

No.	Contents	Page no.
1.	Software Architecture	2-4
	1.1. Overview.....	.....2
	1.2. Tools and Technologies.....	.....2-3
2.	User Interface Mockups	3-15
	2.1. Customer Views.....	.....4-9
	2.1.1. Account Creation.....	.....4-6
	2.1.2. Restaurant Overview.....	.....7
	2.1.3. Menu View.....	.....7
	2.1.4. Shopping Cart.....	.....8
	2.1.5. Payment Method.....	.....8
	2.1.6. Order History.....	.....9
	2.2. Restaurant Views.....	....10-15
	2.2.1. Account Creation.....	...10-12
	2.2.2. Menu Management.....	...13-14
	2.2.3. Order Management.....	.....15
3.	Implementation Plan	16-17
	Phase 1: Database Setup.....	.....16
	Phase 2: Backend Development.....	.....16
	Phase 3: Frontend Development.....	.....16
	Phase 4: Testing.....	.....17
	Phase 5: Deployment.....	.....17
4.	Challenges	17-18
	Challenge 1: Synchronizing Backend And Frontend Communication...	.....17
	Challenge 2: Integration Across Layers.....	.....18
	Challenge 3: Debugging Issues Across Layers.....	.....18
	Challenge 4: Developing Soft Skills.....	.....18
5.	References.....	.....19

# 1. Software Architecture

## 1.1. Overview

The project involves implementing a simplified food and beverage delivery platform, *Lieferspatz*. The platform will allow restaurants to manage their menus and receive orders, while customers can browse restaurants, place orders and track their order statuses.

The platform uses a three-layer architecture to keep the design simple, scalable and easy to maintain:

1. Frontend: HTML, CSS and JavaScript provide an intuitive and responsive user interface, ensuring seamless interaction for customers and restaurants.
2. Backend: Flask (Python) is used to handle business logic and data flow, exposing RESTful APIs for client-server communication
3. Database: SQLite serves as the storage layer, managing persistent data like user accounts, restaurants menus and orders.

## 1.2. Tools and Technologies

Component	Technology	Purpose	Reasons
Frontend Server	HTML, CSS, JavaScript	Serves HTML pages, manages client-side interactions, and sends requests to the backend server.	These tools are standard for building websites and provide flexibility for making the platform look good and work smoothly.
Backend Server	Flask (Python)	Implements business logic, processes user requests, and manages data flow between components.	Flask is lightweight, beginner-friendly, and has extensive community support for creating RESTful services
Database	SQLite	Store user, restaurant, menu and order related data.	SQLite is embedded, easy to set up, and sufficient for small-scale projects or prototypes where scalability requirements are minimal.
Frontend Frameworks	HTML, CSS, JavaScript	Provides structure, design, and interactive elements for the user interface.	These are standard tools for frontend development and ensure compatibility with various devices and browsers.
Inter-Server Communication	AJAX (via fetch API)	Facilitates RESTful communication between the frontend and backend.	The Fetch API is modern, efficient, and integrates well with JavaScript, making communication seamless and easy to maintain.

## Key Functionalities

### Frontend (HTML, CSS, JavaScript):

- Serves static pages for:
  - Registration/Login for both customers and restaurants
  - Viewing nearby restaurants and their menus.
  - Managing the cart, including adding/removing items.
- Sends user data (e.g., registration and login forms) to the backend via AJAX (fetch()) requests.

### Backend (Flask with SQLite):

- Handles API endpoints for:
  - User and restaurant account creation, login and management.
  - CRUD operations for restaurant menus and items.
  - Order management, including updating order statuses.
- Communicates with the SQLite database to store and retrieve information.

### Database (SQLite):

- Stores the following entities:
  - Users: Details of customers and restaurants.
  - Restaurants: Menus, opening hours, and delivery areas (postcodes).
  - Orders: Order details, statuses, items and customer information.
  - Wallets: Virtual wallets for simplified payment management.

## 2.User Interface Mockups

The UI wireframes presented below illustrate the planned design and layout for the *Lieferspatz* food and beverage delivery platform. These mockups provide a clear visual representation of how the application will look and function. They were created using Balsamiq, a tool designed for rapid wireframe creation.

The purpose of these mockups is twofold:

1. To guide the development process by ensuring alignment between the design and functionality requirements.
2. To prioritize usability and intuitive interaction for both customer and restaurant users.

By outlining key components and their interactions, the wireframes aim to reduce ambiguity, streamline implementation, and create a seamless user experience.

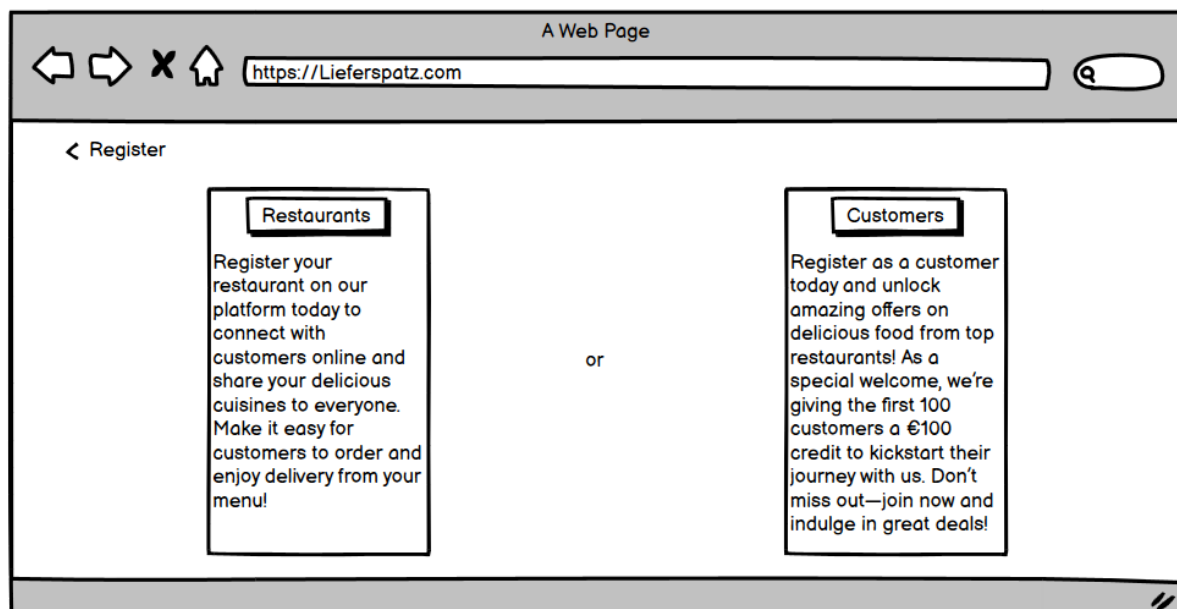
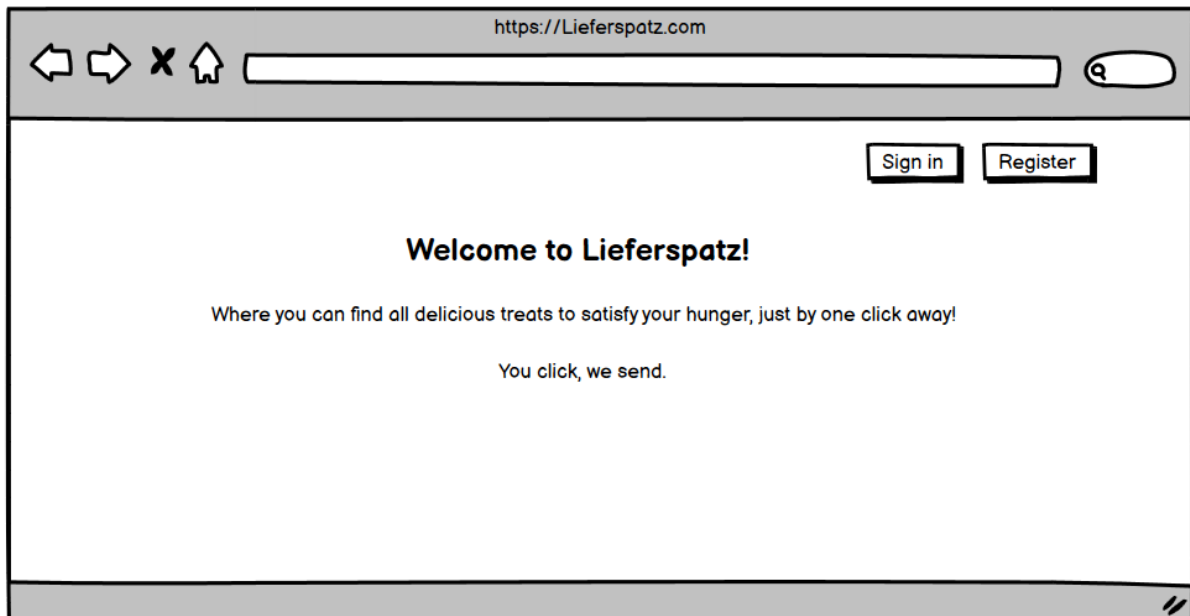
Below are the wireframes organized under specific use-case scenarios. Each section includes a description of the screen's purpose, functionality, and user interactions.

