Pizza Ordering System

1. Introduction

This document outlines the requirements and specifications of the system for a pizza restaurant. The system is mainly used for customer orders ranging from pizza customization with different toppings, drinks selection, bill generation, and preparation time estimation for an order. The system is developed under a microservices architecture to ensure the system is scalable, easy to maintain, and modular. This document is formulated to give a clear description of how the system is supposed to work from the functional requirements perspective, the system architecture, and technical design to serve as a guide for the implementation team.

1.1. Purpose of the Document

The main objective of this document is to describe the following aspects; the functional and non-functional requirements, system architecture, and technology stack to be used in developing the pizza restaurant system. The other aim is to outline these aspects for the development team, which is comprised of myself. It also aims to give the stakeholders the context of the whole project, its objectives, and the deliverables.

1.2. Scope of the System

The system focused on the following activities:

1. Creating customer orders that allow them to customize a pizza with different toppings of choice and can take different sizes and a choice of drinks.
2. Process orders in real-time, generate bills, and prepares each order's time estimate.
3. Offers an administrator platform to update the menu, toppings, drinks, and price adjustments and an order monitoring facility.
4. System integration with other external systems, payment gateways, and notifications services.

The system is developed using a Microservices architecture where all the functions are divided into smaller independent components that maintain both synchronous and asynchronous data exchange. This helps in future scalability, modification, and addition of more features.

1.3. Objectives

The main goal is to make the pizza ordering process much simpler for customers, and also improve order management performance for the restaurant staff. Specifically, we aim to:

1. Allow the customers to quickly and freely make changes to their pizza orders (toppings, drinks, sizes).
2. Calculate the final bill (taxes, discounts) and display on a screen detailed order summary.
3. Offer live tracking of the order, with an expected time before completion.
4. Provide administrators with an easy way to control a menu, check orders and analyze the overall performance.
5. Manage all aspects of the ordering process efficiently.
6. Make certain that the system can handle multiple concurrent orders, scales well and minimize downtime.

1.4. Stakeholders

The key stakeholders involved:

1. **Customers:** End consumer who is going to order and customize pizzas.
2. **Restaurant Staff:** The people who handled the orders, taking them off order tickets and doing preparation work.
3. **Development Team:** Myself as I will design, develop, and maintaining the system.

2. System Overview

2.1. System Description

The backend of the pizza restaurant system is written with the possibility of scale as the main goal, being a service-oriented platform, it manages complete customer orders. I plan to implement a microservices architecture where each service is responsible for a defined task like ordering/payment/menu management etc. In addition, I am taking a modular approach which will help to grow fast and maintain the system better.

The system will be built to give customers the ability to create their own unique pizza orders, they can choose among available pizza sizes, toppings, and drinks. Once an order is submitted, the system processes it, calculates the total cost, and estimates a preparation time. It will also give the restaurant staff the ability to control the menu, track orders in real-time, and monitor the health of the system.

The methods of communication done with services as synchronous and asynchronous. I will use REST API requests to allow real-time interactions, and message queues for background tasks, particularly for tasks such as order notifications. Sending payments and adding external services as well.

2.2. Core Features

1. **Pizza Customization:** When ordered, the customer can get their unique pizza; Choose from size, type of crust, toppings, and sauces. Customers can also choose their preferred drinks.
2. **Real-Time Order Management:** As orders happen, the system processes them in real-time and provide instant comments on their full billings and estimated preparation time.
3. **Bill Generation and Payment Processing:** The system calculates the amount for the purchase order, this includes addons, taxes and deductions. And takes care of the payment securely through third-party payment gateway.
4. **Menu Management:** Adding new items to the menu and updating them regularly is extremely important, the staff additionally can change prices and items availability, to keep the menu up to date.
5. **Order Tracking and Notifications:** Both customers and restaurant staff will be able to track the order status and will get notified when it changes.
6. **Admin Dashboard:** designing an admin panel for the staff of a restaurant to take advantage of.
7. **Scalability and Availability:** the system architecture is being prepared to cope with high volumes of data/simulation events, for certain features.

2.3. Constraints and Assumptions

I identified these constraints and assumptions before developing the system:

2.3.1. Constraints

1. **Network Latency:** Since I will be depending on external services for payments, any delay that will happen in the network might affect the performance of the related services.
2. **Database Performance:** The system should have fast and steady database query functionality, particularly in peak periods. This means I have to index the data properly and implement query optimization techniques.
3. **External Service Integration:** The system will have a dependency on services being provided by a third party for some functionalities. These external systems might

suffer from downtime or performance issues, which will affect the overall system performance.

