users: Stores user account information for authentication and order tracking, related to F1 (User Authentication).

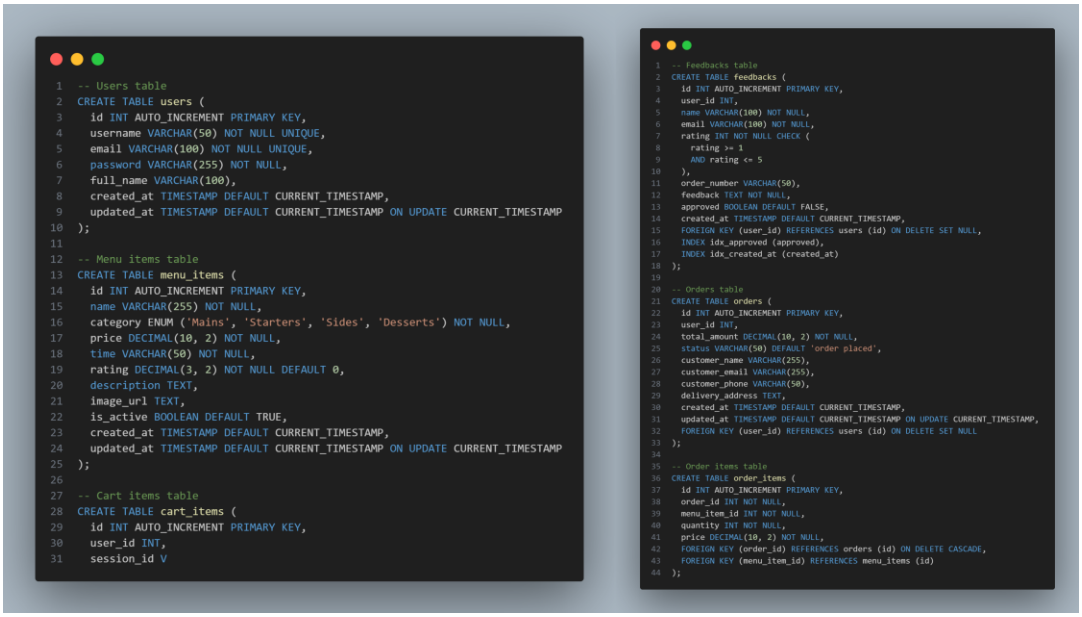
2. Menu Items: Stores restaurant menu with categorised food items, related to F2 (Menu Browsing and Ordering)

3. Cart Items: Stores shopping cart data linked to user sessions, related to F3 (Shopping Cart)

4. Feedbacks: Stores customer feedback and reviews with approval workflow, related to F7

5. Orders: Stores order information with customer and delivery details, related to F4 (Checkout and Order Placement), F5 (Order Status Tracking), F6 (Order History)

6. Order Items: Stores line items for each order, related to F4.



```sql
-- Users table
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(50) NOT NULL UNIQUE,
  email VARCHAR(100) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL,
  full_name VARCHAR(100),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- Menu items table
CREATE TABLE menu_items (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  category ENUM ('Mains', 'Starters', 'Sides', 'Desserts') NOT NULL,
  price DECIMAL(10, 2) NOT NULL,
  time VARCHAR(50) NOT NULL,
  rating DECIMAL(3, 2) NOT NULL DEFAULT 0,
  description TEXT,
  image_url TEXT,
  is_active BOOLEAN DEFAULT TRUE,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- Cart items table
CREATE TABLE cart_items (
  id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT,
  session_id V
```

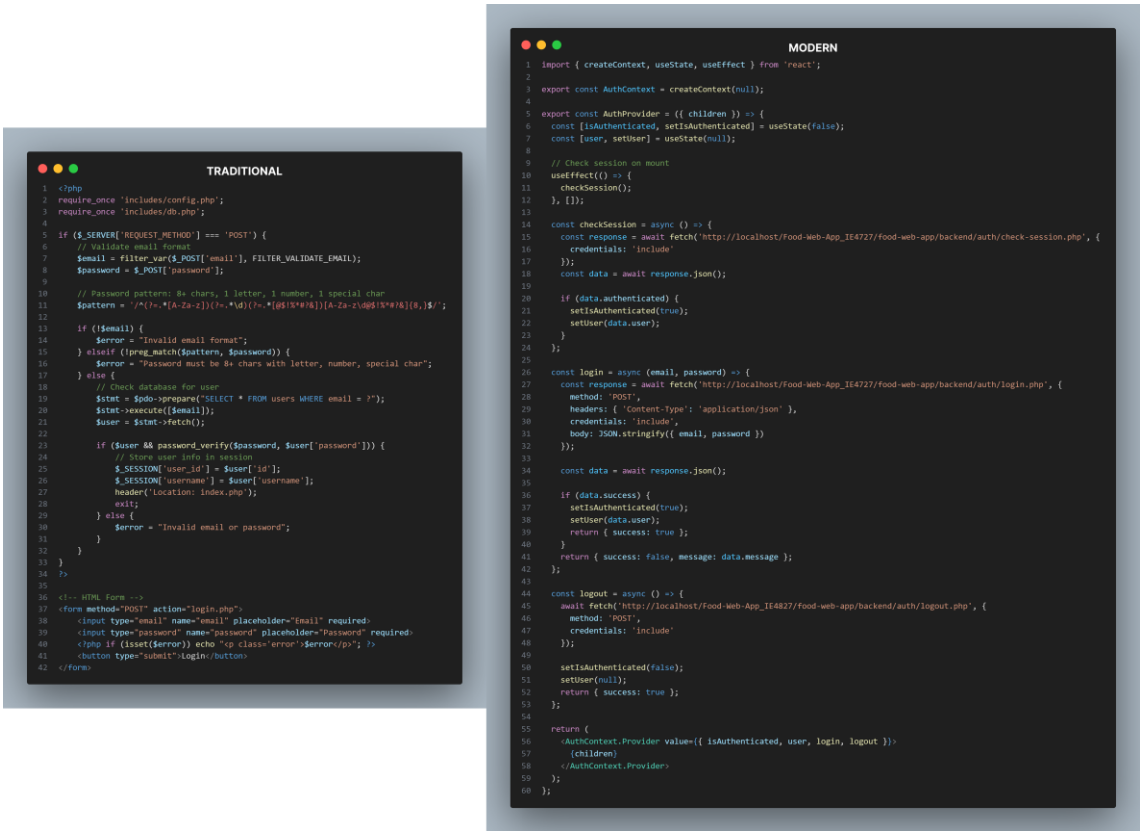
```sql
-- Feedbacks table
CREATE TABLE feedbacks (
  id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT,
  name VARCHAR(100) NOT NULL,
  email VARCHAR(100) NOT NULL,
  rating INT NOT NULL CHECK (
    rating >= 1
    AND rating <= 5
  ),
  order_number VARCHAR(50),
  feedback TEXT NOT NULL,
  approved BOOLEAN DEFAULT FALSE,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users (id) ON DELETE SET NULL,
  INDEX idx_approved (approved),
  INDEX idx_created_at (created_at)
);

-- Orders table
CREATE TABLE orders (
  id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT,
  total_amount DECIMAL(10, 2) NOT NULL,
  status VARCHAR(50) DEFAULT 'order placed',
  customer_name VARCHAR(255),
  customer_email VARCHAR(255),
  customer_phone VARCHAR(50),
  delivery_address TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users (id) ON DELETE SET NULL
);

-- Order items table
CREATE TABLE order_items (
  id INT AUTO_INCREMENT PRIMARY KEY,
  order_id INT NOT NULL,
  menu_item_id INT NOT NULL,
  quantity INT NOT NULL,
  price DECIMAL(10, 2) NOT NULL,
  FOREIGN KEY (order_id) REFERENCES orders (id) ON DELETE CASCADE,
  FOREIGN KEY (menu_item_id) REFERENCES menu_items (id)
);
```

Database Configuration

Traditional	Modern
Defines database connection constants and starts PHP session. Used by all PHP pages via require_once 'includes/config.php'.	Class-based database connection for REST API endpoints. Instantiated in each API file for isolated connections.