## 2.1. Customer Views

These wireframes outline the key interfaces and interactions a customer encounters within the *Lieferspatz* platform, ensuring a smooth and intuitive user experience.

### 2.1.1. Account Creation

This screen enables customers to register for the *Lieferspatz* platform.



A Web Page

https://Lieferspatz.com

< Register as a Customer

First Name	Last Name	Email Address
<input type="text"/>	<input type="text"/>	<input type="text"/>
Street and House Number		Password
<input type="text"/>		<input type="text"/>
PLZ	City	Confirmation of Password
<input type="text"/>	<input type="text"/>	<input type="text"/>

**Register**

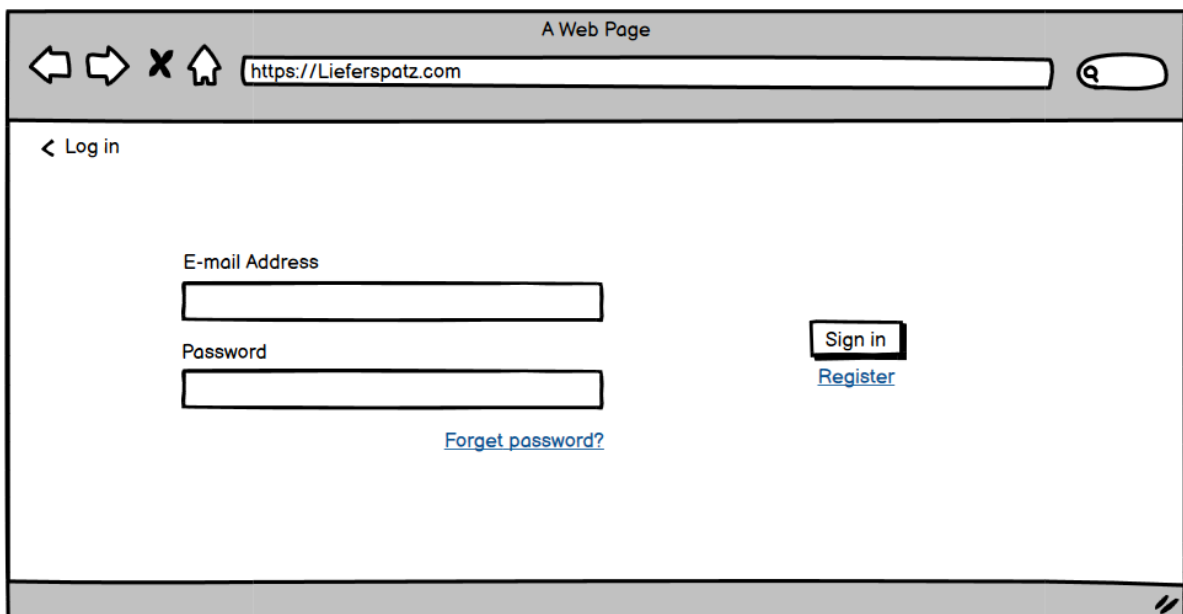
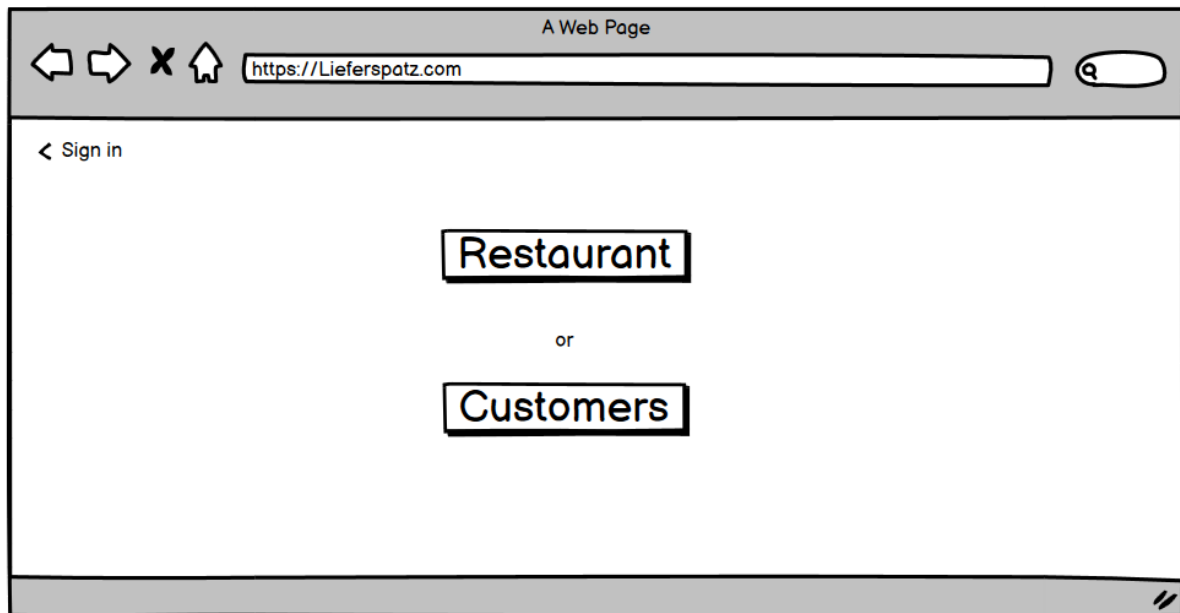
A Web Page

