

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. Sessions (`$_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 `htmlspecialchars()` 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 separate 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 straightforward. 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.

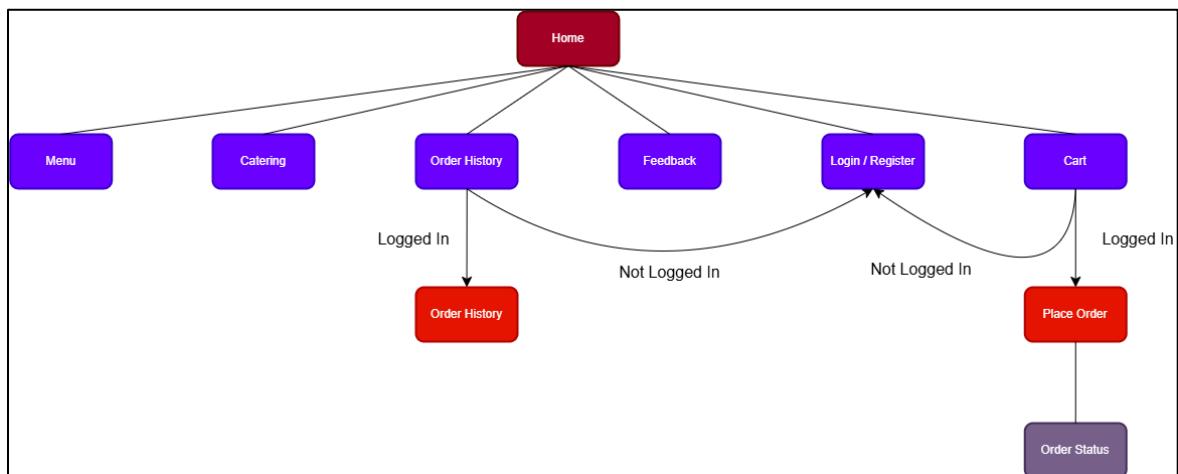


Figure 1. Sitemap

Wireframe



Figure 2. Overview of general wireframing

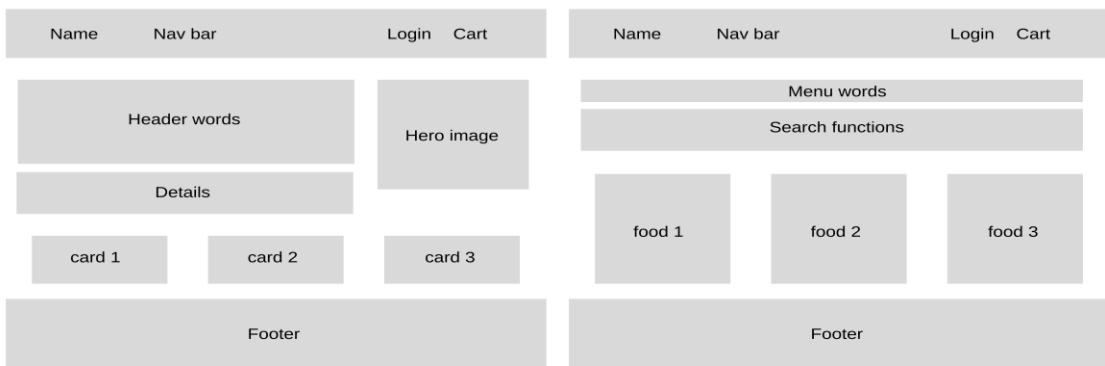


Figure 3. Left: Main page, right: Menu page

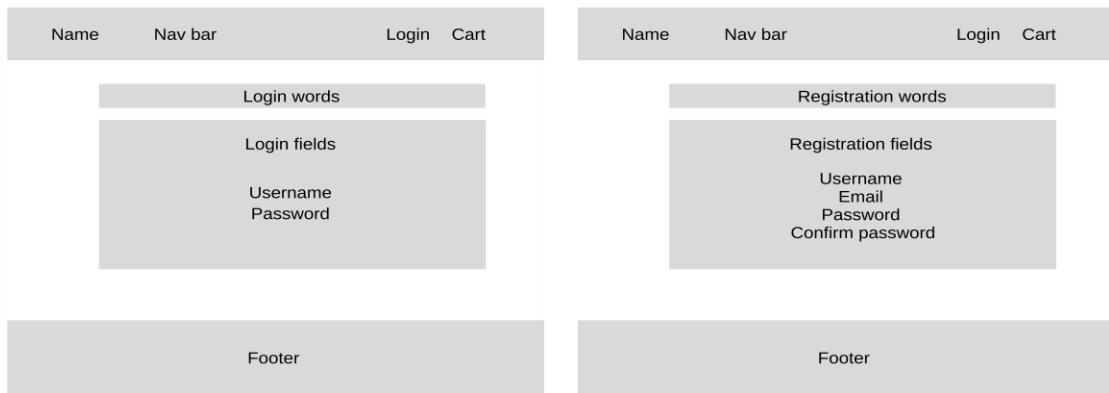


Figure 4. Left: Login page, right: Registration page

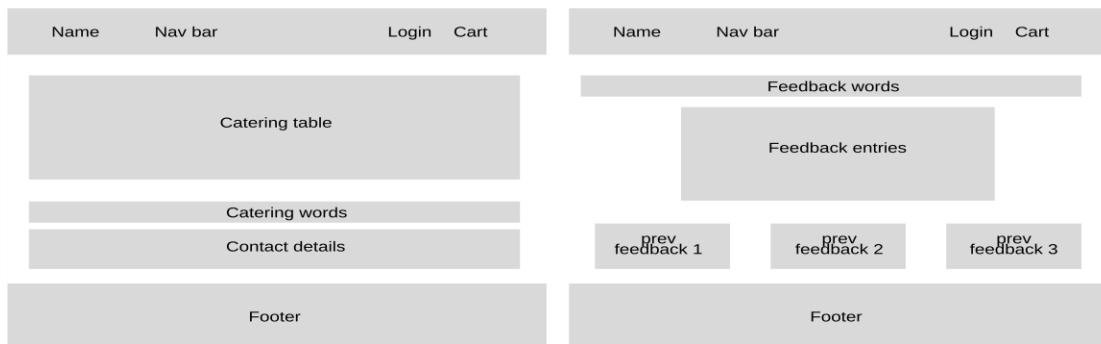


Figure 5. Left: Catering page, right: Feedback page

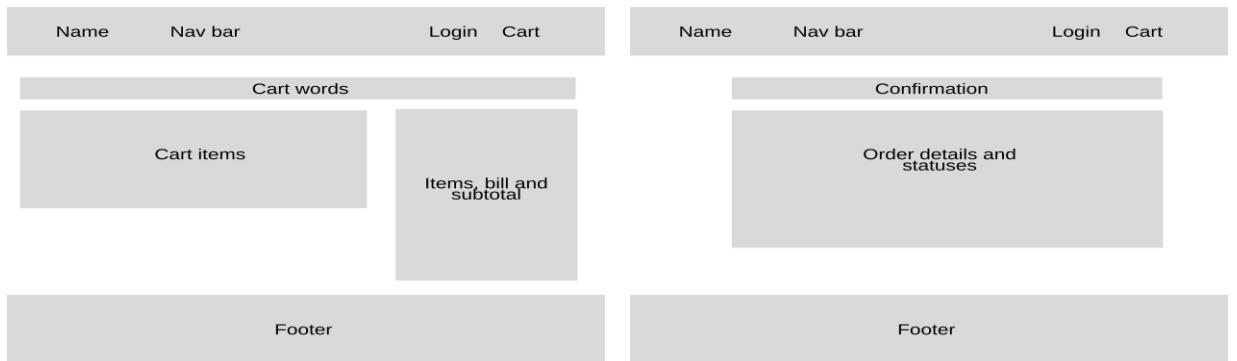


Figure 6. Left: Cart page, right: Confirmation page

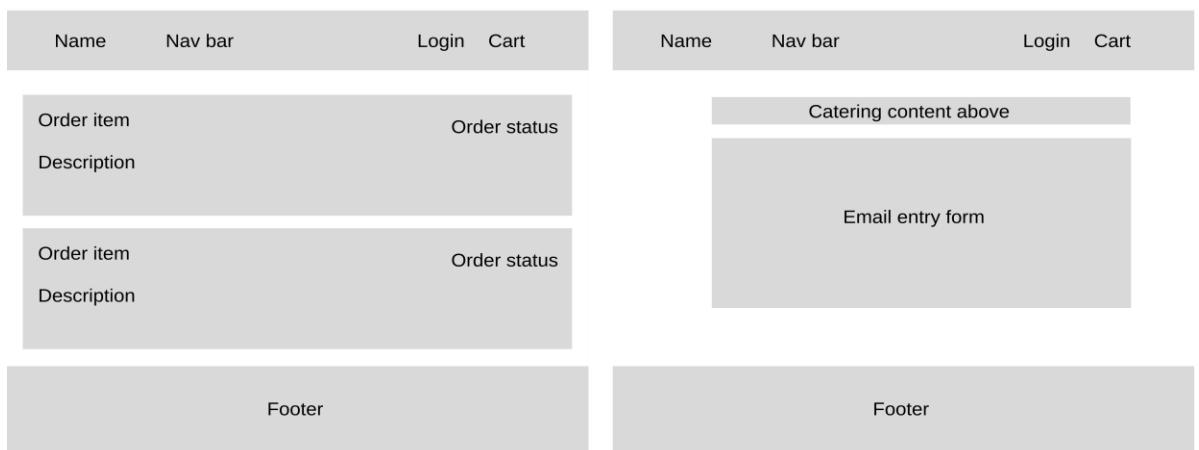


Figure 7. Left: Order history page, right: Email functionality within catering page

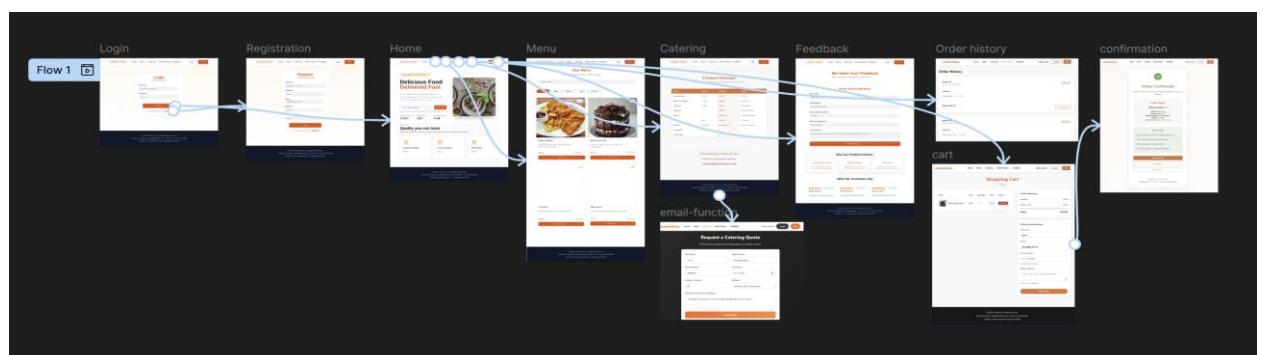
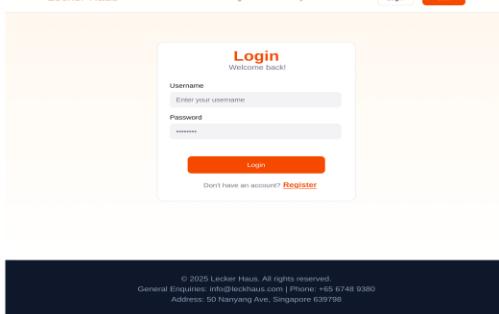
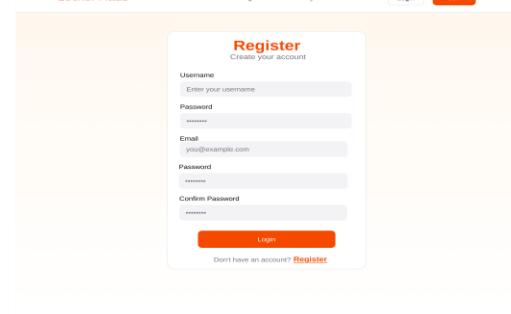


Figure 8. Overview of in-depth wireframing

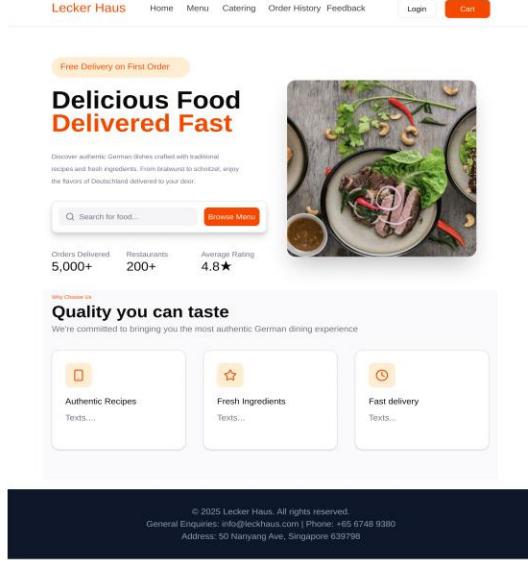


The login page features a central 'Login' form with fields for 'Username' and 'Password'. Below the form is a link 'Don't have an account? Register'.