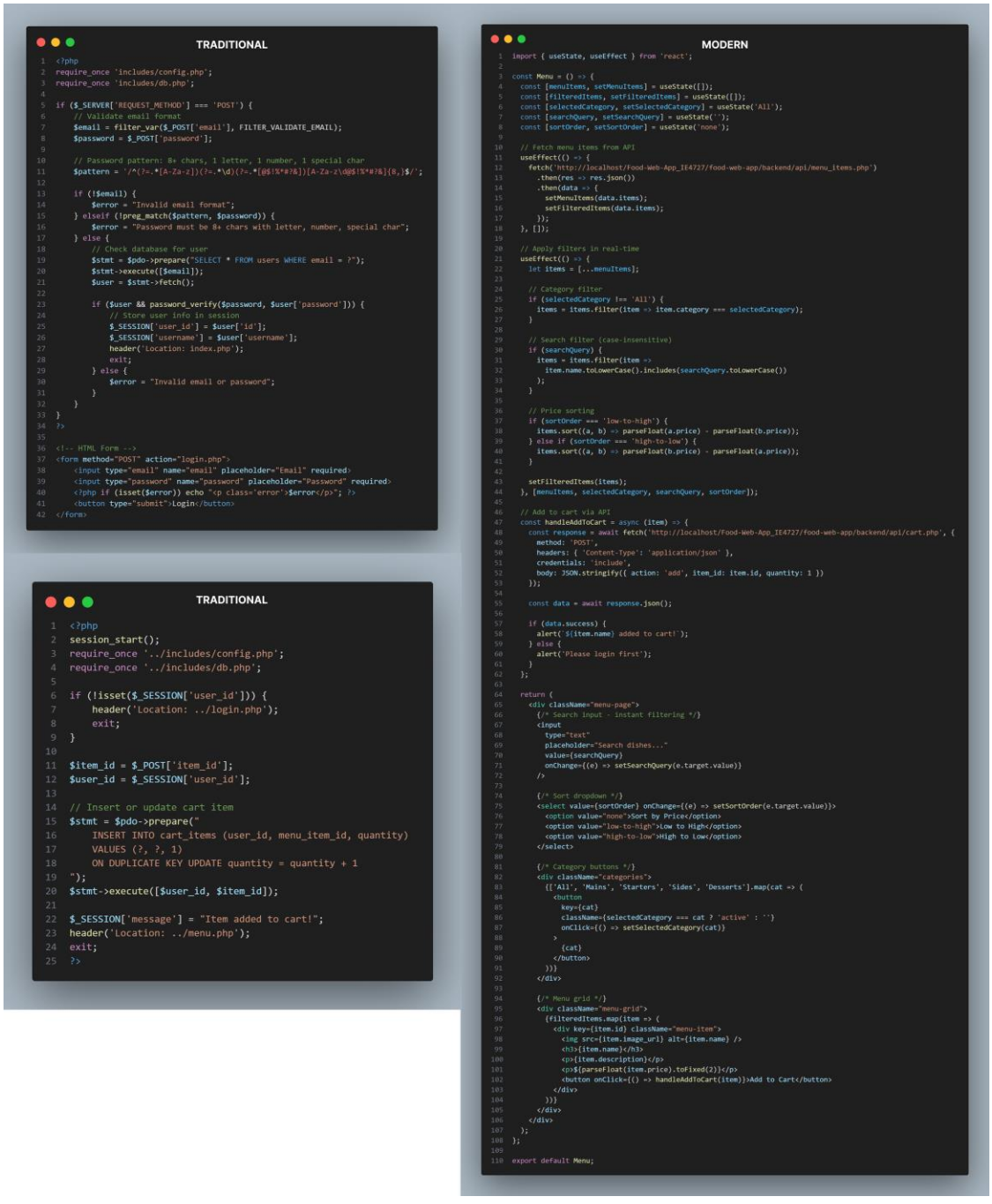
F1: Authentication

Traditional	Modern
Server-side rendered page with form submission to same page. Session created on successful login, redirecting to index.php	React Context for global authentication state management Provides login/logout functions and user state to all components