3.2.1. Assumptions

1. **Good Internet Connectivity:** I presume the restaurant as well as customers will have a good-enough internet connection for using the system.
2. **Regular Menu Updates:** The menu should be always up to date, showing what they have available at any time in terms of toppings, drinks, and other products.
3. **Modular Development:** I am assuming that the system will be built in a micro-service architecture, where each service run and operate independently.

3. System Analysis

3.1. Functional Requirements

This section describes the functional requirements of the system, the roles and permissions requirements, user stories, and key use cases.

3.1.1. User Roles and Permissions

1. **Customer:** They are the end-users of the system. They can browse the menu, customize pizzas, choose their drinks, and order & pay. Customers can track and see the order status and an estimation of preparation time.
2. **Restaurant Administrators/Staff:** They can manage and update the menu, add and remove items, track and manage orders, and see analytical data of the system performance.
3. **System Administrators: They act as the super admins, who have a full access and permissions,**

These roles determine the permissions that are required to be held.

3.1.2. User Stories and Use Cases

The following are the main user stories and use cases that define how different users will interact with the system.

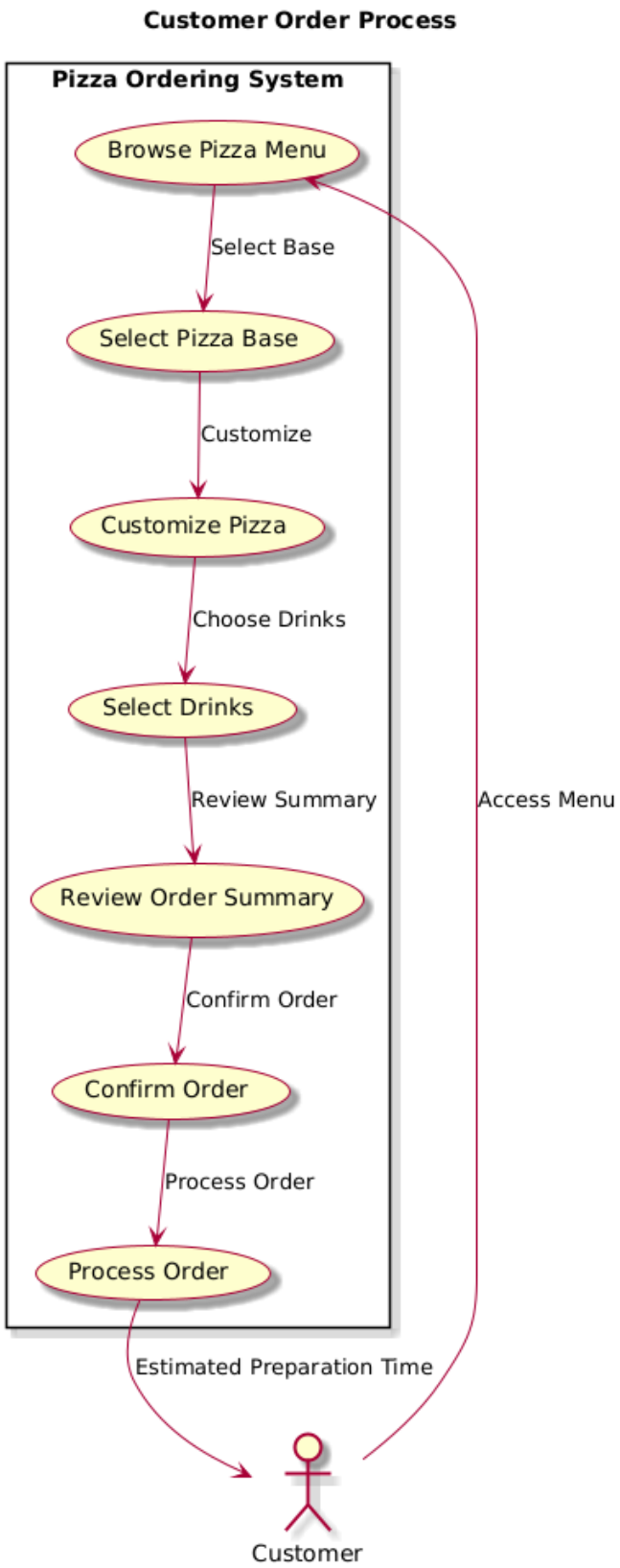
3.1.2.1 Customer Order Process

User Story

As a customer, I want to place an order by selecting my preferred pizza, customizing it with available toppings, and choosing drinks, so that I can complete my meal selection and proceed to payment.