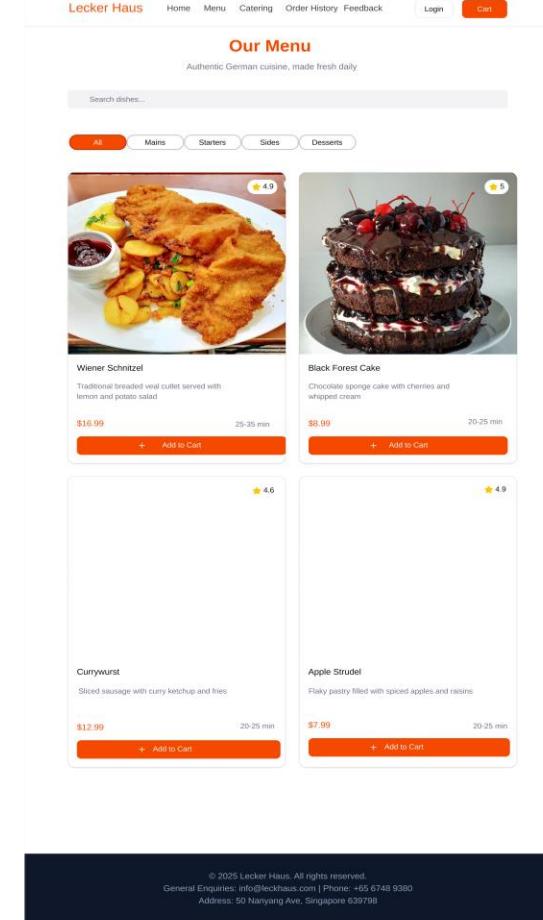


The registration page features a central 'Register' form with fields for 'Username', 'Password', 'Email', and 'Confirm Password'. Below the form is a link 'Don't have an account? Login'.

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



The main page highlights 'Free Delivery on First Order' and 'Delicious Food Delivered Fast'. It shows statistics: 5,000+ orders delivered, 200+ restaurants, and an average rating of 4.8★. A search bar and a 'Browse Menu' button are also present.



The menu page is titled 'Our Menu' and describes 'Authentic German cuisine, made fresh daily'. It includes a search bar and category filters for All, Mains, Starters, Sides, and Desserts. The menu lists several items with images, ratings, and descriptions:

- Wiener Schnitzel**: Traditional breaded veal cutlet served with lemon and potato salad. Rating: 4.9. Price: \$16.99. Delivery time: 25-35 min. Add to Cart button.
- Black Forest Cake**: Chocolate sponge cake with cherries and whipped cream. Rating: 5.0. Price: \$8.99. Delivery time: 20-25 min. Add to Cart button.
- Currywurst**: Sliced sausage with curry ketchup and fries. Rating: 4.6. Price: \$12.99. Delivery time: 20-25 min. Add to Cart button.
- Apple Strudel**: Flaky pastry filled with spiced apples and raisins. Rating: 4.9. Price: \$7.99. Delivery time: 20-25 min. Add to Cart button.

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

Compare Packages

Compare our catering packages to find the perfect fit for your event.

Feature	Essential	Premium	Pro	Luxury
Price per Person	\$15-20	\$25-35	\$40-50	
Main Course Options	3 options	5 options	Unlimited	
Side Dishes	2 sides	4 sides	Full selection	
Appetizers	—	Selection	Premium selection	
Desserts	—	Dessert bar	Dessert & coffee bar	
Setup	Basic	Premium	Full service	
Tableware	Disposable	Basic glassware	Premium Glassware	
Service Staff	—	—	✓	
Custom Menu	—	—	✓	

Custom Requests or Want to Cater?

Contact our catering team directly at
catering@leckerhaus.com

© 2025 Lecker Haus. All rights reserved.
General Enquiries: info@leckerhaus.com | Phone: +65 6748 9380
Address: 50 Nanyang Ave, Singapore 639798

We Value Your Feedback

Help us improve by sharing your experience

Share Your Experience

Your Name *
John Doe

Email Address *
john@example.com

Order Number (Optional)
#12345

Rate Your Experience *
Select Rating

Your Feedback *
Tell us about your experience with our service, food quality, delivery time, etc...

Why Your Feedback Matters

Improve Our Service
Your insights help us enhance our food quality and service standards.

Shape Our Menu
Suggestions from customers like you influence our menu development.

Build Trust
We value transparency and use feedback to maintain high standards.

What Our Customers Say

Sarah Johnson · 2 days ago
★★★★★
Absolutely amazing service!

Sarah Johnson · 2 days ago
★★★★★
Absolutely amazing service!

Sarah Johnson · 2 days ago
★★★★★
Absolutely amazing service!

© 2025 Lecker Haus. All rights reserved.
General Enquiries: info@leckerhaus.com | Phone: +65 6748 9380
Address: 50 Nanyang Ave, Singapore 639798

Request a Catering Quote

Fill out the form below and we'll get back to you within 24 hours

Your Name *
Jun Le

Email Address *
f31ee@localhost

Phone Number *
88469676

Event Date *
26/11/2025

Number of Guests *
99

Package *
Premium (\$25-35 per person)

Additional Information (Optional)
I would like to arrange for a German buffet catering event for my company.

Send Inquiry

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

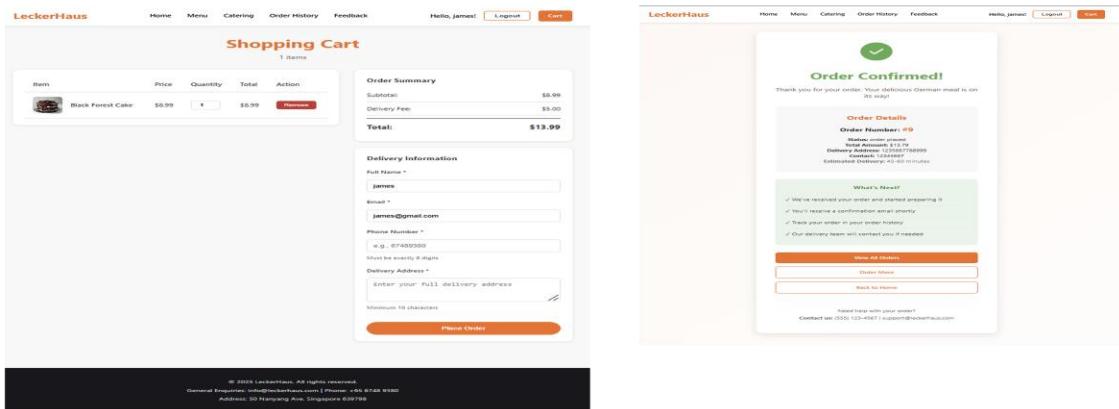


Figure 12. Left: Cart page, right: Confirmation page

Order History

Order #7
Nov 12, 2020 12:00 AM
Items (1):
Black forest cake × 1 - \$13.99
Total: \$13.99

Order #6
Nov 12, 2020 12:15 AM
Items (1):
Black Forest Cake × 1 - \$13.99

Cart

- Cart updated
- Item removed from cart
- Added to cart Apple Strudel

Status

- Preparing
- Out For Delivery
- Delivered
- Order Placed

Validation

- Username must be at least 3 characters
- Passwords do not match
- Password must contain at least one special character (!@#\$%^&*()_!<>)
- Password must contain at least one number
- Password must contain at least one letter
- Password must be at least 8 characters long
- Username or email already registered

Header

Lecker Haus Home Menu Catering Order History Feedback Login Cart

Footer

Option 1:

C 2020 Lecker Haus. All rights reserved.
General Enquiries: info@leckerhaus.com | Phone: +65 6789 5678
Address: 50 Nanyang Ave, Singapore 091950

Option 2:

Lecker Haus Delivery fast delivered to your doorstep

Quick Links	Contact Info
Home	General Enquiries: info@leckerhaus.com
Menu	Phone: +65 6789 5678
Catering	Address: 50 Nanyang Ave, Singapore 091950
Feedback	© 2020 Lecker Haus. All rights reserved.

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.

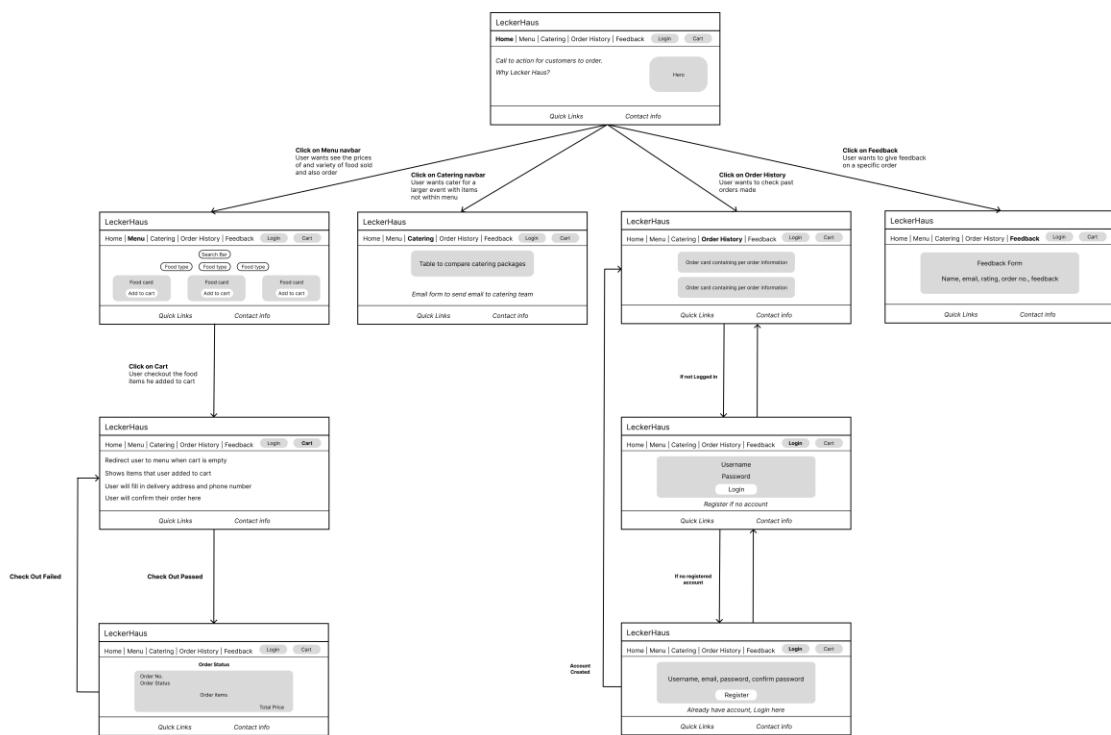


Figure 14. Story Board design

Database

The web application uses a MYSQL relational database with six tables: users, menu_items, cart_items, orders, orders_items and feedbacks. 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.

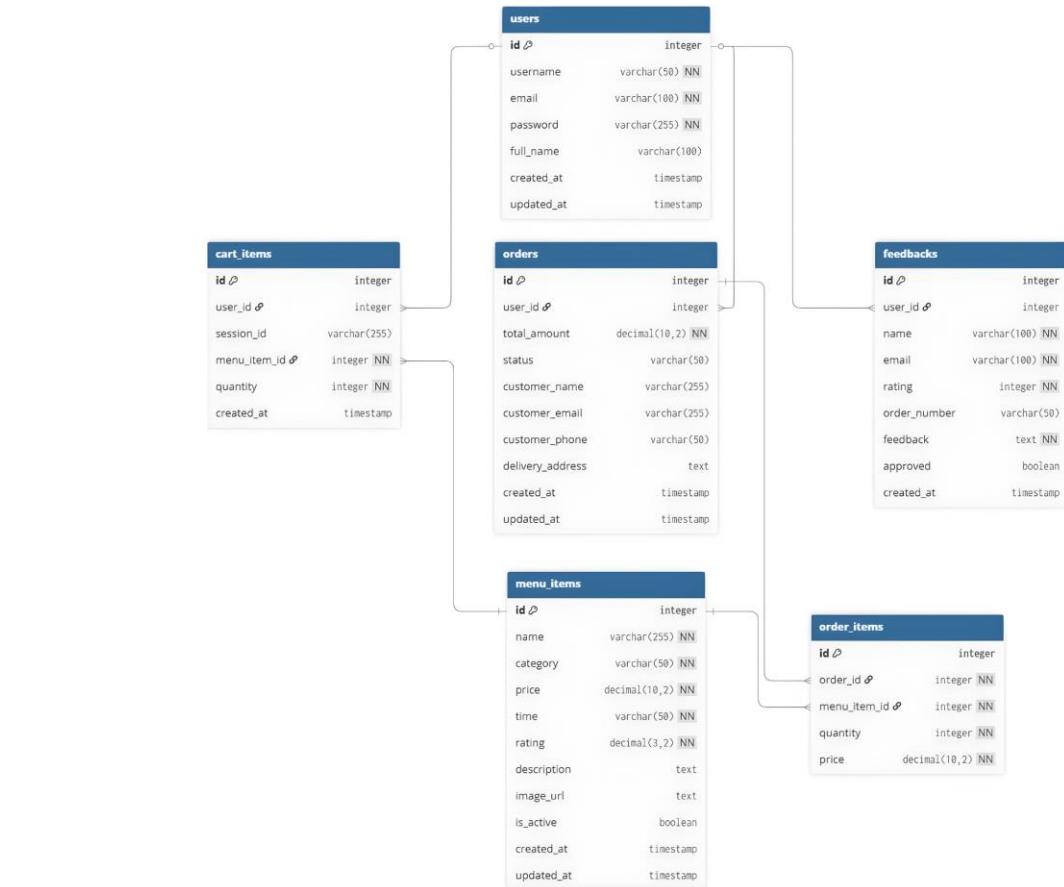


Figure 15. Database design

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.