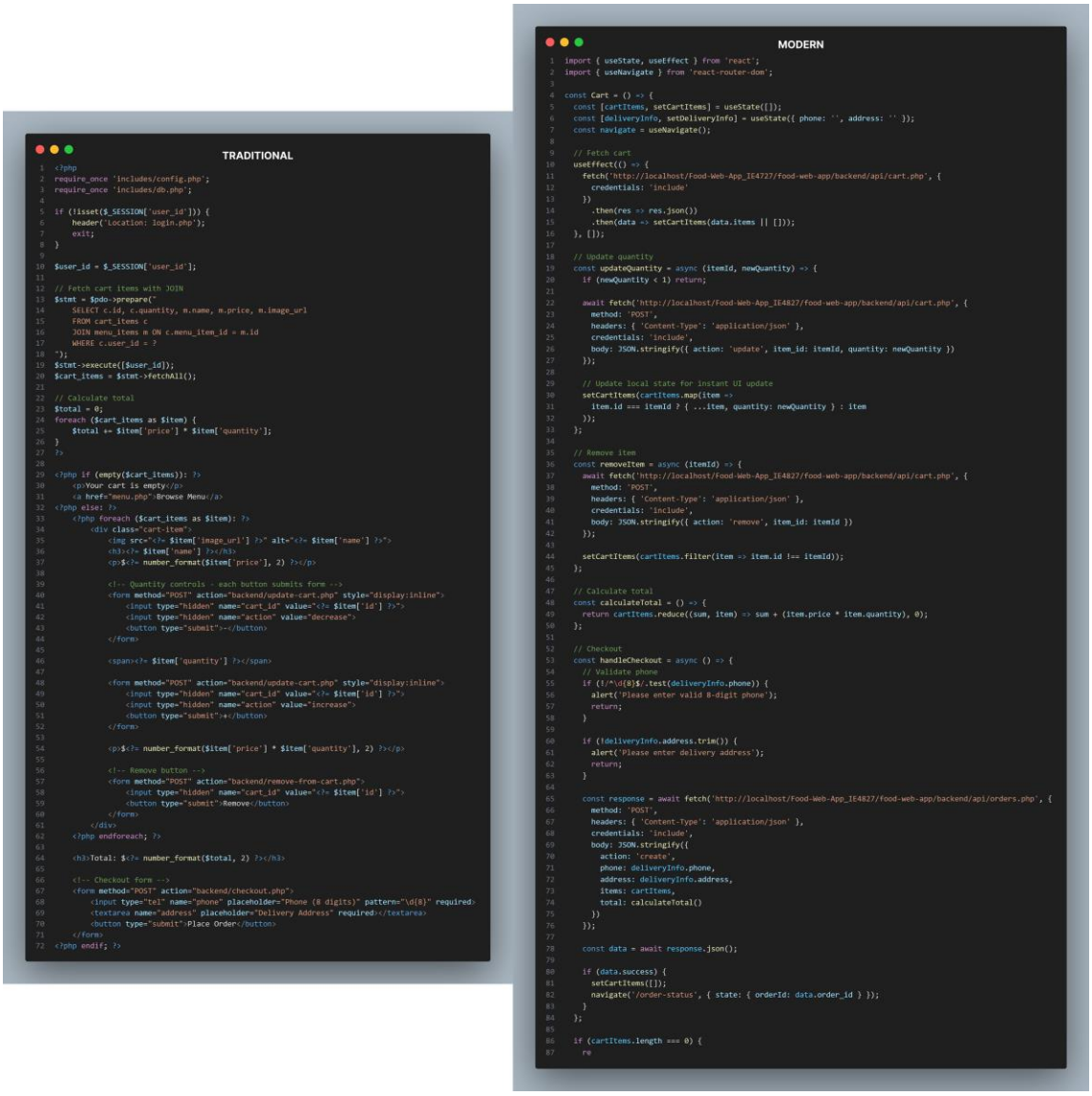
F2: Menu Browsing and Ordering

Traditional	Modern
Server-side filtering, search, and sorting with page reload. Query parameters passed via GET, results generated server-side.	Client-side filtering, search, and sorting without page reload. Real-time updates using React State and useEffect hooks.



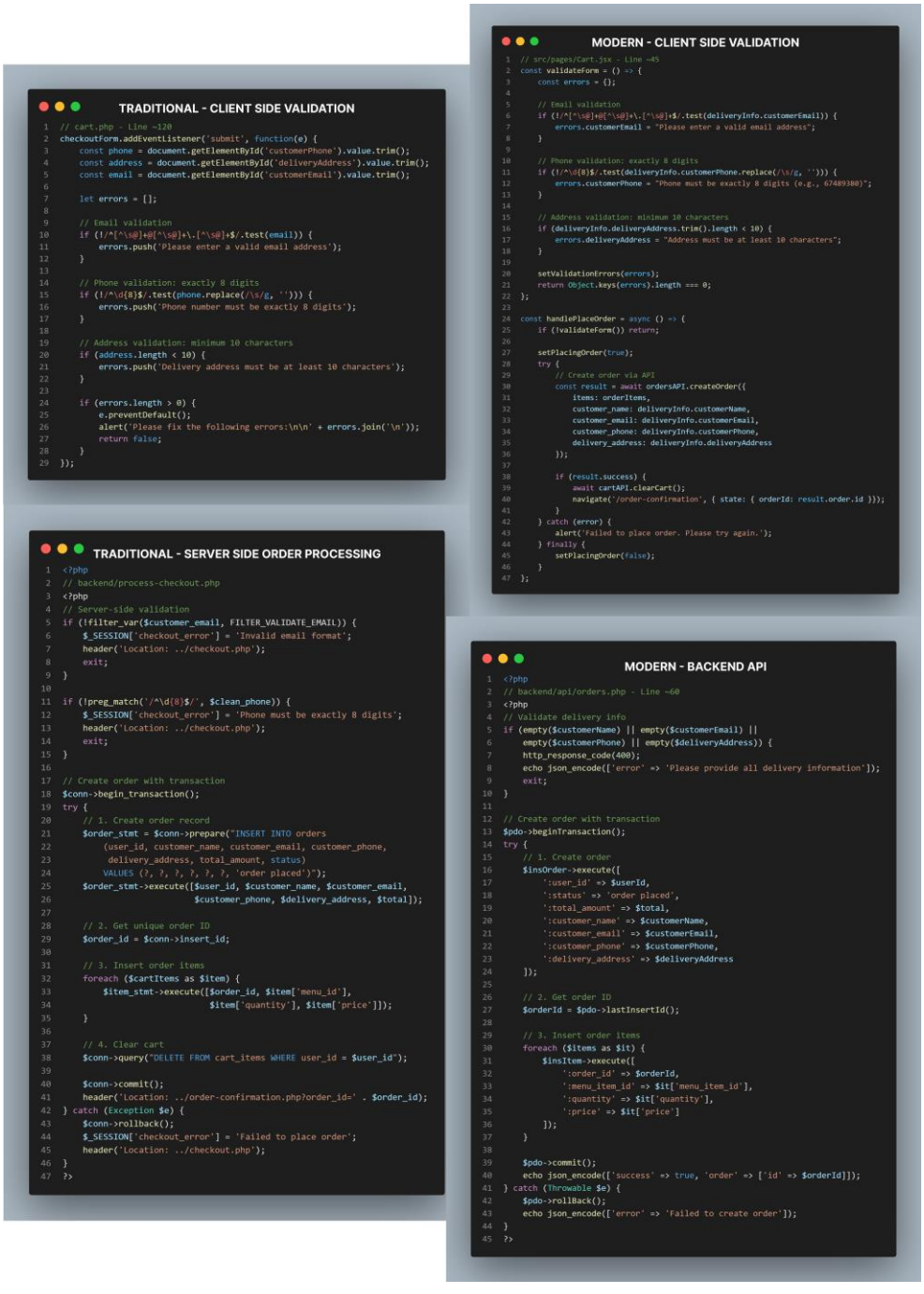
F3: Shopping Cart

Traditional	Modern
Server-side cart page with form submissions for quantity updates.	Client-side cart with quantity updates via API