Actor	Customer
Precondition	The customer has accessed the system and logged in (if required).
Steps	1. The customer browses the pizza menu. 2. The customer selects a pizza base (size, crust type, etc.). 3. The customer customizes the pizza by selecting available toppings. 4. The customer selects drinks (optional). 5. The customer reviews the order summary. 6. The customer confirms the order and proceeds to payment. 7. The system processes the order and returns an estimated preparation time.
Postcondition	The order is confirmed, and the customer is notified of the estimated time for completion.

Diagram


Customer Order Process
Pizza Ordering System
Browse Pizza Menu
Select Base
Select Pizza Base
Customize
Customize Pizza
Choose Drinks
Select Drinks
Review Summary
Review Order Summary
Confirm Order
Confirm Order
Process Order
Process Order
Estimated Preparation Time
Access Menu
Customer

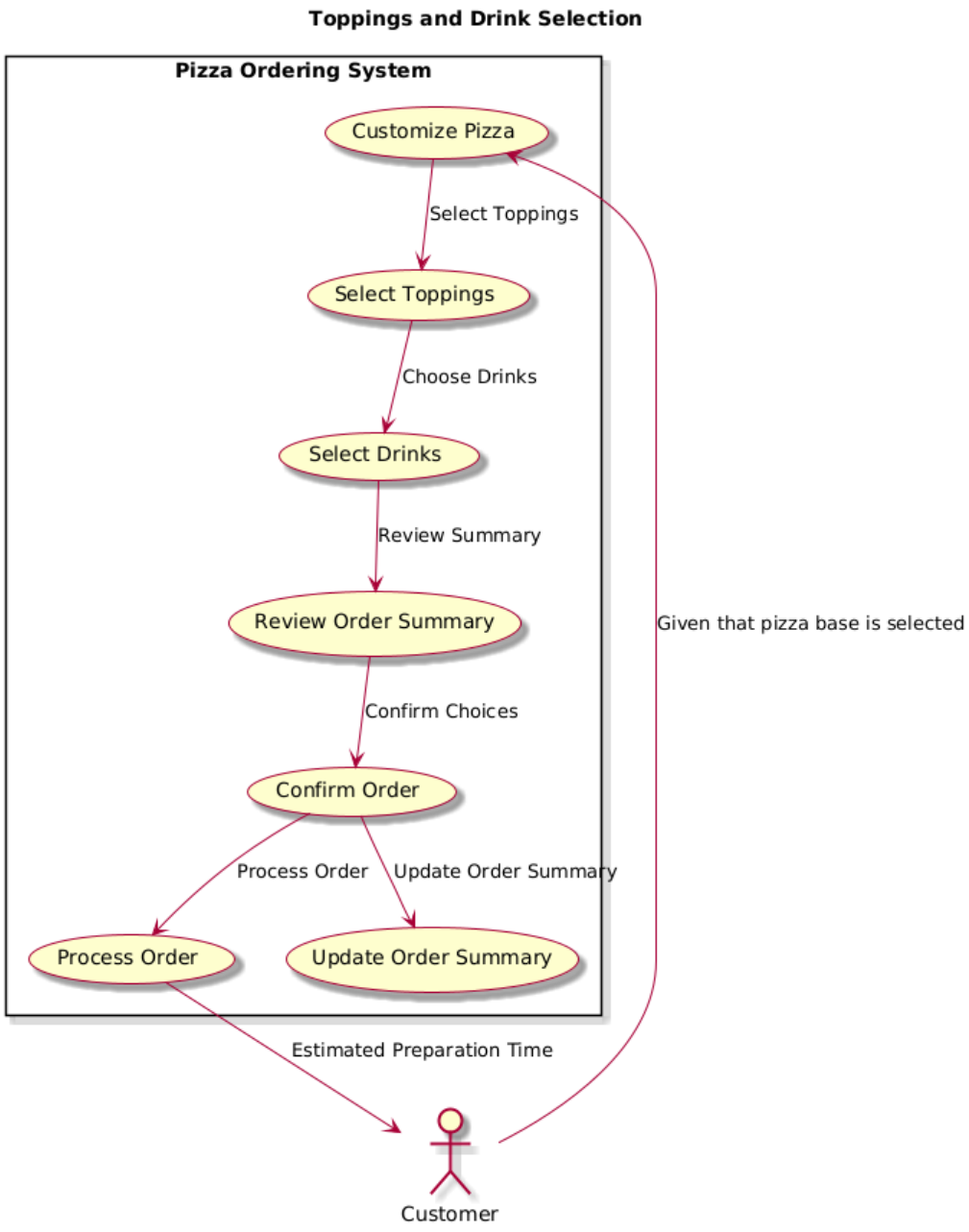
3.1.2.2. Toppings and Drink Selection

User Story:

As a customer, I want to select or modify pizza toppings and drinks so that I can customize my order based on my preferences.

Actor	Customer
Precondition	The customer has selected a pizza base to customize.
Steps	1. The system presents the available toppings (e.g., pepperoni, mushrooms, olives) and drinks. 2. The customer selects the desired toppings and drink. 3. The customer confirms the choices. 4. The system updates the order summary based on the selections.
Postcondition	The selected toppings and drinks are added to the order, and the total cost is recalculated.