The figure consists of seven rectangular boxes arranged in a grid-like pattern, each containing a different error message related to authentication fields:

- Top Left:** "Username or email already exists"
- Top Right:** "Passwords do not match"
- Second Row, Left:** "Username" field with placeholder "Enter your username". Below it: "Email" field with placeholder "w@". Error message: "Please fill out this field." and "Please enter a part following '@'. 'w@' is incomplete."
- Second Row, Right:** "Password" field with placeholder ". ". Error message: "Please lengthen this text to 8 characters or more (you are currently using 1 character)."
- Third Row, Left:** "w" in an "Email" field. Error message: "Please include an '@' in the email address. 'w' is missing an '@'."
- Third Row, Right:** "Password must contain at least one number"
- Bottom Right:** "Password must contain at least one special character (!@#\$%^&*(),.?:>)"

Figure 16. F1 - Possible cases of failed authentications

Our Menu

Authentic German cuisine, made fresh daily

Search dishes... Sort by: Name

All Mains Starters Sides Desserts

Showing 12 dishes

LeckerHaus Home Menu Catering Order History Feedback Hello, James! Logout Cart

Our Menu

Authentic German cuisine, made fresh daily

Search dishes... Sort by: Name

All Mains Starters Sides Desserts

Showing 6 dishes

LeckerHaus Home Menu Catering Order History Feedback Hello, James! Logout Cart

Our Menu

Authentic German cuisine, made fresh daily

Search dishes... Sort by: Rating

All Mains Starters Sides Desserts

Showing 12 dishes

DESSERTS MAINS DESSERTS

Black Forest Cake Wiener Schnitzel Apple Strudel

LeckerHaus Home Menu Catering Order History Feedback Hello, James! Logout Cart

Our Menu

Authentic German cuisine, made fresh daily

Search dishes... Sort by: Price: Low to High

All Mains Starters Sides Desserts

Showing 12 dishes

STARTERS SIDES DESSERTS

Krautsalat Bratkartoffeln Apple Strudel

Traditional German coleslaw with caraway seeds Pan-fried potatoes with onions and herbs Faky pastry filled with spiced apples and raisins

\$6.99 \$7.49 \$7.99

LeckerHaus Home Menu Catering Order History Feedback Hello, James! Logout Cart

Our Menu

Authentic German cuisine, made fresh daily

wurst

All Mains Starters Sides Desserts

Showing 2 dishes

MAIN MAINS

Bratwurst Platter Currywurst

LeckerHaus Home Menu Catering Order History Feedback Hello, James! Logout Cart

Our Menu

Authentic German cuisine, made fresh daily

meat

All Mains Starters Sides Desserts

Showing 1 dish

STARTERS

Königsberger Klopse

Traditional meatballs in creamy caper sauce

4.7

LeckerHaus Home Menu Catering Order History Feedback Hello, James! Logout Cart

Our Menu

Authentic German cuisine, made fresh daily

Added to cart: Apple Strudel

Search dishes... Sort by: Name

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

Delivery Information

Full Name *
James

Email *
James@gmail.com

Phone Number *
12345678
Must be exactly 8 digits

Delivery Address *
123 Main St, #01-23, Singapore 123456
Minimum 10 characters

Place Order

Continue Shopping

Your Cart

Your cart is empty
Add some delicious items from our menu!
Browse Menu

Your Cart

Order Summary

Subtotal (2 items): \$16.98
Delivery Fee: \$5.00
Total: \$21.98

Delivery Information

Figure 18. F3 – Cart page scenarios and functions

Order Confirmed!
Thank you for your order. Your delicious German meal is on its way!

Order Details

Order Number: #4
Status: order placed
Total Amount: \$22.03
Delivery Address: 123 Main St, #01-23, Singapore 123456
Contact: 12345667
Estimated Delivery: 45-60 minutes

What's Next?

- We've received your order and started preparing it
- You'll receive a confirmation email shortly
- Track your order in your order history
- Our delivery team will contact you if needed

View All Orders

Order More

Back to Home

Delivery Information

Full Name *
John Doe
Name is required

Email *
john@example.com
Email is required

Phone Number *
67489380
Phone number must be exactly 8 digits (e.g., 67489380)

Delivery Address *
123 Main St, #01-23, Singapore 123456
Minimum 10 characters
Address must be at least 10 characters

Place Order

Continue Shopping

Your Cart

Bratwurst Platter MAINS \$14.99 Remove

Currywurst MAINS \$12.99 Remove

German Potato Salad SIDES \$8.49 Remove

Krautsalat STARTERS \$6.99 Remove

Figure 19. F4 – Checkout and order placement

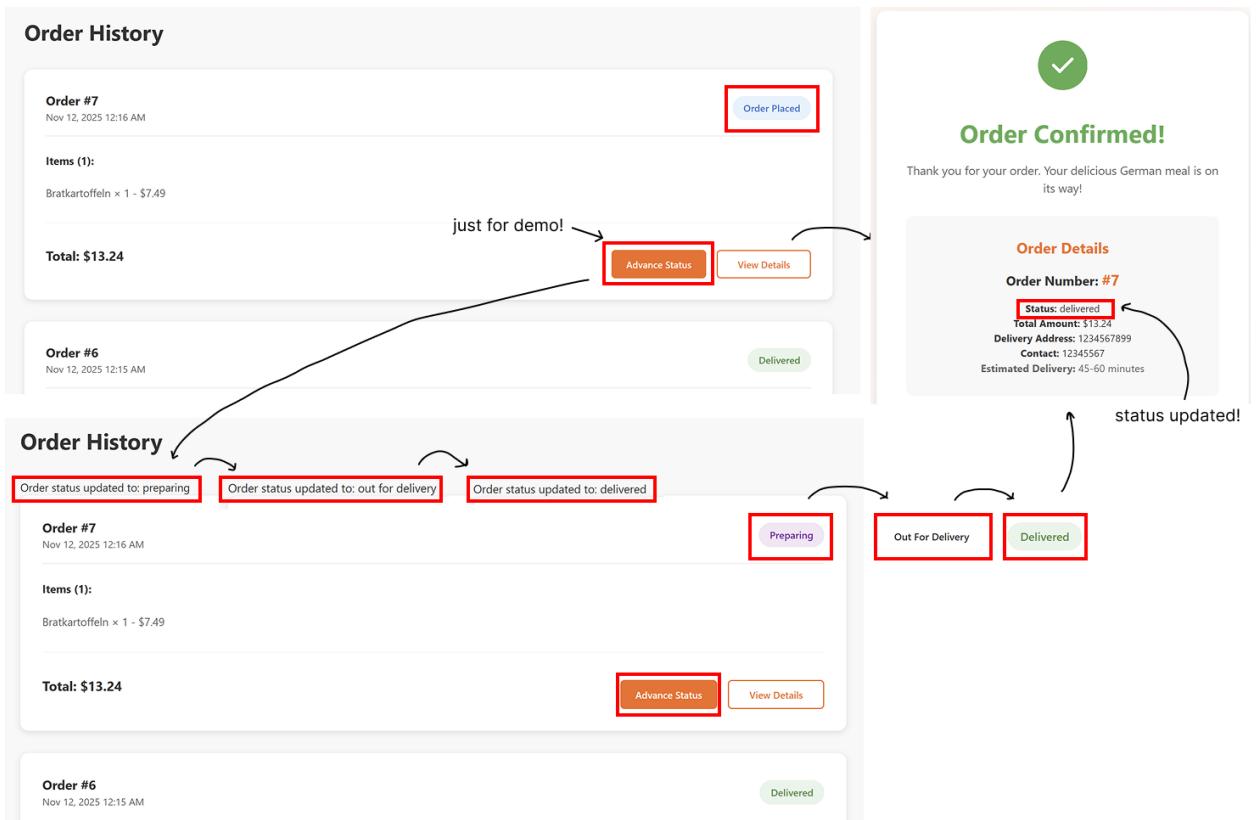


Figure 20. F5 – Order status tracking

The screenshot shows the LeckerHaus website interface with two main sections:

- Order History:** A user profile for "Hello, James!" shows a list of two orders (Order #7 and Order #6) with their details and status ("Delivered").
- Database:** A table listing all orders from the user "James" (user_id 2). The columns include id, user_id, total_amount, status, customer_name, and customer_email. Each row has a "Delete" link.
- Order History (Test User):** A user profile for "Hello, testuser!" shows a message stating "No orders are reflected since none are placed under testuser".

A curved arrow points from the "Hello, James!" profile to the database table, indicating the source of the data.

	ID	User ID	Total Amount	Status	Customer Name	Customer Email
1	1	2	24.78	delivered	james	james@gr...
2	2	2	67.93	delivered	James	James@g...
3	3	2	21.48	delivered	james	james@gr...
4	4	2	22.03	delivered	james	james@gr...
5	5	2	13.24	delivered	james	james@gr...
6	6	2	14.89	delivered	james	james@gr...
7	7	2	13.24	delivered	james	james@gr...

Figure 21. F6 – Order History Viewing

Validation logics

We Value Your Feedback

Help us improve by sharing your experience

Share Your Experience

Your Name *

Email *

Overall Rating *

Select rating

5 Stars - Excellent

4 Stars - Good

3 Stars - Average

2 Stars - Below Average

1 Star - Poor

Your Feedback *

Please fill out this field.

Submit Feedback

Thank you for your feedback! It will be reviewed before being published.

What Our Customers Say

	id	user_id	name	email	rating	order_number	feedback	approved	created_at
Delete	1	1	Test User	test@example.com	5	ORD001	Amazing food! The Wiener Schnitzel was perfectly c...	1	2025-11-1
Delete	2	NULL	Jane Smith	jane@example.com	5	ORD002	Best German food in town! Will definitely order ag...	1	2025-11-1
Delete	3	NULL	Mike Johnson	mike@example.com	4	ORD003	Great flavors and generous portions. Delivery was ...	1	2025-11-1
Delete	4	2	james	james@gmail.com	5	123	Good.	1	2025-11-1
Delete	5	2	james	james@gmail.com	5	#12345	Great!	1	2025-11-1

only 3 approved

James' reviews approved, only then they appear

Figure 22. F7 – Customer Feedback Submission

LeckerHaus

Home Menu Catering Order History Feedback Hello, testuser2! Logout Cart

Compare Packages

Compare our catering packages to find the perfect fit for your event

Feature	Essential	Premium	Popular	Deluxe
Price per Person	\$15–20	\$25–35	\$40–50	
Main Course Options	3 options	5 options	Unlimited	
Side Dishes	2 sides	4 sides	Full selection	
Appetizers	—	Selection	Premium selection	
Desserts	—	Dessert bar	Dessert & coffee bar	
Setup	Basic	Premium	Full service	
Tableware	Disposable	Basic glassware	Premium Glassware	
Service Staff	—	—	✓	
Custom Menu	—	—	✓	

Custom Requests or Want to Cater?