https://Lieferspatz.com

< Register

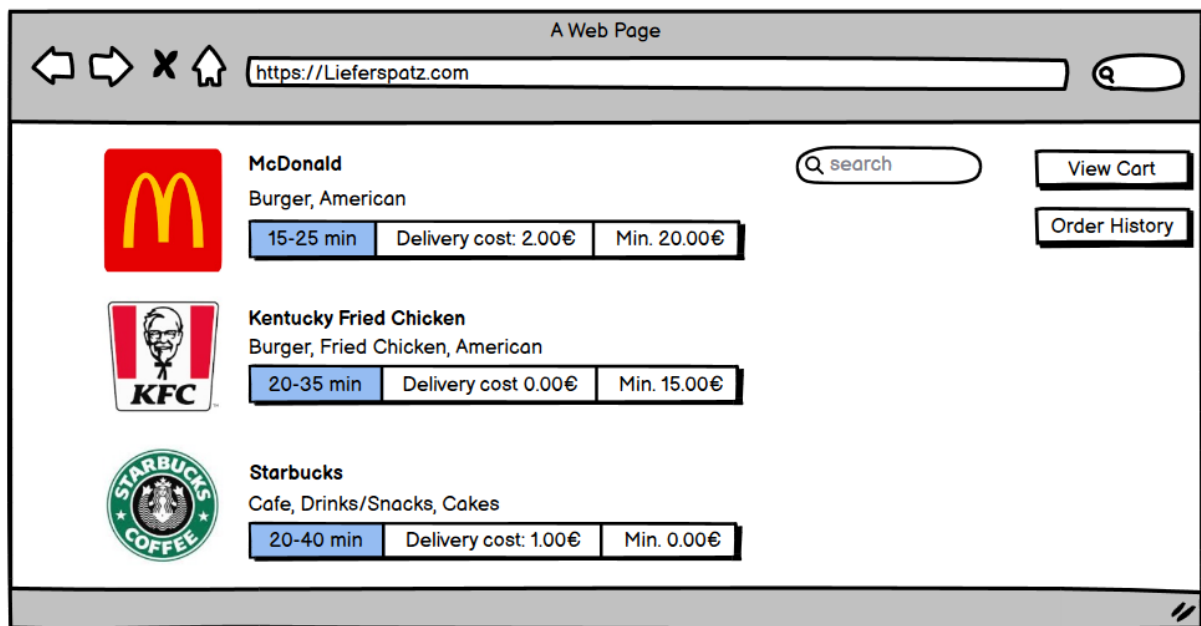
**Email is in use.**

The email address is already in use. Please use a different email address.



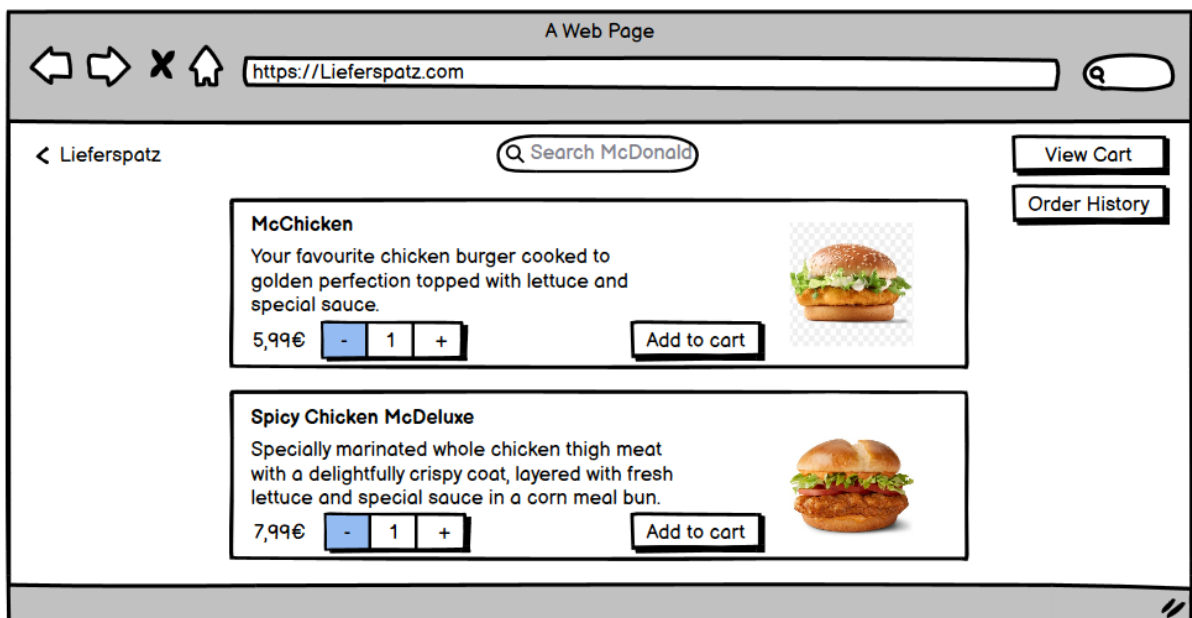
1. This wireframe represents the customer registration page. It includes input fields for essential details like name, address, email, and password, alongside a "Register" button.
2. Optional features such as a login link for returning users and error message placeholders are also present.
3. Upon successful registration, the user is redirected to the login page or restaurant overview.