Diagram



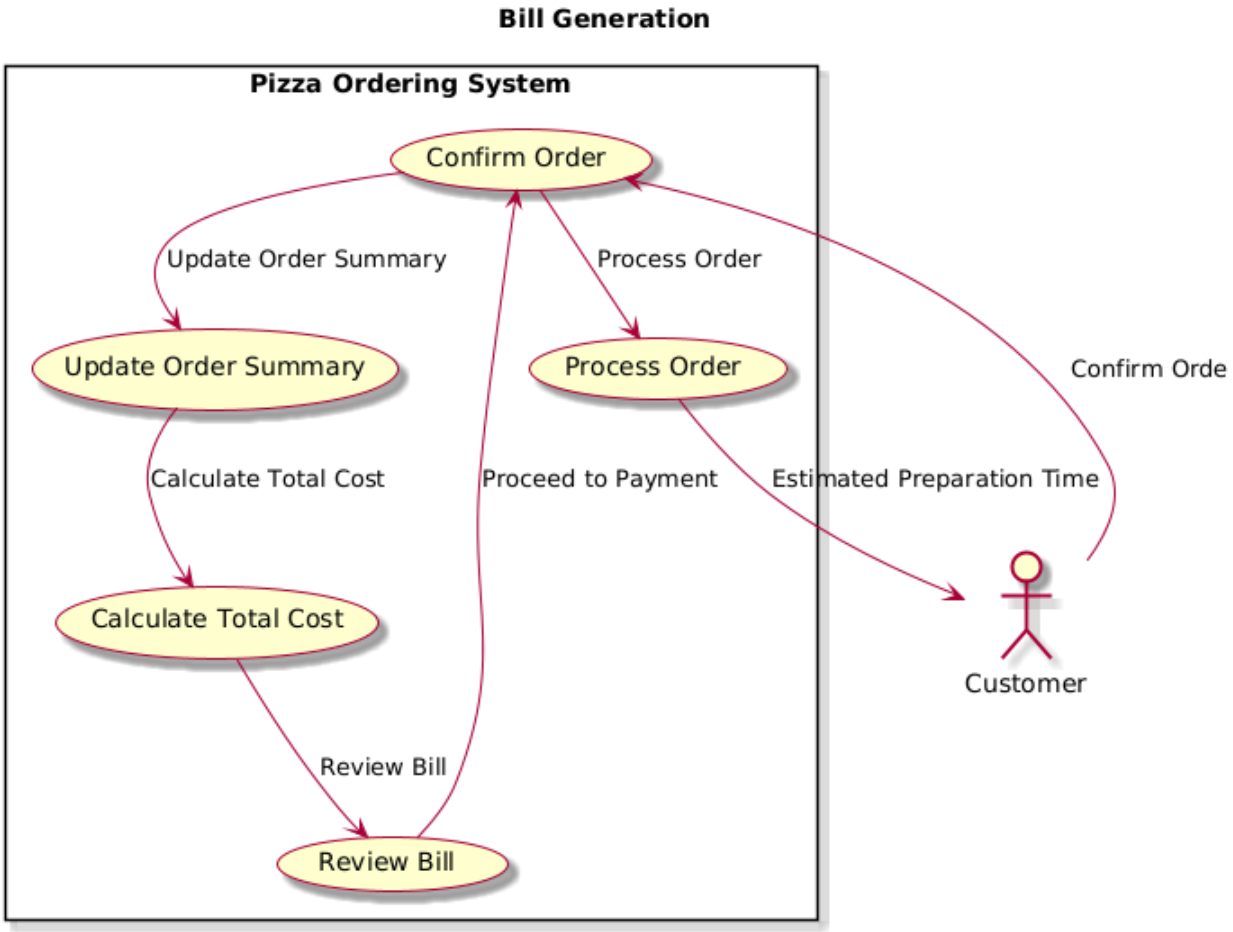
3.1.2.3. Bill Generation

User Story:

As a customer, I want to see the total cost of my order, including all selected toppings and drinks, so that I can review the final price before making a payment.

Actor	Customer
Precondition	The customer has completed the pizza customization process.
Steps	1. The system calculates the total cost based on the pizza base, selected toppings, drinks, and applicable taxes. 2. The customer reviews the bill. 3. The customer proceeds to payment.
Postcondition	The total bill is generated and presented to the customer for confirmation.

Diagram