Contact our catering team directly at

catering@leckerhaus.com

Figure 23. F8 – Catering Page Table

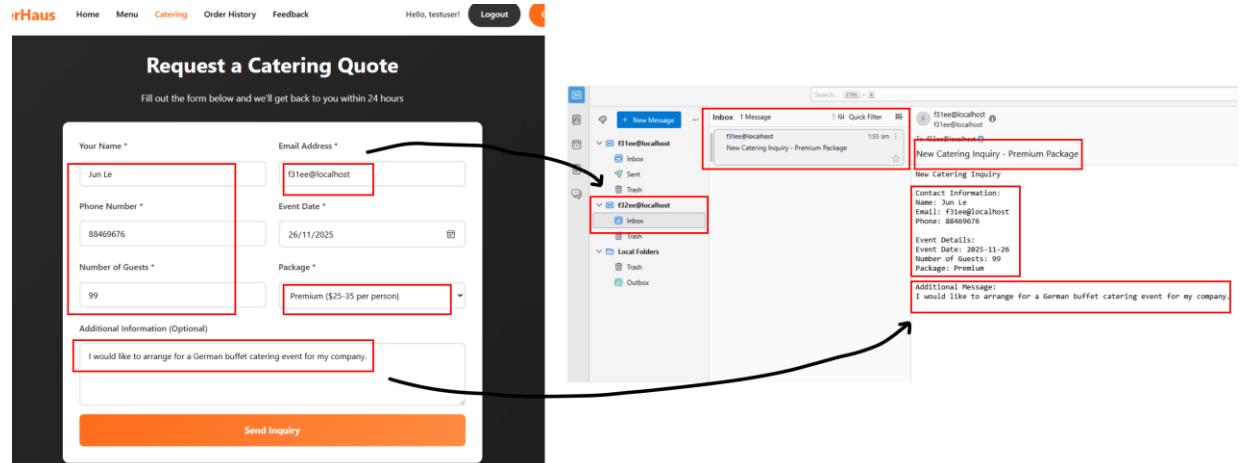


Figure 24. F9 – Catering Form Submission and receiving of email

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

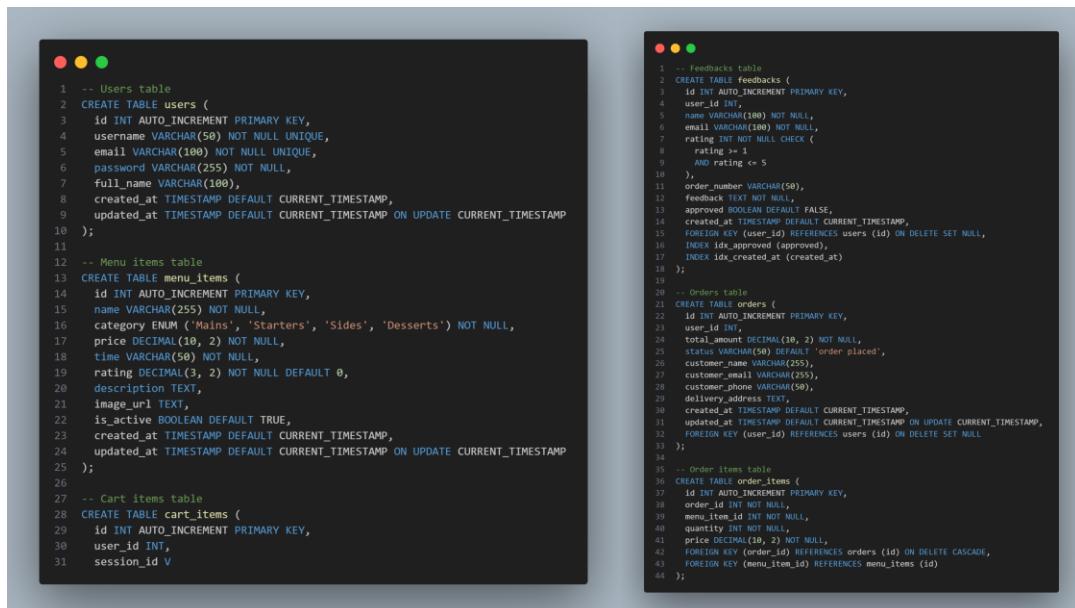
Appendix 2: Contribution to the project by each team member shown in a table (see Table A2.1 and A2.2).

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.



```
1 -- Users table
2 CREATE TABLE users (
3   id INT AUTO_INCREMENT PRIMARY KEY,
4   username VARCHAR(50) NOT NULL UNIQUE,
5   email VARCHAR(100) NOT NULL UNIQUE,
6   password VARCHAR(255) NOT NULL,
7   full_name VARCHAR(100),
8   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
9   updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
10 );
11
12 -- Menu items table
13 CREATE TABLE menu_items (
14   id INT AUTO_INCREMENT PRIMARY KEY,
15   name VARCHAR(255) NOT NULL,
16   category ENUM ('Mains', 'Starters', 'Sides', 'Desserts') NOT NULL,
17   price DECIMAL(10, 2) NOT NULL,
18   time VARCHAR(50) NOT NULL,
19   rating DECIMAL(3, 2) NOT NULL DEFAULT 0,
20   description TEXT,
21   image_url TEXT,
22   is_active BOOLEAN DEFAULT TRUE,
23   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
24   updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
25 );
26
27 -- Cart items table
28 CREATE TABLE cart_items (
29   id INT AUTO_INCREMENT PRIMARY KEY,
30   user_id INT,
31   session_id V
```



```
1 -- Feedbacks table
2 CREATE TABLE feedbacks (
3   id INT AUTO_INCREMENT PRIMARY KEY,
4   user_id INT,
5   name VARCHAR(100) NOT NULL,
6   email VARCHAR(100) NOT NULL,
7   rating INT NOT NULL CHECK (
8     rating >= 1
9     AND rating <= 5
10   ),
11   order_number VARCHAR(50),
12   feedback_text NOT NULL,
13   approved BOOLEAN DEFAULT FALSE,
14   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
15   FOREIGN KEY (user_id) REFERENCES users (id) ON DELETE SET NULL,
16   INDEX idx_approved (approved),
17   INDEX idx_created_at (created_at)
18 );
19
20
21 -- Orders table
22 CREATE TABLE orders (
23   id INT AUTO_INCREMENT PRIMARY KEY,
24   user_id INT,
25   total_amount DECIMAL(10, 2) NOT NULL,
26   customer_name VARCHAR(255),
27   customer_email VARCHAR(255),
28   customer_phone VARCHAR(50),
29   delivery_address TEXT,
30   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
31   updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
32   FOREIGN KEY (user_id) REFERENCES users (id) ON DELETE SET NULL,
33 );
34
35 -- Order items table
36 CREATE TABLE order_items (
37   id INT AUTO_INCREMENT PRIMARY KEY,
38   order_id INT NOT NULL,
39   menu_item_id INT NOT NULL,
40   quantity INT NOT NULL,
41   price DECIMAL(10, 2) NOT NULL,
42   FOREIGN KEY (order_id) REFERENCES orders (id) ON DELETE CASCADE,
43   FOREIGN KEY (menu_item_id) REFERENCES menu_items (id)
44 );
```

Database Configuration

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

TRADITIONAL

```
1 <?php
2 // Database connection parameters
3 define('DB_HOST', 'localhost');      // MySQL server
4 define('DB_USER', 'root');           // XAMPP default user
5 define('DB_PASS', '');              // Empty password for XAMPP
6 define('DB_NAME', 'food_web_app');   // Database name
7
8 // Application settings
9 define('SITE_URL', 'http://localhost/Food-Web-App_IE4727/Food-web-app-php');
10 define('SITE_NAME', 'LeckerHaus');
11
12 // Auto-start session for all pages
13 if (session_status() === PHP_SESSION_NONE) {
14     session_start();
15 }
16 ?>
```

MODERN

```
1 <?php
2 class Database {
3     private $host = 'localhost';
4     private $db_name = 'food_web_app';
5     private $username = 'root';
6     private $password = '';
7     private $conn;
8
9     // Returns PDO connection object
10    public function getConnection() {
11        $this->conn = null;
12
13        try {
14            $this->conn = new PDO(
15                "mysql:host=" . $this->host . ";dbname=" . $this->db_name,
16                $this->username,
17                $this->password
18            );
19            $this->conn->exec("set names utf8");
20            $this->conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
21        } catch(PDOException $exception) {
22            echo "Connection error: " . $exception->getMessage();
23        }
24
25        return $this->conn;
26    }
27 }
28 ?>
```

F1: Authentication

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

The image shows a terminal window with two panes. The left pane, titled 'TRADITIONAL', contains PHP code for a login script. The right pane, titled 'MODERN', contains equivalent logic implemented using React Context and asynchronous fetch requests.

```

TRADITIONAL
1 <?php
2 require_once 'includes/config.php';
3 require_once 'includes/db.php';
4
5 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
6     // Validate email format
7     $email = filter_var($_POST['email'], FILTER_VALIDATE_EMAIL);
8     $password = $_POST['password'];
9
10    // Password pattern: 8+ chars, 1 letter, 1 number, 1 special char
11    $pattern = '/^(?=.*[A-Za-Z])(?=.*\d)(?=.*[$@!%#&])[A-Za-Z\d$@$!%#&&][0-9]$/';
12
13    if (!empty($email)) {
14        $error = "Invalid email format";
15    } elseif (!preg_match($pattern, $password)) {
16        $error = "Password must be 8+ chars with letter, number, special char";
17    } else {
18        // Check database for user
19        $stmt = $pdo->prepare("SELECT * FROM users WHERE email = ?");
20        $stmt->execute([$email]);
21        $user = $stmt->fetch();
22
23        if ($user && password_verify($password, $user['password'])) {
24            // User can now log in to session
25            $_SESSION['user_id'] = $user['id'];
26            $_SESSION['username'] = $user['username'];
27            header('Location: index.php');
28            exit;
29        } else {
30            $error = "Invalid email or password";
31        }
32    }
33 }
34 >
35
36 <!-- HTML Form -->
37 <form method="POST" action="login.php">
38     <input type="email" name="email" placeholder="Email" required>
39     <input type="password" name="password" placeholder="Password" required>
40     <?php if (isset($error)) echo "<p class='error'>$error</p>"; ?>
41     <button type="submit">Login</button>
42 </form>

```

```

MODERN
1 import { createContext, useState, useEffect } from 'react';
2
3 export const AuthContext = createContext(null);
4
5 export const AuthProvider = ({ children }) => {
6     const [isAuthenticated, setIsAuthenticated] = useState(false);
7     const [user, setUser] = useState(null);
8
9     // Check session on mount
10    useEffect(() => {
11        checkSession();
12    }, []);
13
14    const checkSession = async () => {
15        const response = await fetch('http://localhost/Food-Web-App_1E4727/food-web-app/backend/auth/check-session.php', {
16            credentials: 'include'
17        });
18        const data = await response.json();
19
20        if (data.authenticated) {
21            setIsAuthenticated(true);
22            setUser(data.user);
23        }
24    };
25
26    const login = async (email, password) => {
27        const response = await fetch('http://localhost/Food-Web-App_1E4727/food-web-app/backend/auth/login.php', {
28            method: 'POST',
29            headers: { 'Content-Type': 'application/json' },
30            credentials: 'include',
31            body: JSON.stringify({ email, password })
32        });
33
34        const data = await response.json();
35
36        if (data.success) {
37            setIsAuthenticated(true);
38            setUser(data.user);
39            return { success: true };
40        }
41        return { success: false, message: data.message };
42    };
43
44    const logout = async () => {
45        await fetch('http://localhost/Food-Web-App_1E4827/food-web-app/backend/auth/logout.php', {
46            method: 'POST',
47            credentials: 'include'
48        });
49
50        setIsAuthenticated(false);
51        setUser(null);
52        return { success: true };
53    };
54
55    return (
56        <AuthContext.Provider value={{ isAuthenticated, user, login, logout }}>
57            {children}
58        </AuthContext.Provider>
59    );
60};

```

F2: Menu Browsing and Ordering

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

```

TRADITIONAL
1 <?php
2 require_once 'includes/config.php';
3 require_once 'includes/db.php';
4
5 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
6     // Validate email format
7     $email = filter_var($_POST['email'], FILTER_VALIDATE_EMAIL);
8     $password = $_POST['password'];
9
10    // Password pattern: 8+ chars, 1 letter, 1 number, 1 special char
11    $pattern = '/^(?=.*[A-Za-z])(?=.*\d)(?=.*[#?!@#$%^&*])[A-Za-z\d#?!@#$%^&*]{8,}$/';
12
13    if ($email) {
14        $error = "Invalid email format";
15    } elseif (!preg_match($pattern, $password)) {
16        $error = "Password must be 8+ chars with letter, number, special char";
17    } else {
18        // Check database for user
19        $stmt = $pdo->prepare("SELECT * FROM users WHERE email = ?");
20        $stmt->execute([$email]);
21        $user = $stmt->fetch();
22
23        if ($user && password_verify($password, $user['password'])) {
24            // Store user info in session
25            $SESSION['user_id'] = $user['id'];
26            $SESSION['username'] = $user['username'];
27            header('Location: index.php');
28            exit;
29        } else {
30            $error = "Invalid email or password";
31        }
32    }
33 }
34 >
35
36 <!-- HTML Form -->
37 <form method="POST" action="login.php">
38     <input type="email" name="email" placeholder="Email" required>
39     <input type="password" name="password" placeholder="Password" required>
40     <p>If (isset($error)) echo "<p class='error'>$error</p>" ;</p>
41     <button type="submit">Login</button>
42 </form>