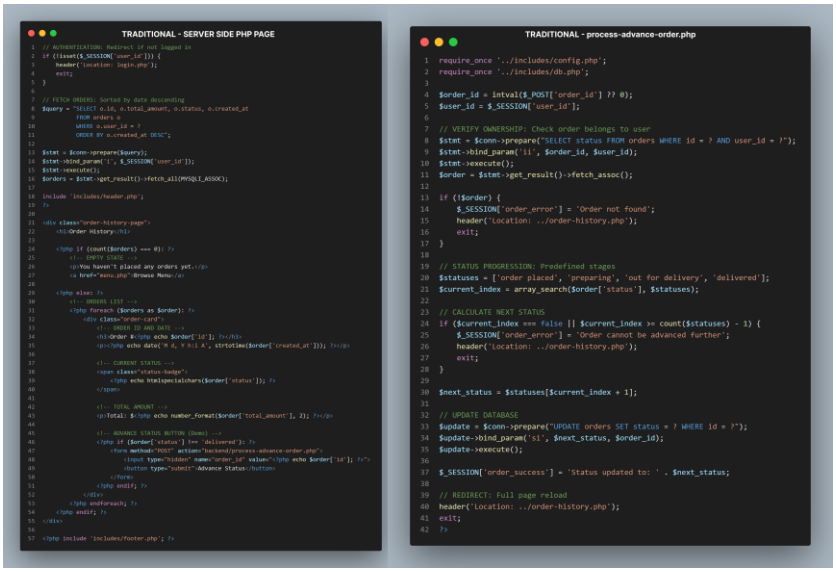
F4: Checkout and Order Placement

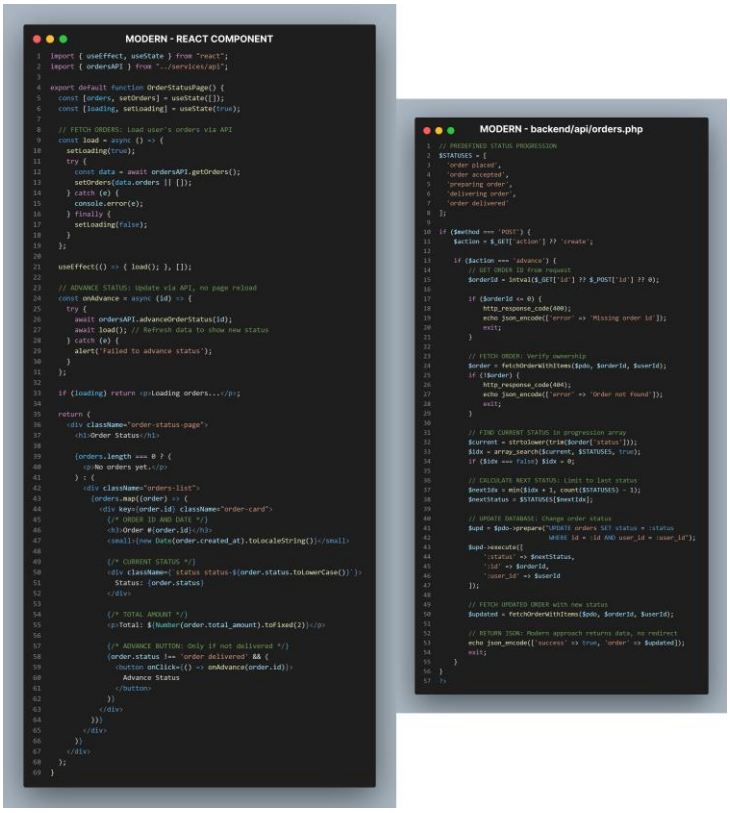
Traditional	Modern
Direct DOM manipulation with vanilla JavaScript. document.getElementbyId() to access form elements. Full page reload occurs on successful submissions to backend.	React state management with controlled components. Form values stored in React state. (deliveryinfo object) No page reloads. Navigation to confirmation page using React router.



F5: Order Status Tracking

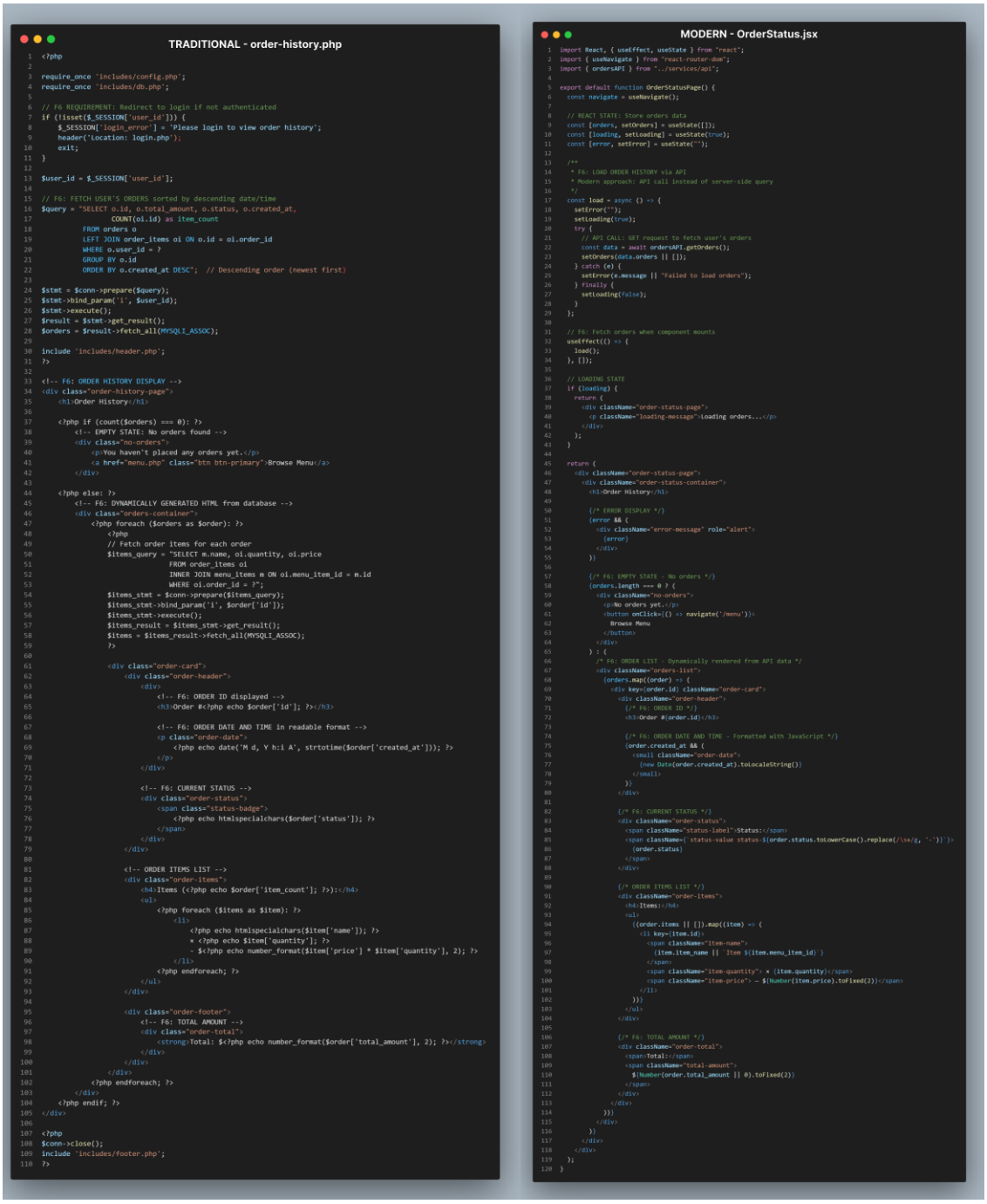
Traditional	Modern
Server-side PHP page that dynamically generates HTML showing user's orders. Allows status updates via form submission (demo delivery flow).	React component that displays order status. Fetches orders via API call. Updates status through API (no form submission and reflects changes in UI without page reload.





F6: Order History Viewing

Traditional	Modern
Server-side PHP page that dynamically generates HTML displaying past orders. Redirects user login if not authenticated. Dynamic HTML generation based on db queries.	React component that fetches and displays order history. Use protected routes to handle authentication. Fetches orders via API call.