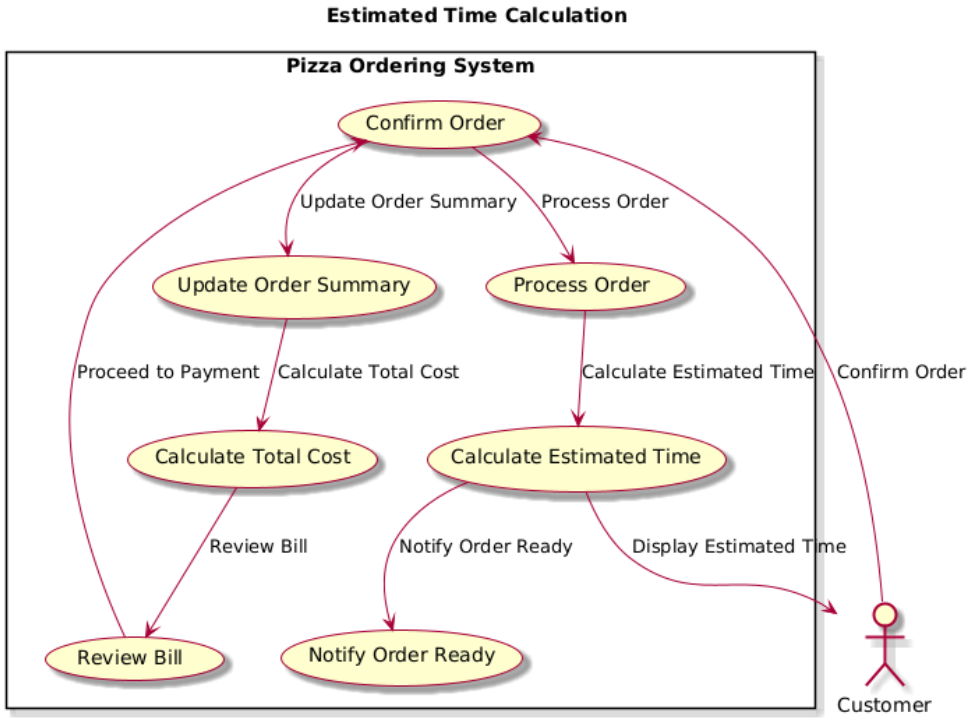
3.1.2.4 Estimated Time Calculation

User Story:

As a customer, I want to receive an estimated time for my order completion so that I know when my pizza will be ready for pickup or delivery.

Actor	Customer
Precondition	The customer has placed an order and the system has confirmed the payment.
Steps	1. The system calculates the estimated preparation time based on the current kitchen capacity and order queue. 2. The system displays the estimated time to the customer. 3. The customer receives a notification when the order is ready.
Postcondition	The estimated preparation time is communicated to the customer.