```



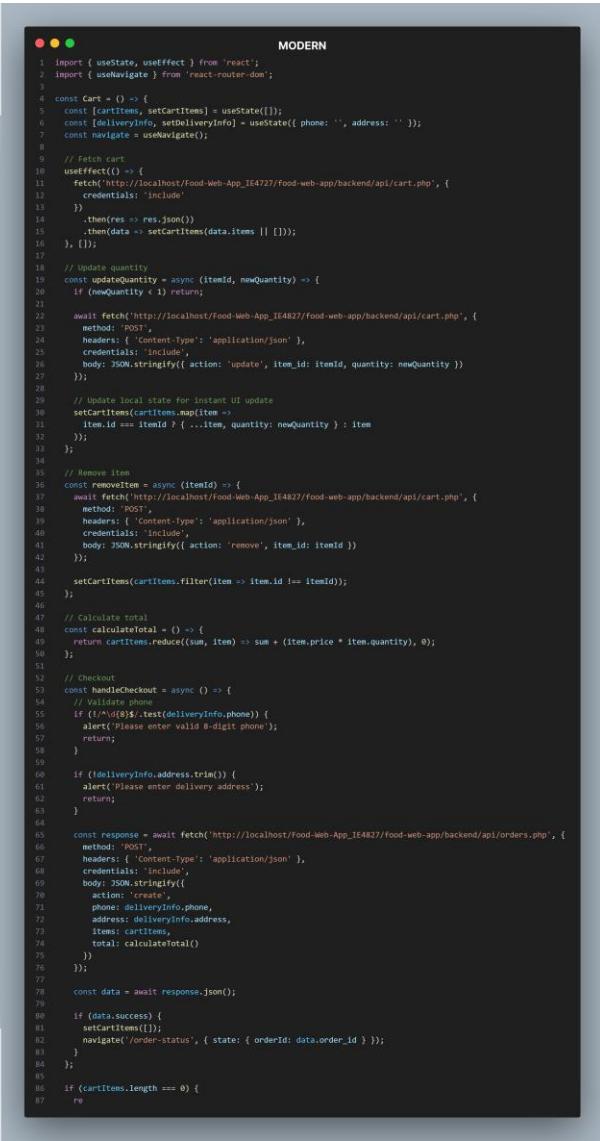
```

MODERN
1 import { useState, useEffect } from 'react';
2
3 const Menu = () => {
4     const [menuItems, setMenuItems] = useState([]);
5     const [filteredItems, setFilteredItems] = useState([]);
6     const [selectedCategory, setSelectedCategory] = useState('All');
7     const [searchQuery, setSearchQuery] = useState('');
8     const [sortOrder, setSortOrder] = useState('none');
9
10    // Fetch menu items from API
11    useEffect(() => {
12        fetch('http://localhost/Food-Web-App_1f4727/food-web-app/backend/api/menu_items.php')
13            .then(res => res.json())
14            .then(data => {
15                setMenuItems(data.items);
16                setFilteredItems(data.items);
17            })
18        }, []);
19
20    // Apply filters in real-time
21    useEffect(() => {
22        let items = [...menuItems];
23
24        // Category Filter
25        if (selectedCategory !== 'All') {
26            items = items.filter(item => item.category === selectedCategory);
27        }
28
29        // Search Filter (case-insensitive)
30        if (searchQuery) {
31            items = items.filter(item =>
32                item.name.toLowerCase().includes(searchQuery.toLowerCase())
33            );
34        }
35
36        // Price sorting
37        if (sortOrder === 'low-to-high') {
38            items.sort((a, b) => parseFloat(b.price) - parseFloat(a.price));
39        } else if (sortOrder === 'high-to-low') {
40            items.sort((a, b) => parseFloat(a.price) - parseFloat(b.price));
41        }
42
43        setFilteredItems(items);
44    }, [menuItems, selectedCategory, searchQuery, sortOrder]);
45
46    // Add to cart via API
47    const handleAddToCart = async (item) => {
48        const response = await fetch('http://localhost/Food-Web-App_1f4727/food-web-app/backend/api/cart.php', {
49            method: 'POST',
50            headers: { 'Content-Type': 'application/json' },
51            credentials: 'include',
52            body: JSON.stringify({ action: 'add', item_id: item.id, quantity: 1 })
53        });
54
55        const data = await response.json();
56
57        if (data.success) {
58            alert(`${item.name} added to cart!`);
59        } else {
60            alert('Please login first!');
61        }
62    };
63
64    return (
65        <div className="menu-page">
66            <!-- Search input - instant filtering -->
67            <input
68                type="text"
69                placeholder="Search dishes..."
70                value={searchQuery}
71                onChange={(e) => setSearchQuery(e.target.value)}
72            />
73
74            <!-- Sort dropdown -->
75            <select value={sortOrder} onChange={(e) => setSortOrder(e.target.value)}>
76                <option value="Sort by Price">Sort by Price</option>
77                <option value="Low-to-High">Low to High</option>
78                <option value="High-to-Low">High to Low</option>
79            </select>
80
81            <!-- Category buttons -->
82            <div className="Categories">
83                ['All', 'Mains', 'Starters', 'Sides', 'Desserts'].map(cat => (
84                    <button
85                        key={cat}
86                        onClick={() => setSelectedCategory(cat)}
87                    >
88                        {cat}
89                    </button>
90                )));
91            </div>
92
93            <!-- Menu grid -->
94            <div className="menu-grid">
95                <filteredItems.map(item => (
96                    <div key={item.id} className="menu-item">
97                        <a href={item.item_url} alt={item.name}>
98                            <img src={item.item_img}/>
99                            <p>{item.description}</p>
100                            <p>${parseFloat(item.price).toFixed(2)}</p>
101                            <button onClick={() => handleAddToCart(item)}>Add to Cart</button>
102                        </div>
103                    )));
104                </div>
105            </div>
106        </div>
107    );
108};
109
110 export default Menu;

```

F3: Shopping Cart

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



The image shows a comparison between traditional server-side PHP code and modern client-side React code for a shopping cart application.

TRADITIONAL (Left):

```

1 <?php
2 require_once 'includes/config.php';
3 require_once 'includes/db.php';
4
5 if (!isset($_SESSION['user_id'])) {
6     header('Location: login.php');
7     exit;
8 }
9
10 $user_id = $_SESSION['user_id'];
11
12 // Fetch cart items with JOIN
13 $stmt = $pdo->prepare(
14     "SELECT c.id, c.quantity, m.name, m.price, m.image_url
15     FROM menu_items m
16     JOIN menu_items_in_cart mi ON m.menu_item_id = c.id
17     WHERE c.user_id = ?"
18 );
19 $stmt->execute([$user_id]);
20 $cart_items = $stmt->fetchAll();
21
22 // Calculate total
23 $total = 0;
24 foreach ($cart_items as $item) {
25     $total += $item['price'] * $item['quantity'];
26 }
27
28
29 <?php if (empty($cart_items)) :>
30     <p>Your cart is empty!</p>
31     <a href="menu.php">Browse Menu</a>
32 </?php else :>
33     <?php foreach ($cart_items as $item):>
34         <div class="cart-item">
35             
36             <div>=$item['name']?</div>
37             <p><?php number_format($item['price'], 2) ?>/p>
38
39             <!-- Quantity controls - each button submits form -->
40             <form method="POST" action="backend/update-cart.php" style="display:inline">
41                 <input type="hidden" name="cart_id" value="=$item['id']?" />
42                 <input type="hidden" name="action" value="increase">
43                 <button type="submit">+</button>
44             </form>
45
46             <span><?php $item['quantity'] ?>/span>
47
48             <form method="POST" action="backend/update-cart.php" style="display:inline">
49                 <input type="hidden" name="cart_id" value="=$item['id']?" />
50                 <input type="hidden" name="action" value="decrease">
51                 <button type="submit">-</button>
52             </form>
53
54             <?php $total += $item['price'] * $item['quantity'], 2 ?>
55
56             <!-- Remove button -->
57             <form method="POST" action="backend/remove-from-cart.php">
58                 <input type="hidden" name="cart_id" value="=$item['id']?" />
59                 <button type="submit">Remove</button>
60             </form>
61         </div>
62     <?php endforeach; ?>
63
64     <h3>Total: <?php number_format($total, 2) ?></h3>
65
66     <!-- Checkout Form -->
67     <form method="POST" action="backend/checkout.php">
68         <input type="tel" name="phone" placeholder="Phone (8 digits)" pattern="\d{8}" required>
69         <input type="text" name="address" placeholder="Delivery Address" required>
70         <button type="submit">Place Order</button>
71     </form>
72 <?php endif; ?>

```

MODERN (Right):

```

1 import { useState, useEffect } from 'react';
2 import { useNavigate } from 'react-router-dom';
3
4 const Cart = () => {
5     const [cartItems, setCartItems] = useState([]);
6     const [deliveryInfo, setDeliveryInfo] = useState({ phone: '', address: '' });
7     const navigate = useNavigate();
8
9     // Fetch cart
10    useEffect(() => {
11        fetch(`http://localhost/Food-Web-App_1E4727/Food-web-app/backend/api/cart.php`, {
12            credentials: 'include'
13        })
14        .then(res => res.json())
15        .then(data => setCartItems(data.items || []));
16    }, []);
17
18    // Update quantity
19    const updateQuantity = async (itemId, newQuantity) => {
20        if (newQuantity < 1) return;
21
22        await fetch(`http://localhost/Food-Web-App_1E4827/Food-web-app/backend/api/cart.php`, {
23            method: 'POST',
24            headers: { 'Content-Type': 'application/json' },
25            credentials: 'include',
26            body: JSON.stringify({ action: 'update', itemId: itemId, quantity: newQuantity })
27        });
28
29    // Update local state for instant UI update
30    setCartItems(cartItems.map(item =>
31        item.id === itemId ? { ...item, quantity: newQuantity } : item
32    ));
33}
34
35 // Remove item
36 const removeItem = async (itemId) => {
37    await fetch(`http://localhost/Food-Web-App_1E4827/Food-web-app/backend/api/cart.php`, {
38        method: 'POST',
39        headers: { 'Content-Type': 'application/json' },
40        credentials: 'include',
41        body: JSON.stringify({ action: 'remove', itemId: itemId })
42    });
43
44    setCartItems(cartItems.filter(item => item.id !== itemId));
45}
46
47 // Calculate total
48 const calculateTotal = () => {
49    return cartItems.reduce((sum, item) => sum + (item.price * item.quantity), 0);
50}
51
52 // Checkout
53 const handleCheckout = async () => {
54    // Validation
55    if (!/^\d{8}$/.test(deliveryInfo.phone)) {
56        alert('Please enter valid 8-digit phone');
57        return;
58    }
59
60    if (!deliveryInfo.address.trim()) {
61        alert('Please enter delivery address');
62        return;
63    }
64
65    const response = await fetch(`http://localhost/Food-Web-App_1E4827/Food-web-app/backend/api/orders.php`, {
66        method: 'POST',
67        headers: { 'Content-Type': 'application/json' },
68        credentials: 'include',
69        body: JSON.stringify({
70            action: 'create',
71            phone: deliveryInfo.phone,
72            address: deliveryInfo.address,
73            items: cartItems,
74            total: calculateTotal()
75        });
76    });
77
78    const data = await response.json();
79
80    if (data.success) {
81        setCartItems([]);
82        navigate('order-success', { state: { orderId: data.order_id } });
83    }
84}
85
86 if (cartItems.length === 0) {
87    return (

```

F4: Checkout and Order Placement

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

TRADITIONAL - CLIENT SIDE VALIDATION

```

1 // cart.php - Line ~120
2 checkoutForm.addEventListener('submit', function(e) {
3   const phone = document.getElementById('customerPhone').value.trim();
4   const address = document.getElementById('deliveryAddress').value.trim();
5   const email = document.getElementById('customerEmail').value.trim();
6
7   let errors = [];
8
9   // Email validation
10  if (!/^\w+@[^\w]+\.\w+$/.test(email)) {
11    errors.push('Please enter a valid email address');
12  }
13
14  // Phone validation: exactly 8 digits
15  if (/^\d{8}$/.test(phone.replace(/\D/g, ''))) {
16    errors.push('Phone number must be exactly 8 digits');
17  }
18
19  // Address validation: minimum 10 characters
20  if (address.length < 10) {
21    errors.push('Delivery address must be at least 10 characters');
22  }
23
24  if (errors.length > 0) {
25    e.preventDefault();
26    alert('Please fix the following errors:\n' + errors.join('\n'));
27    return false;
28  }
29 });

```

TRADITIONAL - SERVER SIDE ORDER PROCESSING

```

1 <?php
2 // backend/process-checkout.php
3 </php>
4 // Server-side validation
5 if ($filter_var($customer_email, FILTER_VALIDATE_EMAIL)) {
6   $_SESSION['checkout_error'] = 'Invalid email format';
7   header('Location: ..checkout.php');
8   exit;
9 }
10
11 if (!preg_match('/^\d{8}$/', $clean_phone)) {
12   $_SESSION['checkout_error'] = 'Phone must be exactly 8 digits';
13   header('Location: ..checkout.php');
14   exit;
15 }
16
17 // Create order with transaction
18 $conn->begin_transaction();
19 try {
20   // 1. Create order record
21   $order_stmt = $conn->prepare("INSERT INTO orders
22     (user_id, customer_name, customer_email, customer_phone,
23      delivery_address, total_amount, status)
24     VALUES (?, ?, ?, ?, ?, ?, ?)");
25   $order_stmt->execute([$user_id, $customer_name, $customer_email,
26                         $customer_phone, $delivery_address, $total]);
27
28   // 2. Get unique order ID
29   $order_id = $conn->lastInsertId;
30
31   // 3. Insert order items
32   foreach ($cartItems as $item) {
33     $item_stmt->execute([$order_id, $item['menu_id']],
34                           [$item['quantity'], $item['price']]);
35   }
36
37   // 4. Clear cart
38   $conn->query("DELETE FROM cart_items WHERE user_id = $user_id");
39
40   $conn->commit();
41   header('Location: ..order-confirmation.php?order_id=' . $order_id);
42 } catch (Exception $e) {
43   $conn->rollback();
44   $_SESSION['checkout_error'] = 'Failed to place order';
45   header('Location: ..checkout.php');
46 }
47