## 2.1.2 Restaurant Overview



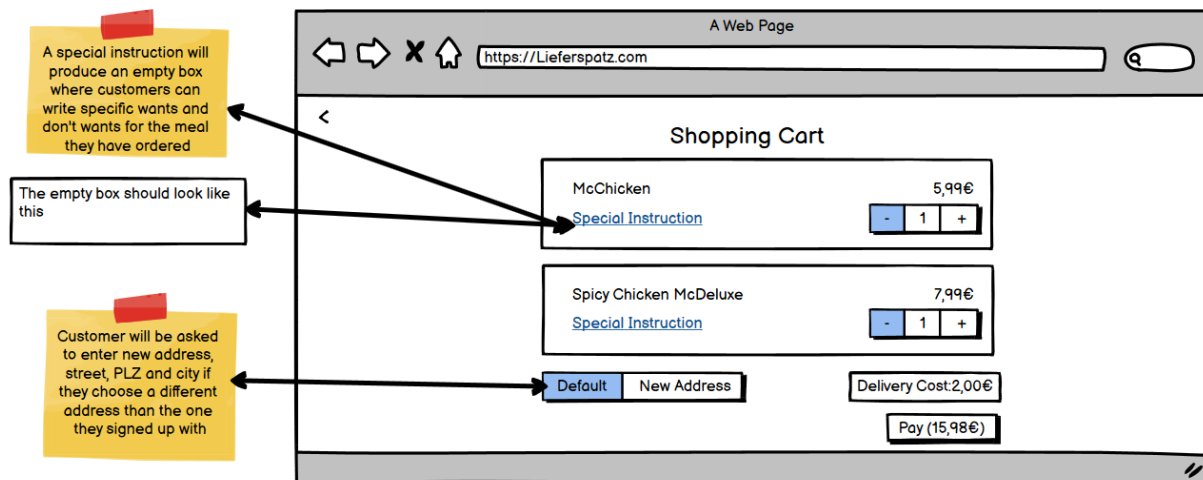
1. This screen showcases a list of restaurants that deliver to the customer's postcode.
2. Each restaurant entry displays the name, a short description, and an estimated delivery time.
3. Optional filters help refine the list, and selecting a restaurant directs the customer to its menu page.

## 2.1.3. Menu View



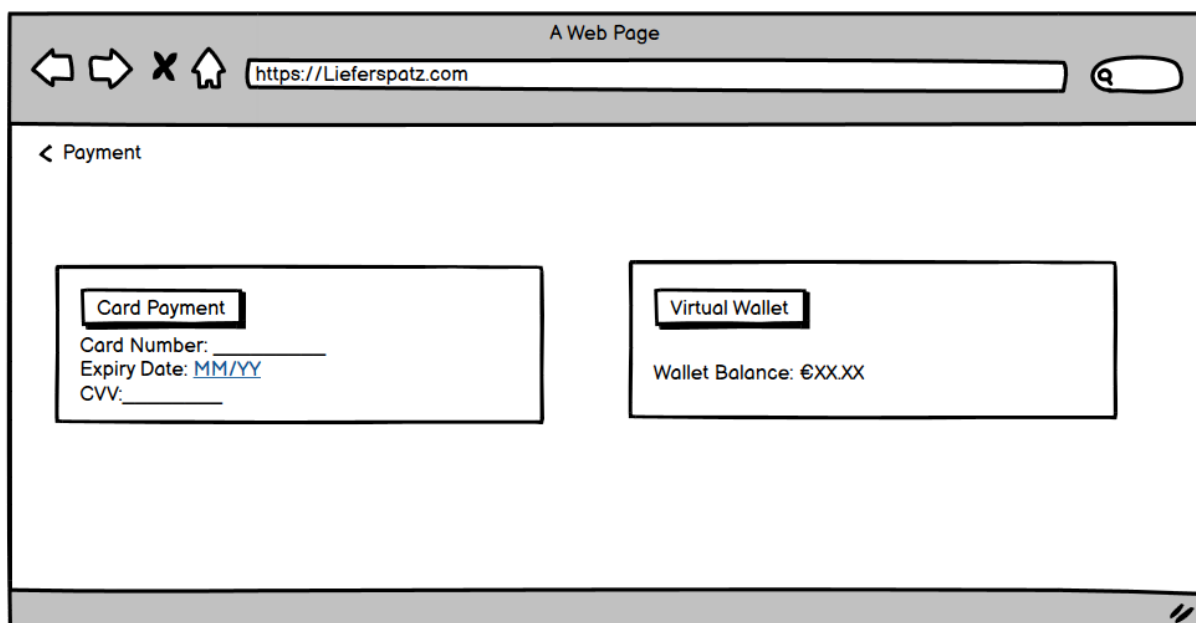
1. The menu view displays all available items for a selected restaurant.
2. Each item includes its name, description, price, and optional image.
3. Users can adjust quantities and add items to their cart.
4. A "View Cart" link provides quick navigation to the shopping cart page.

## 2.1.4. Shopping Cart



1. This wireframe highlights the shopping cart page, summarizing all items added, their quantities, and total costs.
2. Customers can input special instructions and finalize their orders using the "Pay" button.
3. They can also modify the cart by removing items before proceeding.

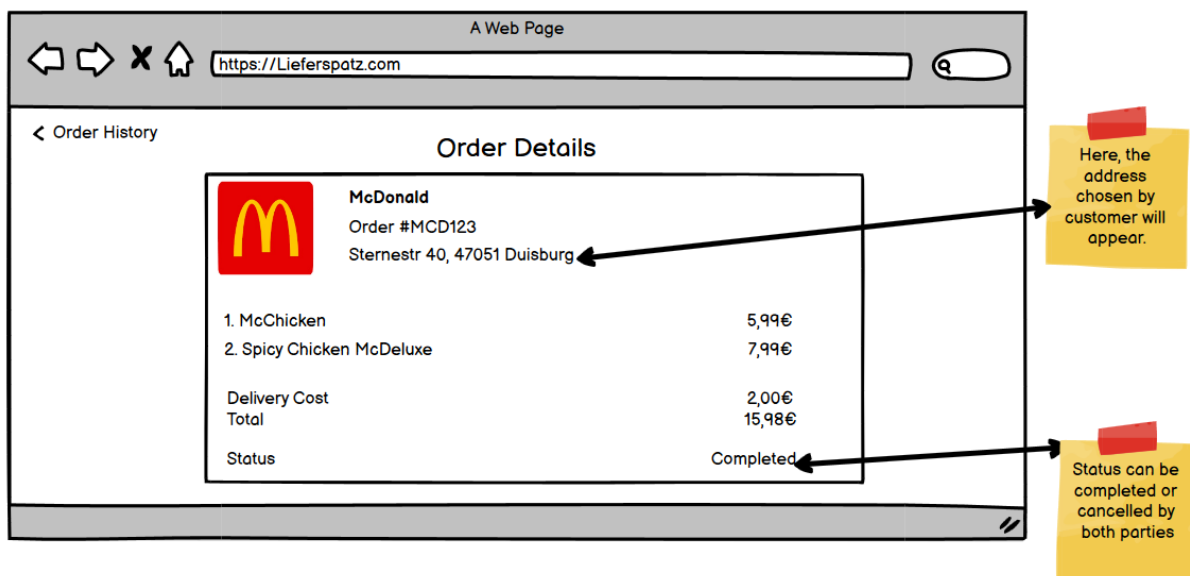
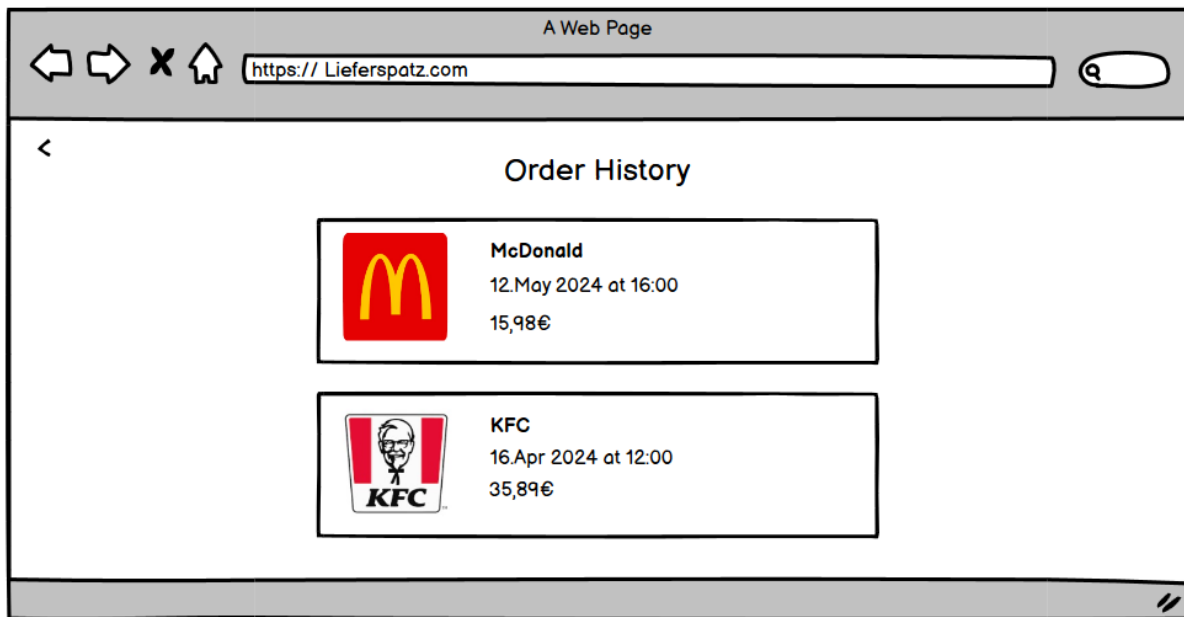
## 2.1.5. Payment Method