TRADITIONAL - order-history.php

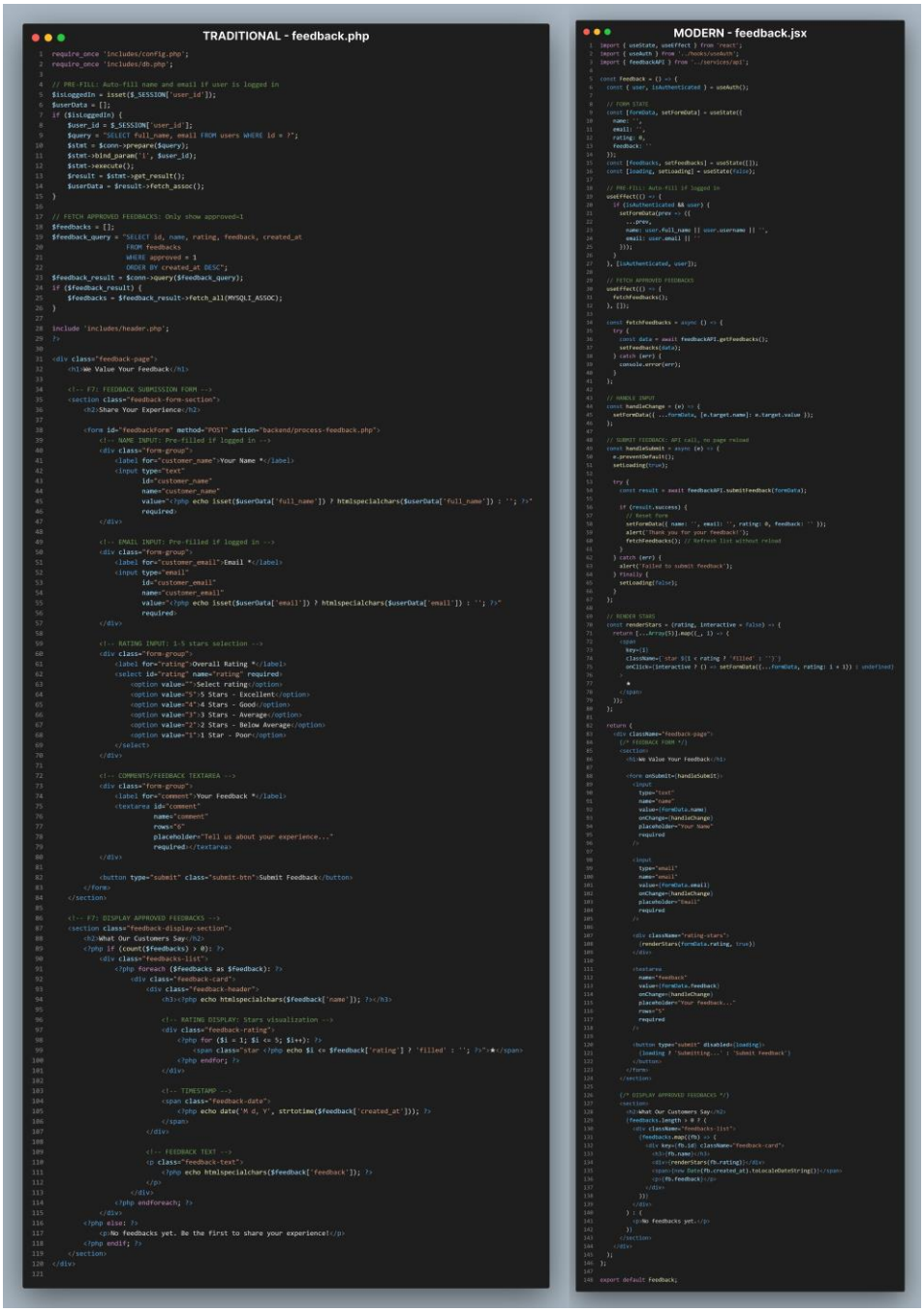
```php
<?php

require_once 'includes/config.php';
require_once 'includes/db.php';

// F6 REQUIREMENT: Redirect to login if not authenticated
if (!isset($_SESSION['user_id'])) {
    $_SESSION['login_error'] = 'Please login to view order history';
    header('Location: login.php');
    exit;
}

$user_id = $_SESSION['user_id'];

// F6: FETCH USER'S ORDERS sorted by descending date/time
$query = "SELECT o.id, o.total_amount, o.status, o.created_at,
                 COUNT(oi.id) as item_count
          FROM orders o
          LEFT JOIN order_items oi ON o.id = oi.order_id
          WHERE o.user_id = ?
          GROUP BY o.id
          ORDER BY o.created_at DESC";  // Descending order (newest first)

$stmt = $conn->prepare($query);
$stmt->bind_param('i', $user_id);
$stmt->execute();
$result = $stmt->get_result();
$orders = $result->fetch_all(MYSQLI_ASSOC);

include 'includes/header.php';
?>

<!-- F6: ORDER HISTORY DISPLAY -->
<div class="order-history-page">
    <h1>Order History</h1>

    <?php if (count($orders) === 0): ?>
        <!-- EMPTY STATE: No orders found -->
        <div class="no-orders">
            <p>You haven't placed any orders yet.</p>
            <a href="menu.php" class="btn btn-primary">Browse Menu</a>
        </div>

    <?php else: ?>
        <!-- F6: DYNAMICALLY GENERATED HTML from database -->
        <div class="orders-container">
            <?php foreach ($orders as $order): ?>
                <?php
                // Fetch order items for each order
                $items_query = "SELECT m.name, oi.quantity, oi.price
                                FROM order_items oi
                                INNER JOIN menu_items m ON oi.menu_item_id = m.id
                                WHERE oi.order_id = ?";
                $items_stmt = $conn->prepare($items_query);
                $items_stmt->bind_param('i', $order['id']);
                $items_stmt->execute();
                $items_result = $items_stmt->get_result();
                $items = $items_result->fetch_all(MYSQLI_ASSOC);
                ?>

                <div class="order-card">
                    <div class="order-header">
                        <div>
                            <!-- F6: ORDER ID displayed -->
                            <h3>Order #<?php echo $order['id']; ?></h3>

                            <!-- F6: ORDER DATE AND TIME in readable format -->
                            <p class="order-date">
                                <?php echo date('M d, Y h:i A', strtotime($order['created_at'])); ?>
                            </p>
                        </div>

                        <!-- F6: CURRENT STATUS -->
                        <div class="order-status">
                            <span class="status-badge">
                                <?php echo htmlspecialchars($order['status']); ?>
                            </span>
                        </div>
                    </div>

                    <!-- ORDER ITEMS LIST -->
                    <div class="order-items">
                        <h4>Items (<?php echo $order['item_count']; ?>):</h4>
                        <ul>
                            <?php foreach ($items as $item): ?>
                                <li>
                                    <?php echo htmlspecialchars($item['name']); ?>
                                    x <?php echo $item['quantity']; ?>
                                    - $<?php echo number_format($item['price'] * $item['quantity'], 2); ?>
                                </li>
                            <?php endforeach; ?>
                        </ul>
                    </div>

                    <div class="order-footer">
                        <!-- F6: TOTAL AMOUNT -->
                        <div class="order-total">
                            <strong>Total: $<?php echo number_format($order['total_amount'], 2); ?></strong>
                        </div>
                    </div>
                </div>
            <?php endforeach; ?>
        </div>
    <?php endif; ?>
</div>

<?php
$conn->close();
include 'includes/footer.php';
?>
```