```

MODERN - CLIENT SIDE VALIDATION

```

1 // src/pages/Cart.jsx - Line ~45
2 const validateForm = () => {
3   const errors = {};
4
5   // Email validation
6   if (!/^\w+@[^\w]+\.\w+$/.test(deliveryInfo.customerEmail)) {
7     errors.customerEmail = 'Please enter a valid email address';
8   }
8
9
10  // Phone validation: exactly 8 digits
11  if (/^\d{8}$/.test(deliveryInfo.customerPhone.replace(/\D/g, ''))) {
12    errors.customerPhone = 'Phone must be exactly 8 digits (e.g., 67483808)';
13  }
14
15  // Address validation: minimum 10 characters
16  if (!/^\d{10}$/.test(deliveryInfo.deliveryAddress.replace(/\D/g, ''))) {
17    errors.deliveryAddress = 'Address must be at least 10 characters';
18  }
19
20  setValidationErrors(errors);
21  return Object.keys(errors).length === 0;
22};
23
24 const handlePlaceOrder = async () => {
25   if (validateForm()) return;
26
27   setPlacingOrder(true);
28   try {
29     // Create order via API
30     const result = await ordersAPI.createOrder({
31       items: orderItems,
32       customer_name: deliveryInfo.customerName,
33       customer_email: deliveryInfo.customerEmail,
34       customer_phone: deliveryInfo.customerPhone,
35       delivery_address: deliveryInfo.deliveryAddress
36     });
37
38     if (result.success) {
39       await cartAPI.clearCart();
40       navigate('/order-confirmation', { state: { orderId: result.order.id } });
41     }
42   } catch (error) {
43     alert('Failed to place order. Please try again.');
44   } finally {
45     setPlacingOrder(false);
46   }
47 };

```

MODERN - BACKEND API

```

1 <?php
2 // backend/api/orders.php - Line ~60
3 </php>
4 // Validate delivery info
5 if (empty($customerName) || empty($customerEmail) ||
6 empty($customerPhone) || empty($deliveryAddress)) {
7   http_response_code(400);
8   echo json_encode(['error' => 'Please provide all delivery information']);
9   exit;
10 }
11
12 // Create order with transaction
13 $pdo->beginTransaction();
14 try {
15   // 1. Create order
16   $insOrder->execute([
17     'user_id' => $userId,
18     'status' => 'order placed',
19     'total_amount' => $total,
20     'customer_name' => $customerName,
21     'customer_email' => $customerEmail,
22     'customer_phone' => $customerPhone,
23     'delivery_address' => $deliveryAddress
24   ]);
25
26   // 2. Get order ID
27   $orderId = $pdo->lastInsertId();
28
29   // 3. Insert order items
30   foreach ($items as $item) {
31     $insItem->execute([
32       'order_id' => $orderId,
33       'menu_item_id' => $item['menu_item_id'],
34       'quantity' => $item['quantity'],
35       'price' => $item['price']
36     ]);
37
38   $pdo->commit();
39   echo json_encode(['success' => true, 'order' => ['id' => $orderId]]);
40 } catch (Throwable $e) {
41   $pdo->rollback();
42   echo json_encode(['error' => 'Failed to create order']);
43 }
44
45 >

```

F5: Order Status Tracking

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

```

TRADITIONAL - SERVER SIDE PHP PAGE
1 // AUTHENTICATION: Rediect if not logged in
2 if (!isset($_SESSION['user_id'])) {
3     header('location: login.php');
4     exit();
5 }
6
7 // RETRIEVE ORDERS: Sorted by date descending
8 $query = "SELECT o.id, o.total_amount, o.status, o.created_at
9         FROM orders o
10        WHERE o.user_id = ?
11        ORDER BY o.created_at DESC";
12
13 $stmt = $conn->prepare($query);
14 $stmt->bind_param('i', $_SESSION['user_id']);
15 $stmt->execute();
16 $orders = $stmt->get_result();
17 $orders->fetch_all(MYSQLI_ASSOC);
18
19 include 'includes/header.php';
20
21 <div class="order-history-page">
22     <div class="order-history">
23         <p>(Type if (Count(Orders) == 0))>
24             (1 - Delivery Status) ->
25             (0 You haven't placed any orders yet.)>
26             A Break Even (Avg. Revenue Number)>
27         </p>
28         <table class="status-table">
29             <thead>
30                 <tr>
31                     <th>Order ID</th>
32                     <th>Status</th>
33                     <th>Total Amount</th>
34                 </tr>
35             </thead>
36             <tbody>
37                 <tr>
38                     <td>$_SESSION['order_id']</td>
39                     <td>$_SESSION['order_status']</td>
40                     <td>$_SESSION['total_amount']</td>
41                 </tr>
42             </tbody>
43         </table>
44         <div class="button-area">
45             <button type="button" value="Update Order Status" onclick="updateOrderStatus(this, '$_SESSION['order_id']');">Update Order Status</button>
46         </div>
47     </div>
48 </div>
49 <div>
50     <pre><?php include 'includes/footer.php'; ?>
51 </div>

```

```

TRADITIONAL - process-advance-order.php
1 require_once '../includes/config.php';
2 require_once '../includes/db.php';
3
4 $order_id = intval($_POST['order_id']) ?? 0;
5 $user_id = $_SESSION['user_id'];
6
7 // VERIFY OWNERSHIP: Check order belongs to user
8 $stmt = $conn->prepare("SELECT status FROM orders WHERE id = ? AND user_id = ?");
9 $stmt->bind_param('ii', $order_id, $user_id);
10 $stmt->execute();
11 $order = $stmt->get_result()->fetch_assoc();
12
13 if (!$order) {
14     $_SESSION['order_error'] = 'Order not found';
15     header('Location: ../order-history.php');
16     exit();
17 }
18
19 // STATUS PROGRESSION: Predefined stages
20 $statuses = ['order_placed', 'preparing', 'out_for_delivery', 'delivered'];
21 $current_index = array_search($order['status'], $statuses);
22
23 // CALCULATE NEXT STATUS
24 if ($current_index === false || $current_index > count($statuses) - 1) {
25     $_SESSION['order_error'] = 'Order cannot be advanced further';
26     header('Location: ../order-history.php');
27     exit();
28 }
29
30 $next_status = $statuses[$current_index + 1];
31
32 // UPDATE DATABASE
33 $update = $conn->prepare("UPDATE orders SET status = ? WHERE id = ?");
34 $update->bind_param('si', $next_status, $order_id);
35 $update->execute();
36
37 $_SESSION['order_success'] = 'Status updated to: ' . $next_status;
38
39 // REDIRECT: Full page reload
40 header('Location: ../order-history.php');
41 exit();
42

```

```

MODERN - REACT COMPONENT
1 import { useState } from 'react';
2 import { ordersAPI } from './api/order';
3
4 export default function OrderStatusPage() {
5     const [orders, setOrders] = useState([]);
6     const [loading, setLoading] = useState(true);
7
8     // FETCH ORDERS: Load user's orders via API
9     const fetchOrders = () => {
10         setLoading(true);
11         try {
12             const data = await ordersAPI.getOrders();
13             setOrders(data);
14         } catch (e) {
15             console.error(e);
16         } finally {
17             setLoading(false);
18         };
19     };
20
21 useEffect(() => {
22     const id = localStorage.getItem('user_id');
23     if (id) {
24         fetchOrders();
25     }
26 }, []);
27
28 // ADVANCE STATUS: Update via API, no page reload
29 const advance = async (id) => {
30     try {
31         const response = await ordersAPI.advancedOrderStatus(id);
32         await load();
33     } catch (e) {
34         alert('Failed to advance status');
35     }
36 };
37
38 // TOTAL AMOUNT
39 const totalAmount = () => {
40     const total = orders.reduce((acc, order) => acc + order.total_amount, 0);
41     return `Total amount: ${total.toFixed(2)}`;
42 };
43
44 // CURRENT STATUS
45 const currentStatus = (order) => {
46     const status = order.status.toLowerCase();
47     return `Status: ${order.status}`.replace(status, status[0].toUpperCase());
48 };
49
50 // ORDERS LIST
51 const ordersList = () => {
52     return orders.map(order) => {
53         const id = order.id;
54         const status = currentStatus(order);
55         const total = totalAmount();
56
57         const advanceButton = `Only if not delivered` === order.status ? '' : (
58             <button onClick={() => advance(id)}> Advance Status </button>
59         );
60
61         return (
62             <div key={id}>
63                 <div class="order-card">
64                     <div class="order-id">{id}</div>
65                     <div class="order-status">{status}</div>
66                     <div class="order-total">{total}</div>
67                 </div>
68                 <div>
69                     <div class="button-area">
70                         <button type="button" value="Update Order Status" onClick="updateOrderStatus(this, '{id}')">Update Order Status</button>
71                     </div>
72                 </div>
73             </div>
74         );
75     });
76 }
77
78 // RENDER ORDERS
79 const renderOrders = () => {
80     return (
81         <div>
82             <h2>Order Status</h2>
83             <div>
84                 <div>
85                     <div>No orders yet.</div>
86                 </div>
87             </div>
88             <div>
89                 <div class="order-list">
90                     {ordersList()}
91                 </div>
92             </div>
93         </div>
94     );
95 }
96
97 // EXPORT
98 export default OrderStatusPage;

```

```

MODERN - backend/api/orders.php
1 // PREDEFINED STATUS PROGRESSION
2 $statuses = [
3     'order_placed',
4     'order_accepted',
5     'preparing_order',
6     'out_for_delivery',
7     'order_delivered'
8 ];
9
10 if (method === 'POST') {
11     $action = $_POST['action'] ?? 'create';
12
13     if ($action === 'advance') {
14         // GET ORDER ID FROM REQUEST
15         $order_id = intval($_POST['id']) ?? 0;
16
17         if ($order_id > 0) {
18             http_response_code(404);
19             echo json_encode(['error' => 'Missing order id']);
20             exit();
21         }
22
23         // FETCH ORDER: Verify ownership
24         $order = get_order_by_id($order_id, $user_id);
25         if (!$order) {
26             http_response_code(404);
27             echo json_encode(['error' => 'Order not found']);
28             exit();
29         }
30
31         // UPDATE CURRENT STATUS: Change order status
32         $current = strtotime(trim($order['status']));
33         $idx = array_search($current, $statuses, true);
34         if ($idx === false) $idx = 0;
35
36         // CALCULATE NEW STATUS: Limit to last status
37         $new_idx = min($idx + 1, count($statuses) - 1);
38         $new_status = $statuses[$new_idx];
39
40         // UPDATE DATABASE: Change order status
41         $update = $db->prepare("UPDATE orders SET status = ? WHERE id = ? AND user_id = ?");
42         $update->bind_param('isi', $new_status, $order_id, $user_id);
43
44         // RETURN JSON: Modern approach returns data, no redirect
45         $response = json_encode(['success' => true, 'order' => $order]);
46         exit();
47     }
48 }
49
50 // FETCH UPDATED ORDERS WITH NEW STATUS
51 $updated = get_order_by_id($order_id, $user_id);
52
53 // RETURN JSON: Modern approach returns data, no redirect
54 $response = json_encode(['success' => true, 'order' => $updated]);
55
56 }
57