1. This wireframe represents the payment method, where customers can complete their transactions.
2. The page provides two options: Card Payment and Virtual Wallet.
3. Customers can select their preferred method to proceed the checkout process.
4. The layout ensures simplicity and usability, focusing on clear instructions and minimal distractions.



### 2.1.6. Order History

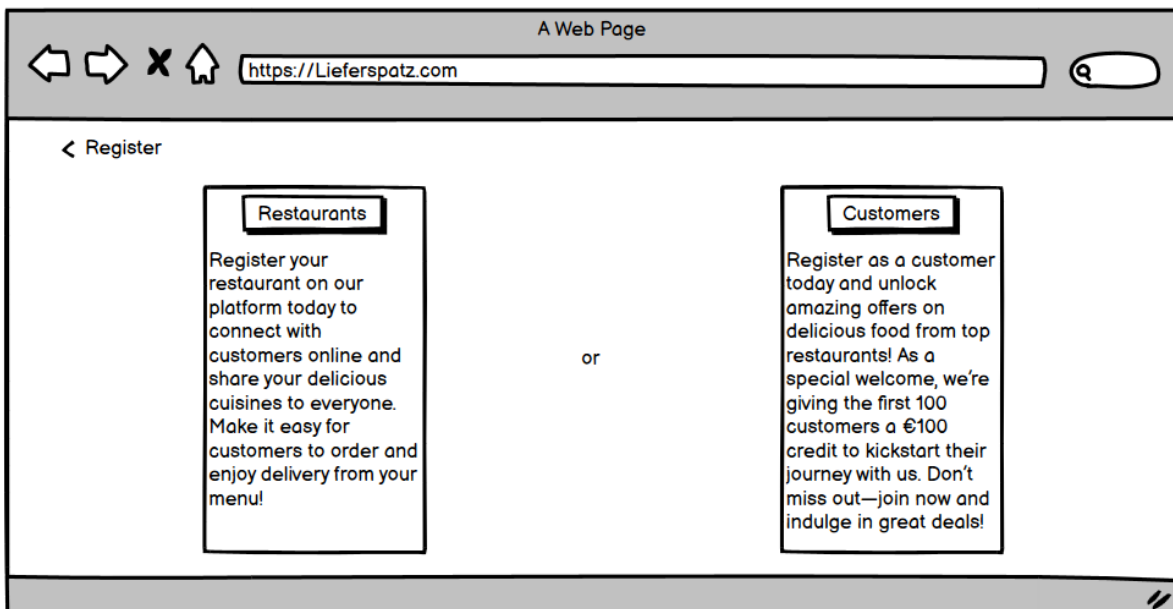
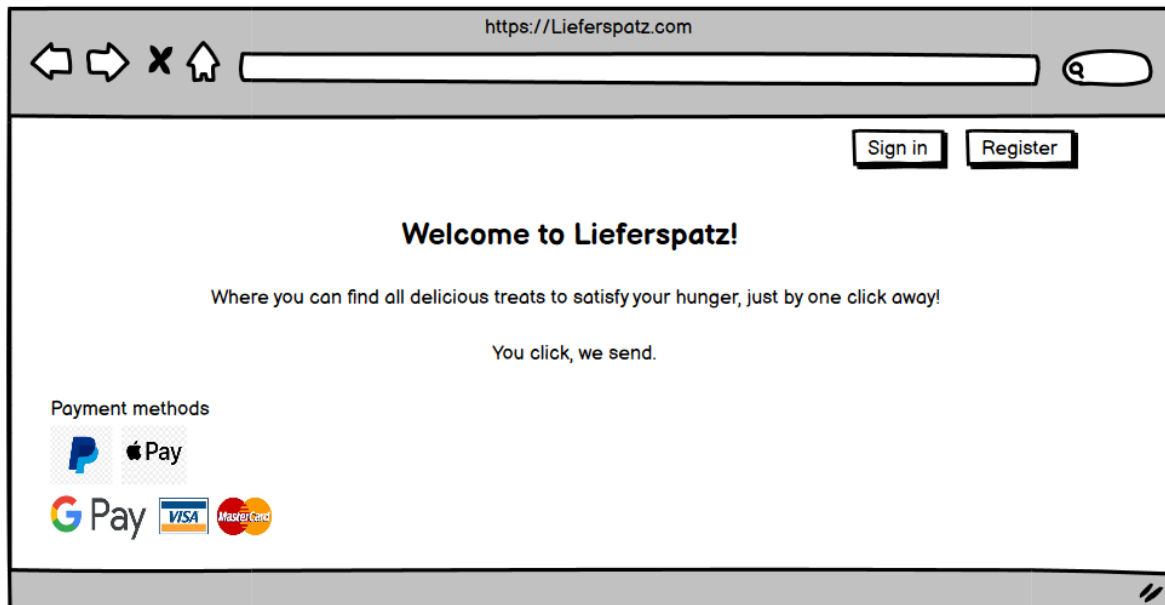


1. This screen shows the customer's past orders, sorted by date, with details such as order ID, total price, and status.
2. Each entry includes a link to view the full order details for further review or tracking, such as if the order is completed or cancelled.