Diagram



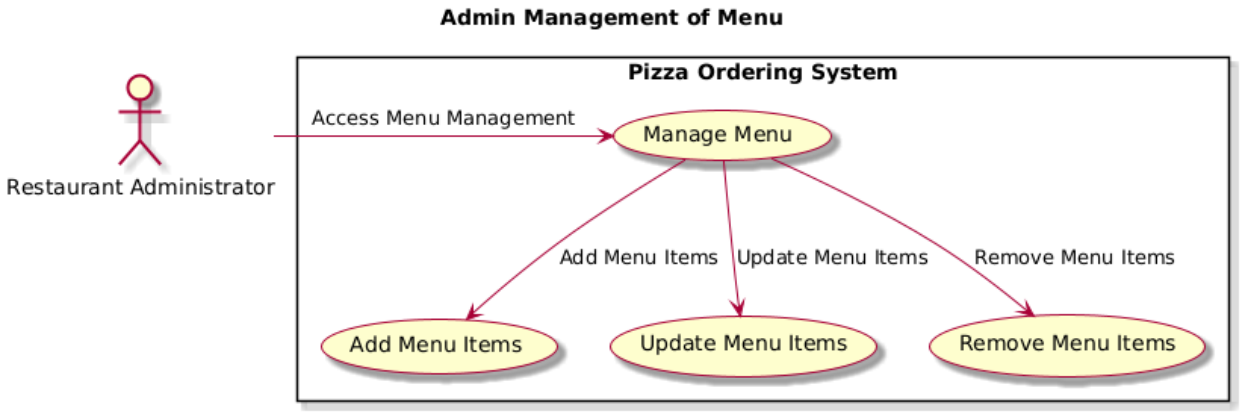
3.1.2.5. Admin Management of Menu and Orders

User Story:

As a restaurant administrator, I want to manage the menu by adding, updating, or removing pizzas, toppings, and drinks, so that I can ensure the menu is accurate and up-to-date.

Actor	Restaurant Administrator
Precondition	The administrator has logged into the system.
Steps	1. The administrator navigates to the menu management section. 2. The administrator adds new menu items (e.g., pizzas, toppings, drinks). 3. The administrator updates existing items (e.g., prices, availability). 4. The administrator removes outdated or unavailable items. 5. The system saves the changes and updates the customer-facing menu.
Postcondition	The menu is successfully updated, and customers can see the latest offerings.

Diagram

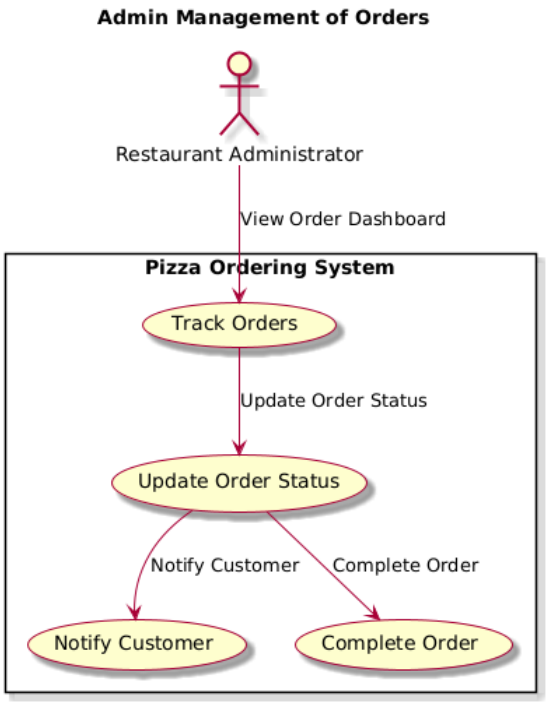


User Story:

As a restaurant administrator, I want to track real-time orders and manage their progress, so that I can efficiently process and prepare customer orders.

Actor	Restaurant Administrator
Precondition	The administrator is logged into the system and viewing the order dashboard.
Steps	1. The system displays real-time incoming orders. 2. The administrator updates the order status (e.g., in preparation, ready for pickup). 3. The system notifies the customer of order progress. 4. The administrator completes the order and marks it as closed.
Postcondition	The order is processed and marked as complete.

Diagram



3.2. Non-Functional Requirements

In this section, I will outline the key non-functional requirements for my pizza restaurant system. These requirements encompass performance, reliability, security, and usability aspects, which are crucial for ensuring a robust and efficient system.