```

F6: Order History Viewing

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

<pre> TRADITIONAL - order-history.php 1 <?php 2 3 require_once 'includes/config.php'; 4 require_once 'includes/db.php'; 5 6 // F6: REQUIREMENT: Redirect to login if not authenticated 7 if (!isset(\$_SESSION['user_id'])) { 8 \$_SESSION['login_error'] = 'Please login to view order history'; 9 header('Location: login.php'); 10 exit; 11 } 12 13 \$user_id = \$_SESSION['user_id']; 14 15 // F6: FETCH USER'S ORDERS sorted by descending date/time 16 \$query = "SELECT o.id, o.total_amount, o.status, o.created_at, 17 COUNT(o.id) AS item_count 18 FROM orders o 19 LEFT JOIN order_items oi ON o.id = oi.order_id 20 WHERE o.user_id = ? 21 GROUP BY o.id 22 ORDER BY o.created_at DESC; // Descending order (newest first) 23 24 \$stmt = \$conn->prepare(\$query); 25 \$stmt->bind_param('i', \$user_id); 26 \$stmt->execute(); 27 \$result = \$stmt->get_result(); 28 \$orders = \$result->fetch_all(MYSQLI_ASSOC); 29 30 include 'includes/header.php'; 31 ?> 32 33 <!-- F6: ORDER HISTORY DISPLAY --> 34 <div class="order-history-page"> 35 <h1>Order History</h1> 36 37 <?php if (count(\$orders) === 0): ?> 38 <!-- EMPTY STATE: No orders found --> 39 <div class="no-orders"> 40 <p>You haven't placed any orders yet.</p> 41 Browse Menu 42 </div> 43 44 <?php else: ?> 45 <!-- DYNAMICALLY GENERATED HTML from database --> 46 <div class="orders-container"> 47 <?php foreach (\$orders as \$order): ?> 48 <?php 49 // Fetch order items for each order 50 \$items_query = "SELECT m.name, oi.quantity, oi.price 51 FROM order_items oi 52 INNER JOIN menu_items m ON oi.menu_item_id = m.id 53 WHERE oi.order_id = ?"; 54 55 \$items_stmt = \$conn->prepare(\$items_query); 56 \$items_stmt->bind_param('i', \$order['id']); 57 \$items_stmt->execute(); 58 \$items_result = \$items_stmt->get_result(); 59 \$items = \$items_result->fetch_all(MYSQLI_ASSOC); 60 > 61 62 <div class="order-card"> 63 <div class="order-header"> 64 <?php if (\$order['id']): ?> 65 <h3>Order #<?php echo \$order['id']; ?></h3> 66 67 <!-- F6: ORDER DATE AND TIME in readable format --> 68 <p>Class="order-date"> 69 <?php echo date('M d, Y h:i A', strtotime(\$order['created_at'])); ?> 70 </p> 71 </div> 72 73 <!-- F6: CURRENT STATUS --> 74 <div class="order-status"> 75 76 <?php echo htmlspecialchars(\$order['status']); ?> 77 78 </div> 79 80 <!-- ORDER ITEMS LIST --> 81 <div class="order-items"> 82 <?php echo \$order['item_count']; ?></h4> 83 84 <?php foreach (\$items as \$item): ?> 85 86 <?php echo htmlspecialchars(\$item['name']); ?> 87 <?php echo \$item['quantity']; ?> 88 - \$?php echo number_format(\$item['price'] * \$item['quantity'], 2); ?> 89 90 </?php endforeach; ?> 91 92 </div> 93 94 <div class="order-footer"> 95 <!-- F6: TOTAL AMOUNT --> 96 <div class="order-total"> 97 Total: \$<?php echo number_format(\$order['total_amount'], 2); ?> 98 </div> 99 </div> 100 101 <?php endforeach; ?> 102 103 </?php endif; ?> 104 105 </div> 106 107 <?php 108 \$conn->close(); 109 include 'includes/footer.php'; 110 > </pre>	<pre> MODERN - OrderStatus.jsx 1 import React, { useEffect, useState } from 'react'; 2 import { useNavigate } from 'react-router-dom'; 3 import { orderData } from '../services/api'; 4 5 export default function OrderStatusPage() { 6 const navigate = useNavigate(); 7 8 // REACT STATE: Store orders data 9 const [orders, setOrders] = useState([]); 10 const [loading, setLoading] = useState(true); 11 const [error, setError] = useState(null); 12 13 /* F6: LOAD ORDER HISTORY via API */ 14 // Modern approach: API call instead of server-side query 15 const load = async () => { 16 setError(''); 17 setLoading(true); 18 try { 19 // API CALL: GET request to fetch user's orders 20 const data = await orderData.getOrders(); 21 setOrders(data.orders []); 22 } catch (e) { 23 setError(e.message 'Failed to load orders'); 24 } finally { 25 setLoading(false); 26 } 27 }; 28 29 // F6: Render orders when component mounts 30 useEffect(() => { 31 load(); 32 }, []); 33 34 // LOADING STATE 35 if (loading) { 36 return (37 <div className="order-status-page"> 38 <div className="order-status-container"> 39 <h1>Order History</h1> 40 41 <div className="error-message" role="alert"> 42 <small>Error</small> 43 </div> 44 45 <div className="order-list"> 46 <table border="1"> 47 <thead> 48 <tr> 49 <th>Order ID</th> 50 <th>Order Date</th> 51 <th>Status</th> 52 </tr> 53 </thead> 54 <tbody> 55 <tr> 56 <td>#1</td> 57 <td>2023-10-01</td> 58 <td>P</td> 59 </tr> 60 </tbody> 61 </table> 62 </div> 63 64 </div> 65 </div> 66); 67 } 68 69 /* F6: ORDER LIST - Dynamically rendered from API data */ 70 return (71 <div className="order-list"> 72 <table border="1"> 73 <thead> 74 <tr> 75 <th>Order ID</th> 76 <th>Order Date</th> 77 <th>Status</th> 78 </tr> 79 </thead> 80 <tbody> 81 <tr> 82 <td>#1</td> 83 <td>2023-10-01</td> 84 <td>P</td> 85 </tr> 86 </tbody> 87 </table> 88 </div> 89 90 <div className="order-items"> 91 <table border="1"> 92 <thead> 93 <tr> 94 <th>Item ID</th> 95 <th>Item Name</th> 96 <th>Quantity</th> 97 <th>Price</th> 98 </tr> 99 </thead> 100 <tbody> 101 <tr> 102 <td>1</td> 103 <td>Burger</td> 104 <td>2</td> 105 <td>\$10.00</td> 106 </tr> 107 </tbody> 108 </table> 109 </div> 110 111 <div className="order-total"> 112 Total: \$<?php echo number_format(\$order['total_amount'], 2); ?> 113 </div> 114); 115 116 117 118 119 120 </pre>
---	--

F7: Customer Feedback Submission

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

<pre> TRADITIONAL - feedback.php 1 require_once 'includes/config.php'; 2 require_once 'includes/db.php'; 3 4 // PREP STATEMENT 5 \$stmt = \$conn->prepare("SELECT full_name, email FROM users WHERE id = ?"); 6 \$userData = []; 7 if (\$stmt->execute()) { 8 \$user_id = \$stmt->fetch_assoc(); 9 \$full_name = \$user_id['full_name']; 10 \$email = \$user_id['email']; 11 \$stmt->close(); 12 \$stmt->execute(); 13 \$result = \$stmt->get_result(); 14 \$userData = \$result->fetch_assoc(); 15 } 16 17 // FETCH APPROVED FEEDBACKS: Only show approved=1 18 \$feedbacks = []; 19 \$feedback_query = "SELECT id, name, rating, feedback, created_at 20 FROM feedbacks 21 ORDER BY created_at DESC"; 22 \$feedback_result = \$conn->query(\$feedback_query); 23 if (\$feedback_result) { 24 \$feedbacks = \$feedback_result->fetch_all(MYSQLI_ASSOC); 25 } 26 27 include "includes/header.php"; 28 ?> 29 30 <div class="feedback-page"> 31 <h1>We Value Your Feedback</h1> 32 33 <!-- F7: FEEDBACK SUBMISSION FORM --> 34 <section class="feedback-form-section"> 35 <h2>Share Your Experience</h2> 36 37 <form id="feedbackForm" method="POST" action="backend/process-feedback.php"> 38 <!-- NAME INPUT: Pre-filled if logged in --> 39 <div class="form-group"> 40 <label for="customer_name">Your Name </label> 41 <input type="text" id="customer_name" 42 name="customer_name" 43 value=<?php echo isset(\$userData['full_name']) ? htmlspecialchars(\$userData['full_name']) : '' ?> 44 required> 45 </div> 46 47 <!-- EMAIL INPUT: Pre-filled if logged in --> 48 <div class="form-group"> 49 <label for="customer_email">Email </label> 50 <input type="email" id="customer_email" 51 name="customer_email" 52 value=<?php echo isset(\$userData['email']) ? htmlspecialchars(\$userData['email']) : '' ?> 53 required> 54 </div> 55 56 <!-- RATING INPUT: 5 stars selection --> 57 <div class="form-group"> 58 <label for="rating">Overall Rating </label> 59 <select id="rating" name="rating" required> 60 <option value="Select rating">Select rating</option> 61 <option value="5">5 Stars - Excellent</option> 62 <option value="4">4 Stars - Good</option> 63 <option value="3">3 Stars - Average</option> 64 <option value="2">2 Stars - Below Average</option> 65 <option value="1">1 Star - Poor</option> 66 </select> 67 </div> 68 69 <!-- COMMENT/FEEDBACK TEXTAREA --> 70 <div class="form-group"> 71 <label for="comment">Your Feedback </label> 72 <textare id="comment" 73 name="comment" 74 rows="5" 75 placeholder="Tell us about your experience..."> 76 required</textare> 77 </div> 78 79 <button type="submit" class="submit-btn">Submit Feedback</button> 80 </form> 81 </section> 82 83 <!-- F7: DISPLAY APPROVED FEEDBACKS --> 84 <section class="feedback-display-section"> 85 <h3>User Feedbacks</h3> 86 <div>If (count(\$feedbacks) > 0):> 87 <ul class="feedbacks-list"> 88 <php foreach (\$feedbacks as \$feedback):> 89 <?php echo \$feedback['name']; ?> 90 <div class="feedback-header"> 91 <?php echo htmlspecialchars(\$feedback['name']); ?> 92 </div> 93 94 <!-- RATING DISPLAY: stars visualization --> 95 <div class="feedback-rating"> 96 <?php for (\$i = 1; \$i <= 5; \$i++): ?> 97 <?php echo \$i <= \$feedback['rating'] ? 'filled' : '' ?> 98 </?php endforeach; ?> 99 </div> 100 101 <!-- TIMESTAMP --> 102 103 <?php echo date('M d, Y', strtotime(\$feedback['created_at'])); ?> 104 105 106 <!-- FEEDBACK TEXT --> 107 <div class="feedback-text"> 108 <?php echo nl2br(htmlspecialchars(\$feedback['feedback'])); ?> 109 </div> 110 111 112 </php endforeach; ?> 113 114 <?php else: ?> 115 <p>No feedbacks yet. Be the first to share your experience!</p> 116 </?php endif; ?> 117 </div> 118 </section> 119 </div> 120 121 <?php export default feedback; ?> </pre>	<pre> MODERN - feedback.jsx 1 import { useState, useEffect } from react; 2 import { Form, Input, Button, Row, Col, Card, Textarea } from 'react-bootstrap'; 3 import FeedbackAPI from '../services/api'; 4 5 const Feedback = () => { 6 const [user, isAuthenticated] = useState(); 7 8 // FORM STATE 9 const [formData, setFormData] = useState({ 10 name: '', 11 email: '', 12 rating: 0, 13 feedback: '' 14 }); 15 16 // [feedbacks, setFeedbacks] = useState([]); 17 const [loading, setLoading] = useState(false); 18 19 // PRE-FILL: auto fill if logged in 20 useEffect(() => { 21 if (user &amp; isAuthenticated) { 22 setLoading(true); 23 const [user, user_email] = user; 24 const [full_name, user_email] = user.username ''; 25 const [email, user_email] = user_email ''; 26 } 27 }, [isAuthenticated, user]); 28 29 // PULL APPROVED FEEDBACKS 30 useEffect(() => { 31 const [feedbacks, setFeedbacks] = useState([]); 32 33 const fetchFeedbacks = async () => { 34 try { 35 const data = await FeedbackAPI.getFeedbacks(); 36 setFeedbacks(data); 37 } catch (error) { 38 console.error(error); 39 } 40 }; 41 42 // handle change 43 const handleChange = (e) => { 44 setFormData({ ...formData, [e.target.name]: e.target.value }); 45 }; 46 47 // SUBMIT FEEDBACK: API call, no page reload 48 const handleSubmit = (e) => { 49 e.preventDefault(); 50 setLoading(true); 51 52 try { 53 const result = await FeedbackAPI.submitFeedback(formData); 54 55 if (result.success) { 56 const [user, user_email] = user; 57 const [full_name, user_email] = user.username ''; 58 const [rating, user_email] = user_email ''; 59 const [feedback, user_email] = user_email ''; 60 61 alert(`Thank you for your feedback! \${full_name} \${rating}`); 62 setLoading(false); 63 setLoading(true); 64 65 const [feedbacks, setFeedbacks] = useState([]); 66 const [loading, setLoading] = useState(false); 67 const [loading, setLoading] = useState(false); 68 69 return (70 <div> 71 <h1>We Value Your Feedback</h1> 72 <h3>Feedbacks</h3> 73 74 <li key={id}> 75 <div> 76 <div>{name}</div> 77 <div>{rating}</div> 78 <div>{feedback}</div> 79 </div> 80 81 82 </div> 83); 84 } 85 } catch (error) { 86 console.error(error); 87 } 88 } 89 90 // RENDER STATE 91 const renderStars = (rating, interactive = false) => { 92 return [...Array(rating)].map((_, i) => { 93 const [key] = user; 94 const [rating_star] = rating[i].filled ''; 95 const [rating_star] = rating[i].interactive user.username ''; 96 const [rating_star] = rating_star ''; 97 const [rating_star] = rating_star ''; 98 99 return (100 <div> 101 * 102 </div> 103); 104 }); 105 } 106 107 const renderFeedback = (feedback, user_email) => { 108 const [feedback] = feedback; 109 const [rating, user_email] = user_email ''; 110 const [rating, user_email] = user_email ''; 111 const [rating, user_email] = user_email ''; 112 113 return (114 <div> 115 <div>{rating}</div> 116 <div>{feedback}</div> 117 <div>{user_email}</div> 118 </div> 119); 120 } 121 122 const renderForm = () => { 123 const [user, user_email] = user; 124 const [full_name, user_email] = user.username ''; 125 const [email, user_email] = user_email ''; 126 127 return (128 <div> 129 <Form> 130 <Row> 131 <Col> 132 <Input type="text" id="customer_name" 133 name="customer_name" 134 value={full_name} 135 onChange={handleChange} 136 placeholder="Name" /> 137 </Col> 138 <Col> 139 <Input type="email" id="customer_email" 140 name="customer_email" 141 value={email} 142 onChange={handleChange} 143 placeholder="Email" /> 144 </Col> 145 </Row> 146 147 <div> 148 <Label for="rating">Overall Rating</Label> 149 <Select id="rating" name="rating" required> 150 <option value="Select rating">Select rating</option> 151 <option value="5">5 Stars - Excellent</option> 152 <option value="4">4 Stars - Good</option> 153 <option value="3">3 Stars - Average</option> 154 <option value="2">2 Stars - Below Average</option> 155 <option value="1">1 Star - Poor</option> 156 </Select> 157 </div> 158 159 <Textarea id="comment" 160 name="comment" 161 rows="5" 162 placeholder="Tell us about your experience..." /> 163 </Form> 164 <Button type="submit" disabled={loading}> 165 {loading ? 'Submitting...' : 'Submit Feedback'} 166 </Button> 167 </div> 168); 169 } 170 171 // DISPLAY APPROVED FEEDBACKS 172 const renderFeedbacks = () => { 173 const [feedbacks, setFeedbacks] = useState([]); 174 const [loading, setLoading] = useState(true); 175 176 return (177 <div> 178 <h3>What Our Customers Say</h3> 179 180 <li key={id}> 181 <div> 182 <div>{name}</div> 183 <div>{rating}</div> 184 <div>{feedback}</div> 185 <div>{user_email}</div> 186 </div> 187 188 189 </div> 190); 191 } 192 193 // EXPORT 194 export default feedback; </pre>
---	--

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.