## 2.2. Restaurant Views

These wireframes illustrate the key interfaces designed for restaurant owners to manage their accounts, menus, and incoming orders on the *Lieferspatz* platform. Each screen is optimized for ease of use and efficient management.


### 2.2.1. Account Creation



A Web Page

https://Lieferspatz.com

< Register as Restaurant

Restaurant Name	Description	 Image of Restaurants
Street	Operation Hours	
PLZ	From <input type="text"/> Until <input type="text"/>	
City	Telephone Number	
E-mail Address	Password	

Register

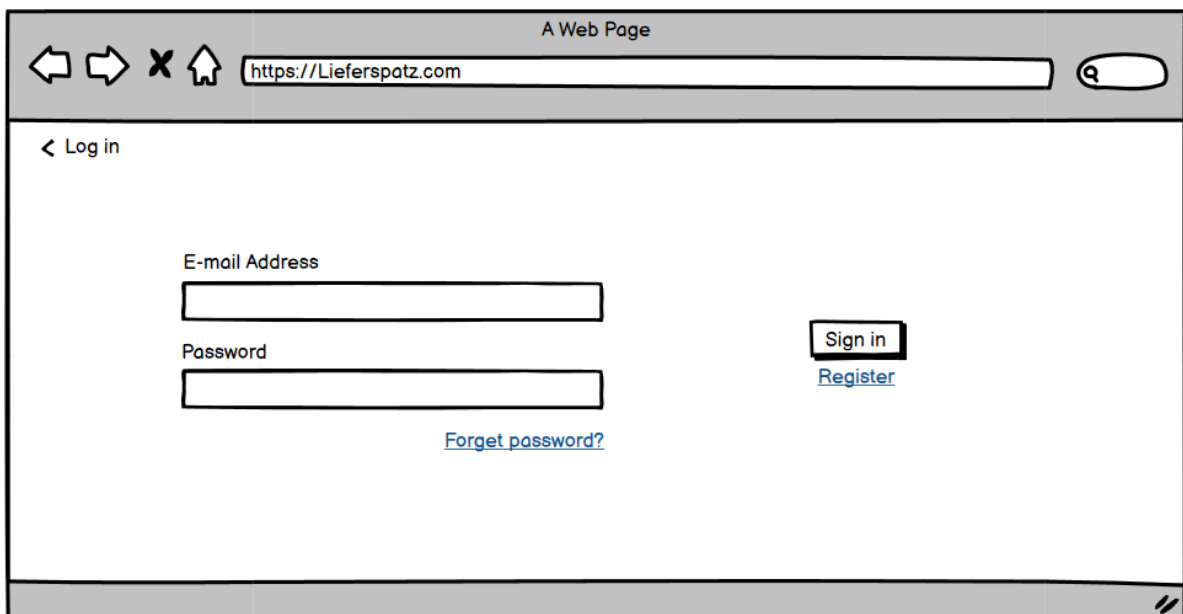
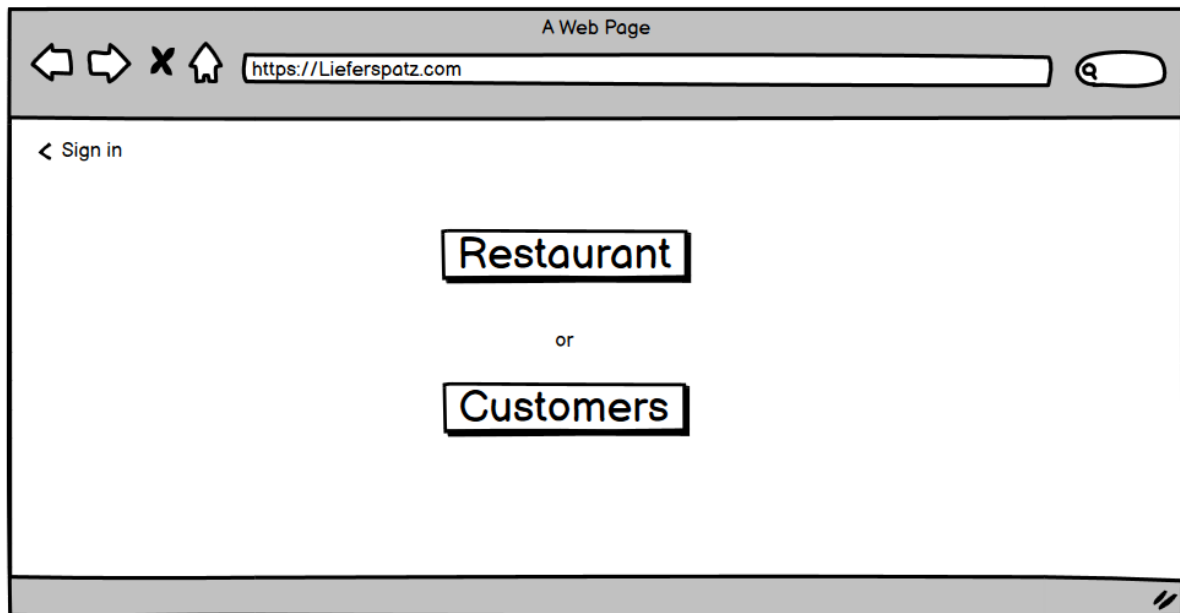
A Web Page