3. 2.1. Performance Requirements

1. **Response Time:** In normal circumstances, the system should react to user activities in two seconds or less; at peak load, this time should be five seconds.
2. **Order Processing:** It should take 5 seconds or less to process an order, including payments and customizations.
3. **Concurrency:** To provide a flawless experience during peak hours, the system should support a high number of concurrent users (the exact number will be determined after analyzing the usage statistics).

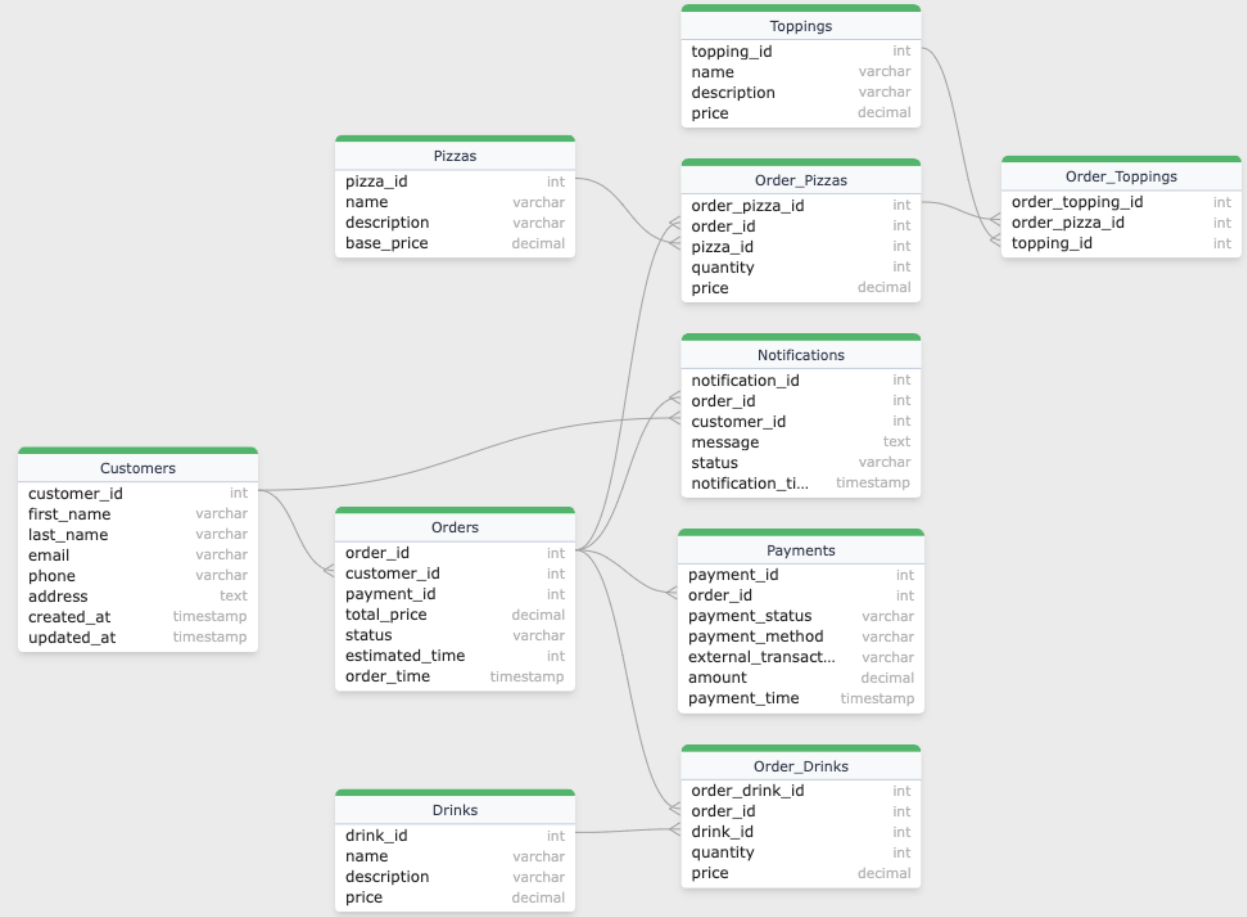
3.2.2. Security Requirements

1. Data Encryption: Use SSL/TLS for secure communication to encrypt sensitive customer data while it's in transit and at rest.
2. Use role-based access control, or RBAC, to restrict access according to user roles, such as admin and customer.
3. Secure every input field to guard against vulnerabilities like SQL injection and cross-site scripting (XSS).