MODERN - OrderStatus.jsx

```jsx
import React, { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import { ordersAPI } from "../services/api";

export default function OrderStatusPage() {
  const navigate = useNavigate();

  // REACT STATE: Store orders data
  const [orders, setOrders] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState("");

  /**
   * F6: LOAD ORDER HISTORY via API
   * Modern approach: API call instead of server-side query
   */
  const load = async () => {
    setError("");
    setLoading(true);
    try {
      // API CALL: GET request to fetch user's orders
      const data = await ordersAPI.getOrders();
      setOrders(data.orders || []);
    } catch (e) {
      setError(e.message || "Failed to load orders");
    } finally {
      setLoading(false);
    }
  };

  // F6: Fetch orders when component mounts
  useEffect(() => {
    load();
  }, []);

  // LOADING STATE
  if (loading) {
    return (
      <div className="order-status-page">
        <p className="loading-message">Loading orders...</p>
      </div>
    );
  }

  return (
    <div className="order-status-page">
      <div className="order-status-container">
        <h1>Order History</h1>

        {/* ERROR DISPLAY */}
        {error && (
          <div className="error-message" role="alert">
            {error}
          </div>
        )}

        {/* F6: EMPTY STATE - No orders */}
        {orders.length === 0 ? (
          <div className="no-orders">
            <p>No orders yet.</p>
            <button onClick={() => navigate('/menu')}>
              Browse Menu
            </button>
          </div>
        ) : (
          /* F6: ORDER LIST - Dynamically rendered from API data */
          <div className="orders-list">
            {orders.map((order) => (
              <div key={order.id} className="order-card">
                <div className="order-header">
                  {/* F6: ORDER ID */}
                  <h3>Order #{order.id}</h3>

                  {/* F6: ORDER DATE AND TIME - Formatted with JavaScript */}
                  {order.created_at && (
                    <small className="order-date">
                      {new Date(order.created_at).toLocaleString()}
                    </small>
                  )}
                </div>

                {/* F6: CURRENT STATUS */}
                <div className="order-status">
                  <span className="status-label">Status:</span>
                  <span className={`status-value status-${order.status.toLowerCase().replace(/\s+/g, '-')}`}>
                    {order.status}
                  </span>
                </div>

                {/* ORDER ITEMS LIST */}
                <div className="order-items">
                  <h4>Items:</h4>
                  <ul>
                    {(order.items || []).map((item) => (
                      <li key={item.id}>
                        <span className="item-name">
                          {item.item_name || `Item ${item.menu_item_id}`}
                        </span>
                        <span className="item-quantity">× {item.quantity}</span>
                        <span className="item-price">— ${Number(item.price).toFixed(2)}</span>
                      </li>
                    ))}
                  </ul>
                </div>

                {/* F6: TOTAL AMOUNT */}
                <div className="order-total">
                  <span>Total:</span>
                  <span className="total-amount">
                    ${Number(order.total_amount || 0).toFixed(2)}
                  </span>
                </div>
              </div>
            ))}
          </div>
        )}
      </div>
    </div>
  );
}
```

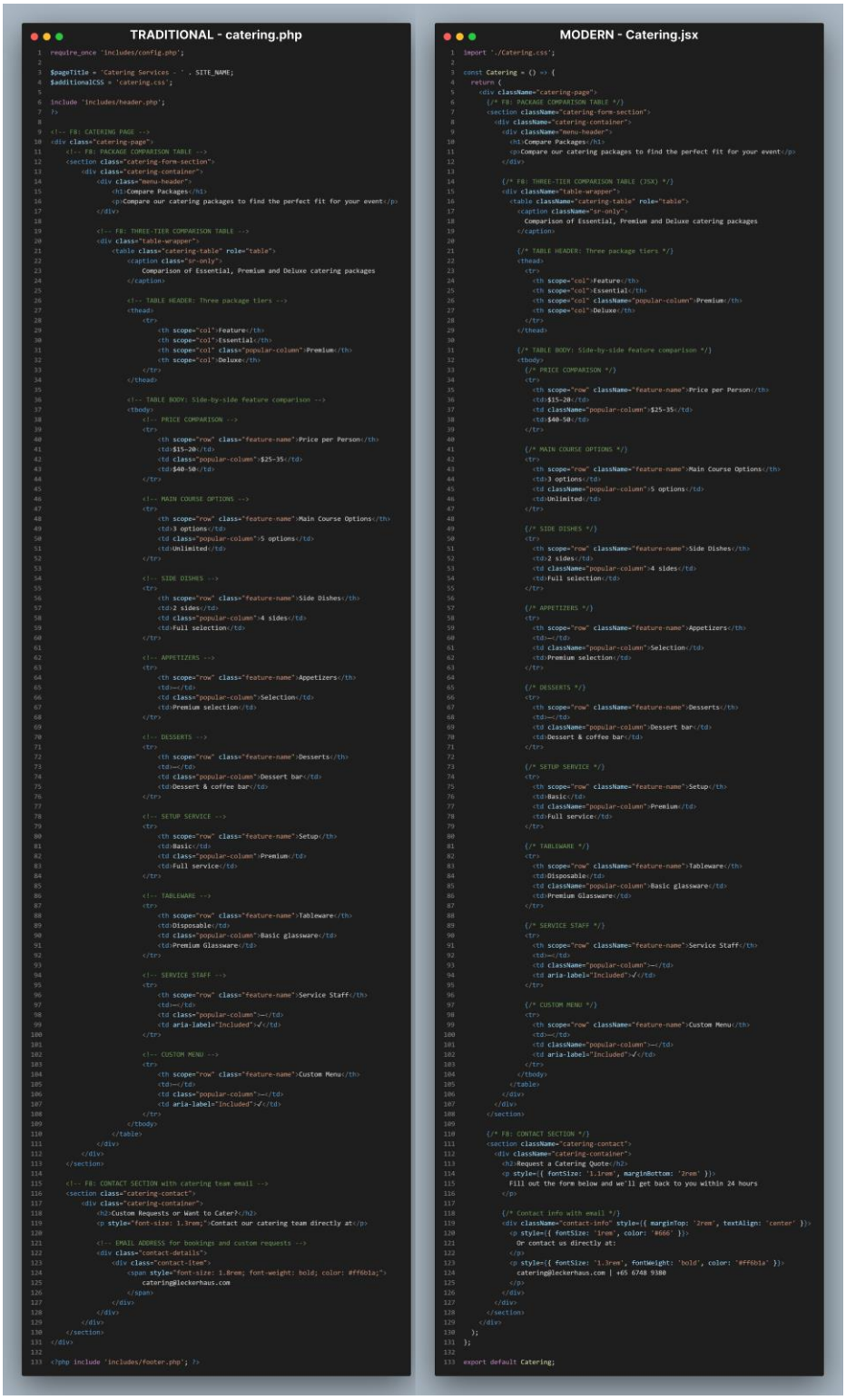
F7: Customer Feedback Submission

Traditional	Modern
Server-side PHP page with feedback form validated on both client-side and server-side. Stores feedback with "approved" flag, defaults to 0/unapproved. Full page reload after submission.	React component with API-based feedback submission, client-side validation. Stores feedback with approved=0 via API. No page reload on submission.



F8: Catering Package

Traditional	Modern
Static page showing three-tier package comparison table. Served with Server-side PHP.	Client-side rendered static page, served with Client-side React JSX.