https://Lieferspatz.com

< Register

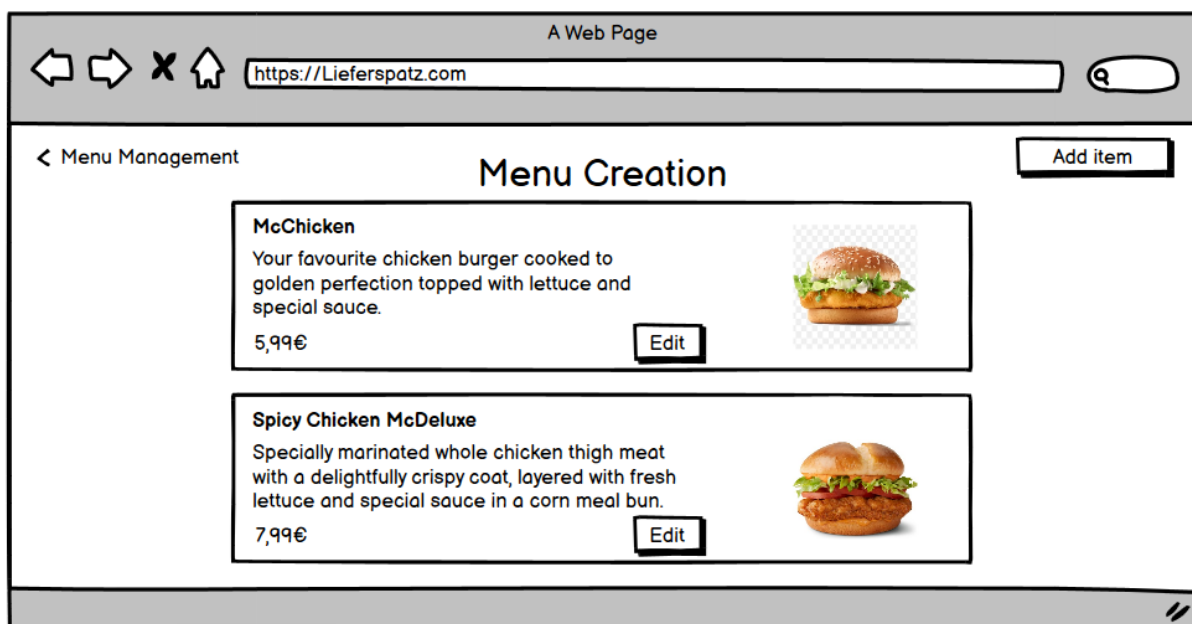
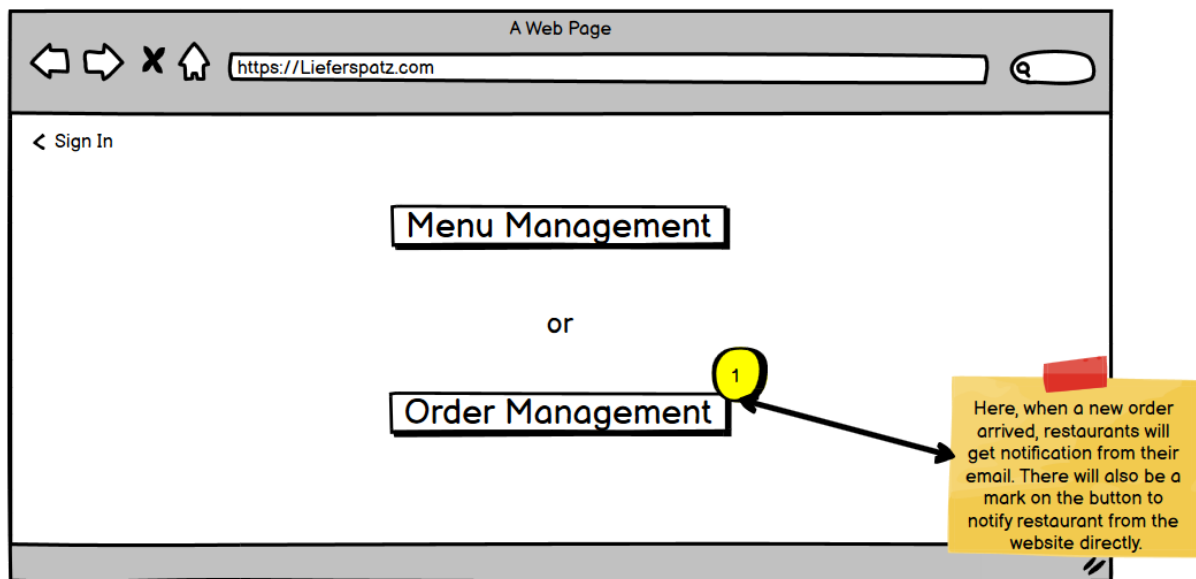
Email is in use.

The email address is already in use. Please use a different email address.



1. This wireframe represents the restaurant registration page.
2. It includes input fields for essential details such as the restaurant's name, restaurant's image, address, description, telephone number, operation hours, and password.
3. A "Register" button completes the process, redirecting the restaurant owner to the login page or dashboard upon successful registration.

## 2.2.2. Menu Management



A Web Page

https://Lieferspatz.com

< Add item

Name of Product

Description of Product

Price

Image of Product

Save Changes

A Web Page

https://Lieferspatz.com

< Edit

Name of Product

McChicken

Description

Your favourite chicken burger cooked to golden perfection topped with lettuce and special sauce.

Price

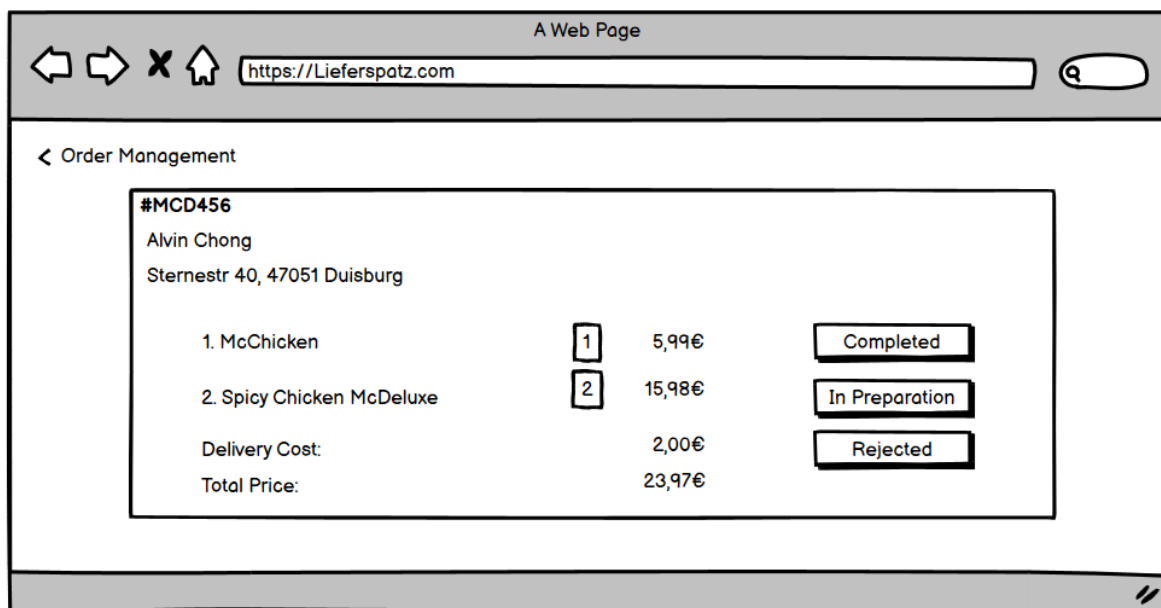
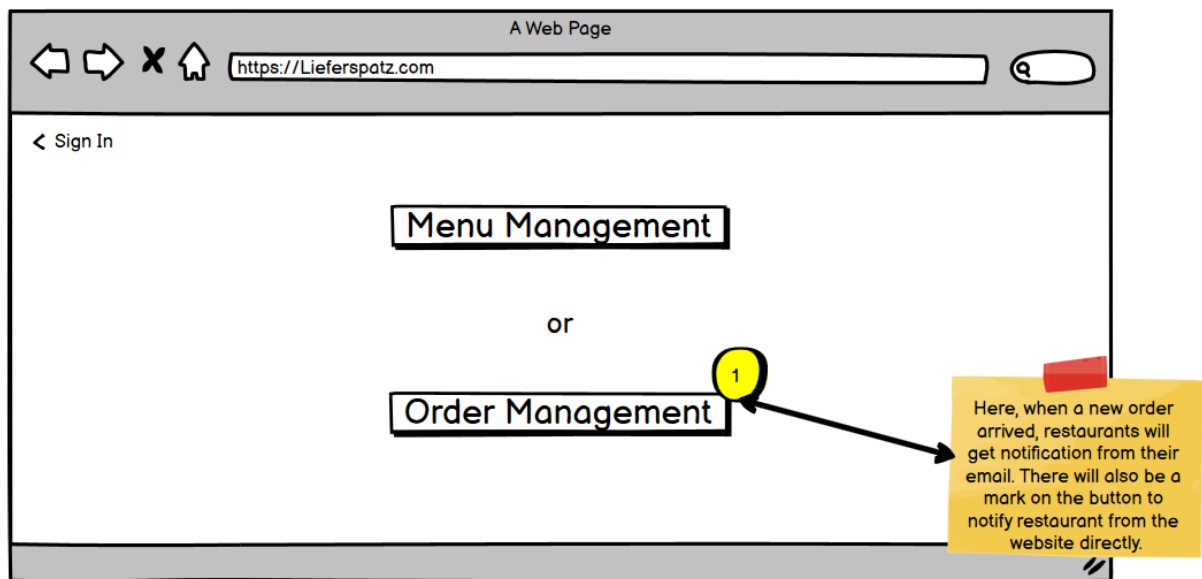
5,99€