<pre> 1 require_once 'includes/config.php'; 2 3 \$pageTitle = 'Catering Services - ' . SITE_NAME; 4 \$additionalCSS = "catering.css"; 5 6 include 'includes/header.php'; 7 8 9 <!-- FB: CATERING PAGE --> 10 <div class="catering-page"> 11 <!-- FB: THREE-TIER COMPARISON TABLE --> 12 <section class="catering-section"> 13 <div class="catering-container"> 14 <div class="menu-header"> 15 <h3>Compare Packages</h3> 16 <p>Compare our catering packages to find the perfect fit for your event!</p> 17 </div> 18 19 <!-- FB: THREE-TIER COMPARISON TABLE --> 20 <div class="table-wrap"> 21 <table class="catering-table" role="table"> 22 <caption class="sr-only"> 23 Comparison of Essential, Premium and Deluxe catering packages 24 </caption> 25 26 <thead> 27 <tr> 28 <th scope="col">Feature</th> 29 <th scope="col">Essential</th> 30 <th scope="col">Popular</th> 31 <th scope="col">Premium</th> 32 <th scope="col">Deluxe</th> 33 </tr> 34 </thead> 35 36 <!-- TABLE BODY: Side-by-side feature comparison --> 37 <tbody> 38 <!-- PRICE COMPARISON --> 39 <tr> 40 <th scope="row" class="feature-name">Price per Person</th> 41 <td>\$15-20</td> 42 <td class="popular-column">\$25-35</td> 43 <td>\$40-50</td> 44 <td></td> 45 </tr> 46 47 <!-- MAIN COURSE OPTIONS --> 48 <tr> 49 <th scope="row" class="feature-name">Main Course Options</th> 50 <td>3 options</td> 51 <td class="popular-column">5 options</td> 52 <td>Unlimited</td> 53 <td></td> 54 </tr> 55 56 <!-- SIDE DISHES --> 57 <tr> 58 <th scope="row" class="feature-name">Side Dishes</th> 59 <td>2 sides</td> 60 <td class="popular-column">4 sides</td> 61 <td>Full selection</td> 62 <td></td> 63 </tr> 64 65 <!-- APPETIZERS --> 66 <tr> 67 <th scope="row" class="feature-name">Appetizers</th> 68 <td></td> 69 <td class="popular-column">Selection</td> 70 <td>Premium Selection</td> 71 <td></td> 72 </tr> 73 74 <!-- DESSERTS --> 75 <tr> 76 <th scope="row" class="feature-name">Desserts</th> 77 <td></td> 78 <td class="popular-column">Dessert bar</td> 79 <td>Dessert & coffee bar</td> 80 <td></td> 81 </tr> 82 83 <!-- SETUP SERVICE --> 84 <tr> 85 <th scope="row" class="feature-name">Setup</th> 86 <td>Basic</td> 87 <td class="popular-column">Premium</td> 88 <td>Full Service</td> 89 <td></td> 90 </tr> 91 92 <!-- TABLEWARE --> 93 <tr> 94 <th scope="row" class="feature-name">Tableware</th> 95 <td>Disposable</td> 96 <td class="popular-column">Basic Glassware</td> 97 <td>Premium Glassware</td> 98 <td></td> 99 </tr> 100 101 <!-- SERVICE STAFF --> 102 <tr> 103 <th scope="row" class="feature-name">Service Staff</th> 104 <td></td> 105 <td class="popular-column">1 staff</td> 106 <td>2 staff</td> 107 <td aria-label="Included"></td> 108 </tr> 109 110 </tbody> 111 </table> 112 </div> 113 </section> 114 115 <!-- FB: CONTACT SECTION with catering team email --> 116 <div class="catering-contact"> 117 <h2>Custom Requests or Want to Cater?</h2> 118 <p>Contact our catering team directly at:</p> 119 120 <!-- EMAIL ADDRESS for bookings and custom requests --> 121 <div class="contact-details"> 122 <div class="contact-item"> 123 124 Catering@leckerhaus.com 125 126 </div> 127 </div> 128 </div> 129 </div> 130 </section> 131 </div> 132 133 <?php include 'includes/footer.php'; > </pre>	<pre> 1 import './Catering.css'; 2 3 const Catering = () => { 4 return (5 <div className="catering-page"> 6 <!-- FB: PACKAGE COMPARISON TABLE --> 7 <section className="catering-section"> 8 <div className="catering-container"> 9 <div className="menu-header"> 10 <h3>Compare Packages</h3> 11 <p>Compare our catering packages to find the perfect fit for your event!</p> 12 </div> 13 14 <!-- FB: THREE-TIER COMPARISON TABLE (JSX) --> 15 <div className="table-wrapper"> 16 <table className="catering-table" role="table"> 17 <caption class="sr-only"> 18 Comparison of Essential, Premium and Deluxe catering packages 19 </caption> 20 21 <thead> 22 <tr> 23 <th>Feature</th> 24 <th>Essential</th> 25 <th>Popular</th> 26 <th>Premium</th> 27 <th>Deluxe</th> 28 </tr> 29 </thead> 30 31 <tbody> 32 <tr> 33 <th>Feature Name</th> 34 <td>\$15-20</td> 35 <td class="popular-column">\$25-35</td> 36 <td>\$40-50</td> 37 <td></td> 38 </tr> 39 40 <tr> 41 <th>Main Course Options</th> 42 <td>3 options</td> 43 <td class="popular-column">5 options</td> 44 <td>Unlimited</td> 45 <td></td> 46 </tr> 47 48 <tr> 49 <th>Side Dishes</th> 50 <td>2 sides</td> 51 <td class="popular-column">4 sides</td> 52 <td>Full selection</td> 53 <td></td> 54 </tr> 55 56 <tr> 57 <th>Appetizers</th> 58 <td></td> 59 <td class="popular-column">Selection</td> 60 <td>Premium Selection</td> 61 <td></td> 62 </tr> 63 64 <tr> 65 <th>Desserts</th> 66 <td></td> 67 <td class="popular-column">Dessert bar</td> 68 <td>Dessert & coffee bar</td> 69 <td></td> 70 </tr> 71 72 <tr> 73 <th>Setup Service</th> 74 <td>Basic</td> 75 <td class="popular-column">Premium</td> 76 <td>Full Service</td> 77 <td></td> 78 </tr> 79 80 <tr> 81 <th>Tableware</th> 82 <td>Disposable</td> 83 <td class="popular-column">Basic Glassware</td> 84 <td>Premium Glassware</td> 85 <td></td> 86 </tr> 87 88 <tr> 89 <th>Service Staff</th> 90 <td></td> 91 <td class="popular-column">1 staff</td> 92 <td>2 staff</td> 93 <td aria-label="Included"></td> 94 </tr> 95 96 <tr> 97 <th>Custom Menu</th> 98 <td></td> 99 <td class="popular-column">Included</td> 100 <td></td> 101 <td></td> 102 </tr> 103 104 </tbody> 105 </table> 106 </div> 107 </div> 108 </section> 109 110 <!-- FB: CONTACT SECTION --> 111 <div className="catering-contact"> 112 <div className="catering-container"> 113 <h3>Request a Catering Quote</h3> 114 <p>Fill out the form below and we'll get back to you within 24 hours</p> 115 <div> 116 <div> 117 <!-- Contact info with email --> 118 <div className="contact-info" style={{ marginTop: '2rem', textAlign: 'center' }}> 119 <input type="text" style={{ width: '100%', height: '1.2rem', color: '#666666' }} /> 120 <small>Or contact us directly at:</small> 121 </div> 122 <div style={{ width: '100%', height: '1.2rem', color: '#ff00a1' }}> 123 <input type="text" value="catering@leckerhaus.com" /> 124 </div> 125 </div> 126 </div> 127 </div> 128 </div> 129 </div> 130 </div> 131 </div> 132 133 export default Catering; </pre>
--	--

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
1 import { useState } from 'react';
2
3 const Catering = () => {
4   const [formData, setFormData] = useState({
5     name: '',
6     email: '',
7     phone: '',
8     eventDate: '',
9     guestCount: '',
10    package: '',
11    message: ''
12  });
13
14  const handleChange = (e) => {
15    setFormData({ ...formData, [e.target.name]: e.target.value });
16  };
17
18  try {
19    const response = await fetch('.../backend/api/catering.php', {
20      method: 'POST',
21      headers: { 'Content-type': 'application/json' },
22      body: JSON.stringify(formData)
23    });
24
25    const data = await response.json();
26    if (response.ok) {
27      alert(data.message);
28      setFormData({ name: '', email: '', phone: '', eventDate: '',
29                  guestCount: '', package: '', message: '' });
30    } else {
31      alert(data.error);
32    }
33  } catch (err) {
34    alert('Failed to send inquiry');
35  } finally {
36    setLoading(false);
37  }
38};
39
40 return (
41   <section className="catering-contact">
42     <h2>Request a Catering Quote</h2>
43
44     <form onSubmit={handleSubmit}>
45       <input type="text" name="name" value={formData.name}
46             onChange={handleChange} placeholder="Your Name" required />
47
48       <input type="email" name="email" value={formData.email}
49             onChange={handleChange} placeholder="Email" required />
50
51       <input type="tel" name="phone" value={formData.phone}
52             onChange={handleChange} placeholder="67489380" required />
53
54       <input type="date" name="eventDate" value={formData.eventDate}
55             onChange={handleChange} required />
56
57       <input type="number" name="guestCount" value={formData.guestCount}
58             onChange={handleChange} placeholder="Number of Guests" required />
59
60       <select name="package" value={formData.package} onChange={handleChange} required>
61         <option value="">Select Package</option>
62         <option value="Essential">Essential ($15-20)</option>
63         <option value="Premium">Premium ($25-35)</option>
64         <option value="Deluxe">Deluxe ($40-50)</option>
65       </select>
66
67       <textarea name="message" value={formData.message}
68             onChange={handleChange} placeholder="Special requests..." />
69
70     <button type="submit" disabled={loading}>
71       {(loading ? 'Sending...' : 'Send Inquiry')
72     </button>
73   </form>
74
75   <p>Or email: <strong>f32ee@localhost</strong></p>
76 </div>
77 };
78
79 <div>
80   export default Catering;
```

```
MODERN - backend/api/catering.php
1 <?php
2
3 require_once __DIR__ . '/..../config/cors.php';
4 header('Content-Type: application/json');
5
6 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
7   // Parse JSON from React
8   $Input = json_decode(file_get_contents('php://input'), true);
9
10  $name = $Input['name'] ?? '';
11  $email = $Input['email'] ?? '';
12  $phone = $Input['phone'] ?? '';
13  $eventDate = $Input['eventDate'] ?? '';
14  $guestCount = $Input['guestCount'] ?? '';
15  $package = $Input['package'] ?? '';
16  $message = $Input['message'] ?? '';
17
18  // VALIDATION
19  if (empty($name) || empty($email) || empty($phone) ||
20      empty($eventDate) || empty($guestCount) || empty($package)) {
21    http_response_code(400);
22    echo json_encode(['error' => 'Please fill in all required fields']);
23    exit;
24  }
25
26  // Prepare email
27  $to = 'f32ee@localhost';
28  $subject = 'New Catering Inquiry - ' . $package;
29
30  $body = "New Catering Inquiry\n\n";
31  $body .= "Contact: $name ($email, $phone)\n";
32  $body .= "Event: $eventDate | Guests: $guestCount\n";
33  $body .= "Package: $package\n";
34  $body .= "Message: $message\n";
35
36  $headers = "From: $email\r\nReply-To: $email\r\n";
37
38  // Send email using PHP mail() function
39  if (mail($to, $subject, $body, $headers)) {
40    echo json_encode([
41      'success' => true,
42      'message' => 'Your inquiry has been sent! We will contact you soon.'
43    ]);
44  } else {
45    http_response_code(500);
46    echo json_encode(['error' => 'Failed to send email']);
47  }
48 } else {
49  http_response_code(405);
50  echo json_encode(['error' => 'Method not allowed']);
51 }
52 ?>
```

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

	<p>management.</p> <p>Usage:</p> <ul style="list-style-type: none"> - 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	<p>Purpose: Restaurant menu catalog.</p> <p>Usage:</p> <ul style="list-style-type: none"> - 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	<p>Purpose: Shopping cart persistence for checkout process.</p> <p>Usage:</p>		Yin Yu / Jun Le

	<ul style="list-style-type: none"> - 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	<p>Purpose: Customer review and rating system with control.</p> <p>Usage:</p> <ul style="list-style-type: none"> - 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	<p>Purpose: Order transaction history and tracking.</p> <p>Usage:</p> <ul style="list-style-type: none"> - Records all completed orders with customer details 		Yin Yu / Jun Le

	<ul style="list-style-type: none"> - 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	<p>Purpose: Order details for each order.</p> <p>Usage:</p> <ul style="list-style-type: none"> - 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%