TRADITIONAL - catering.php

```php
require_once 'includes/config.php';

$pageTitle = 'Catering Services - ' . SITE_NAME;
$additionalCSS = 'catering.css';

include 'includes/header.php';
?>

<!-- F8: CATERING PAGE -->
<div class="catering-page">
    <!-- F8: PACKAGE COMPARISON TABLE -->
    <section class="catering-form-section">
        <div class="catering-container">
            <div class="menu-header">
                <h1>Compare Packages</h1>
                <p>Compare our catering packages to find the perfect fit for your event</p>
            </div>

            <!-- F8: THREE-TIER COMPARISON TABLE -->
            <div class="table-wrapper">
                <table class="catering-table" role="table">
                    <caption class="sr-only">
                        Comparison of Essential, Premium and Deluxe catering packages
                    </caption>

                    <!-- TABLE HEADER: Three package tiers -->
                    <thead>
                        <tr>
                            <th scope="col">Feature</th>
                            <th scope="col">Essential</th>
                            <th scope="col" class="popular-column">Premium</th>
                            <th scope="col">Deluxe</th>
                        </tr>
                    </thead>

                    <!-- TABLE BODY: Side-by-side feature comparison -->
                    <tbody>
                        <!-- PRICE COMPARISON -->
                        <tr>
                            <th scope="row" class="feature-name">Price per Person</th>
                            <td>$15-20</td>
                            <td class="popular-column">$25-35</td>
                            <td>$40-50</td>
                        </tr>

                        <!-- MAIN COURSE OPTIONS -->
                        <tr>
                            <th scope="row" class="feature-name">Main Course Options</th>
                            <td>3 options</td>
                            <td class="popular-column">5 options</td>
                            <td>Unlimited</td>
                        </tr>

                        <!-- SIDE DISHES -->
                        <tr>
                            <th scope="row" class="feature-name">Side Dishes</th>
                            <td>2 sides</td>
                            <td class="popular-column">4 sides</td>
                            <td>Full selection</td>
                        </tr>

                        <!-- APPETIZERS -->
                        <tr>
                            <th scope="row" class="feature-name">Appetizers</th>
                            <td>—</td>
                            <td class="popular-column">Selection</td>
                            <td>Premium selection</td>
                        </tr>

                        <!-- DESSERTS -->
                        <tr>
                            <th scope="row" class="feature-name">Desserts</th>
                            <td>—</td>
                            <td class="popular-column">Dessert bar</td>
                            <td>Dessert & coffee bar</td>
                        </tr>

                        <!-- SETUP SERVICE -->
                        <tr>
                            <th scope="row" class="feature-name">Setup</th>
                            <td>Basic</td>
                            <td class="popular-column">Premium</td>
                            <td>Full service</td>
                        </tr>

                        <!-- TABLEWARE -->
                        <tr>
                            <th scope="row" class="feature-name">Tableware</th>
                            <td>Disposable</td>
                            <td class="popular-column">Basic glassware</td>
                            <td>Premium Glassware</td>
                        </tr>

                        <!-- SERVICE STAFF -->
                        <tr>
                            <th scope="row" class="feature-name">Service Staff</th>
                            <td>—</td>
                            <td class="popular-column">—</td>
                            <td aria-label="Included">✓</td>
                        </tr>

                        <!-- CUSTOM MENU -->
                        <tr>
                            <th scope="row" class="feature-name">Custom Menu</th>
                            <td>—</td>
                            <td class="popular-column">—</td>
                            <td aria-label="Included">✓</td>
                        </tr>
                    </tbody>
                </table>
            </div>
        </div>
    </section>

    <!-- F8: CONTACT SECTION with catering team email -->
    <section class="catering-contact">
        <div class="catering-container">
            <h2>Custom Requests or Want to Cater?</h2>
            <p style="font-size: 1.3rem;">Contact our catering team directly at</p>

            <!-- EMAIL ADDRESS for bookings and custom requests -->
            <div class="contact-details">
                <div class="contact-item">
                    <span style="font-size: 1.8rem; font-weight: bold; color: #ff6b1a;">
                        catering@leckerhaus.com
                    </span>
                </div>
            </div>
        </div>
    </section>
</div>

<?php include 'includes/footer.php'; ?>
```

MODERN - Catering.jsx

```jsx
import './Catering.css';

const Catering = () => {
  return (
    <div className="catering-page">
      {/* F8: PACKAGE COMPARISON TABLE */}
      <section className="catering-form-section">
        <div className="catering-container">
          <div className="menu-header">
            <h1>Compare Packages</h1>
            <p>Compare our catering packages to find the perfect fit for your event</p>
          </div>

          {/* F8: THREE-TIER COMPARISON TABLE (JSX) */}
          <div className="table-wrapper">
            <table className="catering-table" role="table">
              <caption className="sr-only">
                Comparison of Essential, Premium and Deluxe catering packages
              </caption>

              {/* TABLE HEADER: Three package tiers */}
              <thead>
                <tr>
                  <th scope="col">Feature</th>
                  <th scope="col">Essential</th>
                  <th scope="col" className="popular-column">Premium</th>
                  <th scope="col">Deluxe</th>
                </tr>
              </thead>

              {/* TABLE BODY: Side-by-side feature comparison */}
              <tbody>
                {/* PRICE COMPARISON */}
                <tr>
                  <th scope="row" className="feature-name">Price per Person</th>
                  <td>$15-20</td>
                  <td className="popular-column">$25-35</td>
                  <td>$40-50</td>
                </tr>

                {/* MAIN COURSE OPTIONS */}
                <tr>
                  <th scope="row" className="feature-name">Main Course Options</th>
                  <td>3 options</td>
                  <td className="popular-column">5 options</td>
                  <td>Unlimited</td>
                </tr>

                {/* SIDE DISHES */}
                <tr>
                  <th scope="row" className="feature-name">Side Dishes</th>
                  <td>2 sides</td>
                  <td className="popular-column">4 sides</td>
                  <td>Full selection</td>
                </tr>

                {/* APPETIZERS */}
                <tr>
                  <th scope="row" className="feature-name">Appetizers</th>
                  <td>—</td>
                  <td className="popular-column">Selection</td>
                  <td>Premium selection</td>
                </tr>

                {/* DESSERTS */}
                <tr>
                  <th scope="row" className="feature-name">Desserts</th>
                  <td>—</td>
                  <td className="popular-column">Dessert bar</td>
                  <td>Dessert & coffee bar</td>
                </tr>

                {/* SETUP SERVICE */}
                <tr>
                  <th scope="row" className="feature-name">Setup</th>
                  <td>Basic</td>
                  <td className="popular-column">Premium</td>
                  <td>Full service</td>
                </tr>

                {/* TABLEWARE */}
                <tr>
                  <th scope="row" className="feature-name">Tableware</th>
                  <td>Disposable</td>
                  <td className="popular-column">Basic glassware</td>
                  <td>Premium Glassware</td>
                </tr>

                {/* SERVICE STAFF */}
                <tr>
                  <th scope="row" className="feature-name">Service Staff</th>
                  <td>—</td>
                  <td className="popular-column">—</td>
                  <td aria-label="Included">✓</td>
                </tr>

                {/* CUSTOM MENU */}
                <tr>
                  <th scope="row" className="feature-name">Custom Menu</th>
                  <td>—</td>
                  <td className="popular-column">—</td>
                  <td aria-label="Included">✓</td>
                </tr>
              </tbody>
            </table>
          </div>
        </div>
      </section>

      {/* F8: CONTACT SECTION */}
      <section className="catering-contact">
        <div className="catering-container">
          <h2>Request a Catering Quote</h2>
          <p style={{ fontSize: '1.1rem', marginBottom: '2rem' }}>
            Fill out the form below and we'll get back to you within 24 hours
          </p>

          {/* Contact info with email */}
          <div className="contact-info" style={{ marginTop: '2rem', textAlign: 'center' }}>
            <p style={{ fontSize: '1rem', color: '#666' }}>
              Or contact us directly at:
            </p>
            <p style={{ fontSize: '1.1rem', fontWeight: 'bold', color: '#ff6b1a' }}>
              catering@leckerhaus.com | +65 6748 9380
            </p>
          </div>
        </div>
      </section>
    </div>
  );
};

export default Catering;
```

F9: Email Processing

Modern
React form for submitting catering information. Client-side validation for phone number and email fields. Uses PHP mail() function on backend, email interfaced locally via Thunderbird.