Image of Product

Remove Save Changes

1. This screen outlines the menu management interface, allowing restaurant owners to maintain their offerings.
2. It features a list of current menu items with options to add new items, edit existing ones, or delete them.
3. Fields for item name, description, price, and optional images are provided.
4. Changes are saved dynamically with a "Save Changes" button, ensuring the menu stays up to date.

### 2.2.3. Order Management



1. This wireframe showcases the order management dashboard, displaying a list of current orders along with key details such as customer name, address, ordered items, quantities, and total price.
2. Action buttons enable the restaurant to update order statuses to "Completed," "In Preparation", or "Rejected" in real-time, ensuring efficient handling of customer requests.

## 3.Implementation Plan

### Phase 1: Database Setup

- Design the Database Schema:
  - Use SQLite to define relationships between users, restaurants, orders and items.
  - Entities:
    - Restaurants: 10 restaurants, each offering 10 menu items.
    - Customers: 5 customers, each with 2 completed orders.
    - Orders: Include status, customer details, restaurant details and items.
    - Wallets: Track simplified virtual payments.
- Populate the Database:
  - Insert sample data for initial testing and validation.
  - Test relationships with JOIN queries to ensure schema integrity.

### Phase 2: Backend Development

- Framework: Flask (Python)
- Implement RESTful APIs:
  - /register (POST): Register a new user or restaurant.
  - /login (POST): Authenticate users.
  - /restaurants (GET): Fetch nearby open restaurants for a given postcode.
  - /menu/<restaurant\_id> (GET): Retrieve menu items for a specific restaurant.
  - /orders (POST): Place a new order.
  - /orders/<order\_id> (PATCH): Update the status of an existing order.
- Database Integration:
  - Use SQLite to store and retrieve data for each endpoint.
  - Validate proper CRUD operations for users, restaurants, menus and orders.

### Phase 3: Frontend Development

- Framework: HTML, CSS, JavaScript
- UI Design:
  - Use custom CSS to design a responsive, user friendly interface.
  - Employ CSS Flexbox and Grid for layout and responsiveness.
  - Consider using CSS variables for consistency in styling (e.g., colours, spacing, fonts).
- Pages to Implement:
  - Registration and Login Forms for customers and restaurants.
  - Homepage displaying nearby restaurants based on postcode.
  - Restaurant Menu with an interactive cart.
  - Order History for customers and order management for restaurants.
- Frontend Functionality:
  - Use fetch() to send HTTP requests to the Flask backend.
  - Dynamically update UI elements based on JSON responses from the backend.



## Phase 4: Testing

- API Testing:
  - Use Postman to verify all API endpoints:
    - Ensure proper request/response handling for successful and invalid inputs.
  - Validate edge cases, such as invalid logins or orders with unavailable items.
- Frontend Testing:
  - Perform user acceptance testing (UAT) with sample data to simulate common workflows.
  - Verify smooth integration between the UI and backend APIs.
- Database Validation:
  - Test data consistency with SQL queries to ensure correct relationships and results.

## Phase 5: Deployment

- Local Deployment:
  - Run the application locally during the oral exam:
    - Flask backend on port 5000.
    - Static frontend served on port 3000 using a simple HTTP server.
  - Test end to end functionality with sample data.

## 4.Challenges

### Challenge 1: Synchronizing Backend and Frontend Communication

Ensuring consistent and real time communication between the backend and frontend can be challenging, especially for dynamic applications like *Lieferspatz*.

Solution:

- Implement polling for periodic updates or WebSocket for real-time bidirectional communication, depending on the application's requirements for responsiveness and scalability.
- Thorough testing and debugging of API endpoints to ensure data integrity during transmission.

## **Challenge 2: Integration Across Layers**

Seamlessly integrating the database, backend, and frontend requires careful planning and execution to avoid inconsistencies or failures.

Solution:

- Maintain consistent and well documented API structures.
- Schedule regular team reviews to ensure all components align with project goals.
- Use testing framework to verify end to end functionality.

## **Challenge 3: Debugging Issues Across Layers**

Identifying and resolving issues across interconnected components (frontend, backend, database) can be time invasive.

Solution:

- Utilize debugging tools such as browser developer tools, logging libraries and testing utilities.
- Research debugging best practices and systematically apply them during development.
- Conduct regular debugging sessions to pre-emptively identify and resolve potential issues.

## **Challenge 4: Developing Soft Skills**

Personal growth and adaptability are crucial when tackling new and unfamiliar challenges.

Solution

- Emphasize teamwork, open communication, and resilience during the development process.
- Foster a culture of perseverance and adaptability by viewing challenges as learning opportunities.
- Celebrate milestones to maintain motivation and a positive team dynamic.

## 5.Reference

- Schafer, C.(n.d.). *Flask Tutorials-Part 1 [Video]*. YouTube. [https://youtu.be/NFE\\_avBNGfU](https://youtu.be/NFE_avBNGfU)
- Tech with Tim. (n.d.). *Flask Crash Course [Video]*. YouTube. <https://youtu.be/ZkbWCYXQTYQ>
- Programming with Mosh. (n.d.). *Flask Tutorial for Beginners [Video]*. YouTube. <https://youtu.be/Uq7TkegTXRU>
- CIAT. (n.d.). *HTML, CSS and JavaScript: Building Blocks of Web Development*. CIAT Blog, <https://www.ciat.edu/blog/html-css-javascript/#:~:text=HTML%20defines%20the%20structure%20of,and%20imperative%20styles%20of%20programming>
- Programming with Mosh. (n.d.). *Why Learn HTML, CSS and JavaScript? [Video]*. YouTube, <https://www.youtube.com/watch?v=MsnQ5ueplaE>