MODERN - Catering.jsx

```jsx
import { useState } from 'react';

const Catering = () => {
  const [formData, setFormData] = useState({
    name: '', email: '', phone: '', eventDate: '',
    guestCount: '', package: '', message: ''
  });
  const [loading, setLoading] = useState(false);

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  // F9: Submit inquiry via API
  const handleSubmit = async (e) => {
    e.preventDefault();
    setLoading(true);

    try {
      const response = await fetch('.../backend/api/catering.php', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(formData)
      });

      const data = await response.json();
      if (response.ok) {
        alert(data.message);
        setFormData({ name: '', email: '', phone: '', eventDate: '',
                      guestCount: '', package: '', message: '' });
      } else {
        alert(data.error);
      }
    } catch (err) {
      alert('Failed to send inquiry');
    } finally {
      setLoading(false);
    }
  };

  return (
    <section className="catering-contact">
      <h2>Request a Catering Quote</h2>

      <form onSubmit={handleSubmit}>
        <input type="text" name="name" value={formData.name}
               onChange={handleChange} placeholder="Your Name" required />

        <input type="email" name="email" value={formData.email}
               onChange={handleChange} placeholder="Email" required />

        <input type="tel" name="phone" value={formData.phone}
               onChange={handleChange} placeholder="67489380" required />

        <input type="date" name="eventDate" value={formData.eventDate}
               onChange={handleChange} required />

        <input type="number" name="guestCount" value={formData.guestCount}
               onChange={handleChange} placeholder="Number of Guests" required />

        <select name="package" value={formData.package} onChange={handleChange} required>
          <option value="">Select Package</option>
          <option value="Essential">Essential ($15-20)</option>
          <option value="Premium">Premium ($25-35)</option>
          <option value="Deluxe">Deluxe ($40-50)</option>
        </select>

        <textarea name="message" value={formData.message}
                  onChange={handleChange} placeholder="Special requests..." />

        <button type="submit" disabled={loading}>
          {loading ? 'Sending...' : 'Send Inquiry'}
        </button>
      </form>

      <p>Or email: <strong>f32ee@localhost</strong></p>
    </section>
  );
};

export default Catering;
```

MODERN - backend/api/catering.php

```php
<?php

require_once __DIR__ . '/../config/cors.php';
header('Content-Type: application/json');

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Parse JSON from React
    $input = json_decode(file_get_contents('php://input'), true);

    $name = $input['name'] ?? '';
    $email = $input['email'] ?? '';
    $phone = $input['phone'] ?? '';
    $eventDate = $input['eventDate'] ?? '';
    $guestCount = $input['guestCount'] ?? '';
    $package = $input['package'] ?? '';
    $message = $input['message'] ?? '';

    // VALIDATION
    if (empty($name) || empty($email) || empty($phone) ||
        empty($eventDate) || empty($guestCount) || empty($package)) {
        http_response_code(400);
        echo json_encode(['error' => 'Please fill in all required fields']);
        exit;
    }

    // Prepare email
    $to = 'f32ee@localhost';
    $subject = 'New Catering Inquiry - ' . $package;

    $body = "New Catering Inquiry\n\n";
    $body .= "Contact: $name ($email, $phone)\n";
    $body .= "Event: $eventDate | Guests: $guestCount\n";
    $body .= "Package: $package\n\n";
    $body .= "Message: $message\n";

    $headers = "From: $email\r\nReply-To: $email\r\n";

    // Send email using PHP mail() function
    if (mail($to, $subject, $body, $headers)) {
        echo json_encode([
            'success' => true,
            'message' => 'Your inquiry has been sent! We will contact you soon.'
        ]);
    } else {
        http_response_code(500);
        echo json_encode(['error' => 'Failed to send email']);
    }
} else {
    http_response_code(405);
    echo json_encode(['error' => 'Method not allowed']);
}
?>
```

Table A2.1. Contribution to the Project by each team member

Note: create more rows if needed, but DO NOT change the column size of “Remarks by Tutor”

Functional Requirement	Filename(s) and line numbers created for this requirement	Remarks by Tutor	Contributed by which member
F1. User Authentication	**Traditional:** login.php (1-101), process-login.php (1-138), process-logout.php (1-10), validation.js (4-26 and 55-73), config.php (1-16) **Modern:** login.php (1-55), register.php (9-110), logout.php (1-29), check-session.php (1-40), api.js (31-59)		Jun Le
F2. Menu browsing and ordering	**Traditional:** menu.php (1-160), process-add-to-cart.php (1-70) **Modern:** menu_items.php (6-154), Menu,jsx (42-225), api.js (61-76) and (119-124)		Yin Yu

F3. Shopping Cart	**Traditional:** cart.php (1-98), process-update-cart.php (1-52), process-remove-from-cart.php (1-38) **Modern:** cart.php (13-148), api.js (104-108), Cart.jsx (22-360)		Yin Yu
F4. Checkout and order placement F3. xxx	**Traditional:** cart.php (100-150), process-checkout.php(1-119), order-confirmation.php (1-121) **Modern:** orders.php (87-164), cart.php (162-204), api.js (200-207), Checkout.jsx (74-218)	Jun	Jun Le
F5. Order status tracking	**Traditional:** order-history.php (1-119), process-advance-order.php (1-66)		Jun Le

	Modern: orders.php (56-85), OrderStatus.jsx (1-138)		
F6. Order History Viewing	**Traditional:** order-history.php (1-119) **Modern:** order history.php (1-73 and 316-396)		Yin Yu
F7. Customer Feedback Submission	**Traditional:** feedback.php (1-167), process-feedback.php (1-52) **Modern:** feedback.php (6-135), database.php (1-30), cors.php (1-19), api.js (167-180), Feedback.jsx (1-269)		Yin Yu
F8. Catering Package	**Traditional:** catering.php (1-116) **Modern:** Catering.jsx (68-147)		Jun Le

F9. Email for catering (localhost)	**Modern:** catering.php (8-70) Catering.jsx (4-66 and 149-303)		Jun Le

Table A2.2. Database Tables created

Table Name	Purpose/Usage of this Table	Remarks by Tutor	Contributed by which member
users	Purpose: User Authentication and profile		Yin Yu / Jun Le

	management. Usage: - Stores user account information (username, email, hashed password, full name) - Used in login.php for authentication - Used in register.php for account creation - Session management via check-session.php - Linked to orders, feedbacks and cart_items tables via user_id foreign key		
menu_items	Purpose: Restaurant menu catalog. Usage: - Stores all menu items with pricing, descriptions, categories, and ratings - Displayed on menu page via menu_items.php - Supports filtering by menu category - Linked to cart_items and order_items tables via menu_item_id foreign key		Yin Yu / Jun Le
cart_items	Purpose: Shopping cart persistence for checkout process. Usage:		Yin Yu / Jun Le

	- Stores items added to cart before checkout - Used for checking authenticated user for access to cart functionality - Updated via cart.php API endpoint - Cleared after successful order placement - Used in Menu.jsx when adding items - Linked to menu_items and users tables via id primary key		
feedbacks	Purpose: Customer review and rating system with control. Usage: - Stores customer feedback with ratings - Requires admin approval before public display ('approved' flag) - Used in feedback.php - Displayed on feedback page via Feedback.jsx - Linked to users via id primary key		Yin Yu / Jun Le
orders	Purpose: Order transaction history and tracking. Usage: - Records all completed orders with customer details		Yin Yu / Jun Le

	- Tracks order status progression - Used in orders.php - Linked to order_items table via order_id foreign key, linked to users table via id primary key		
order_items	Purpose: Order details for each order. Usage: - Stores individual menu items within each order - Records quantity and price at time of purchase - Created when order is placed from cart - Used to display order details and receipts - Links to both orders and menu_items tables via id primary key		Yin Yu / Jun Le

Table A3. Contribution to the Report by each team member

Chapter	Contributed by which member
Chapter 1 to 3	Jun Le
Chapter 4 to 6	Yin Yu
Appendix 1 to 2	Yin Yu / Jun Le

If possible, state Percentage Contribution (out of 100%) by each member to the project and report. If it is equal contribution, state 50% for each member:

Member 1 (Name: Tan Yin Yu) - Percentage: 50%
Member 2 (Name: Liw Jun Le) - Percentage: 50%