3.2.3. Usability

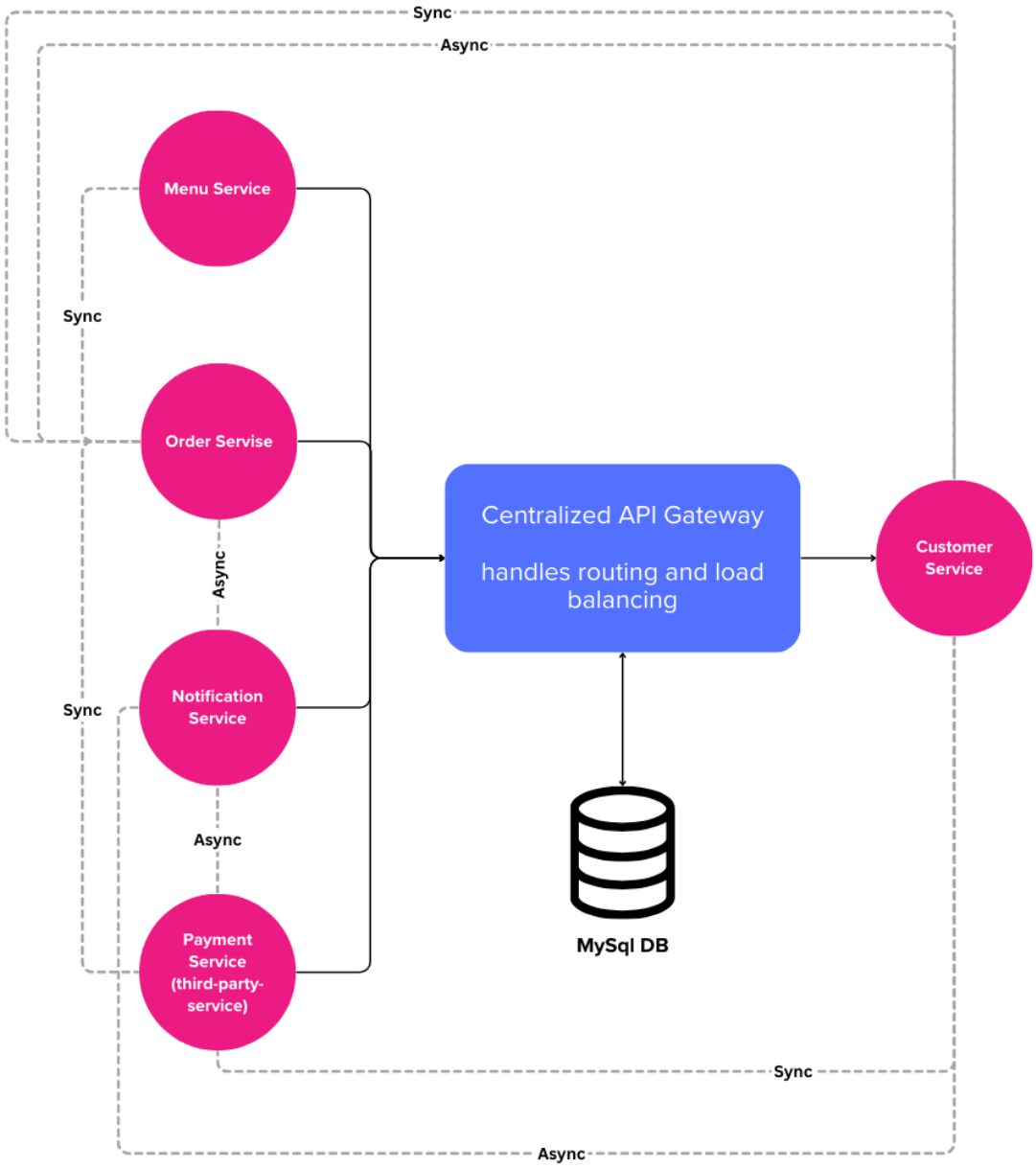
1. **Intuitive UI**: The system will have an easy to use and to understand UI which will make it easier for new user to use the app.
2. **Admin Dashboard:** The system will provide a simple, rich and accurate admin panel, which will be essential for the restaurant staff to use in order to manage the orders, menus and get a general understanding of the system performance.
3. **Device Compatibility:** the system should be working on different devices, to ensure accessibility for the maximum number of users.

3.3. Database Schema



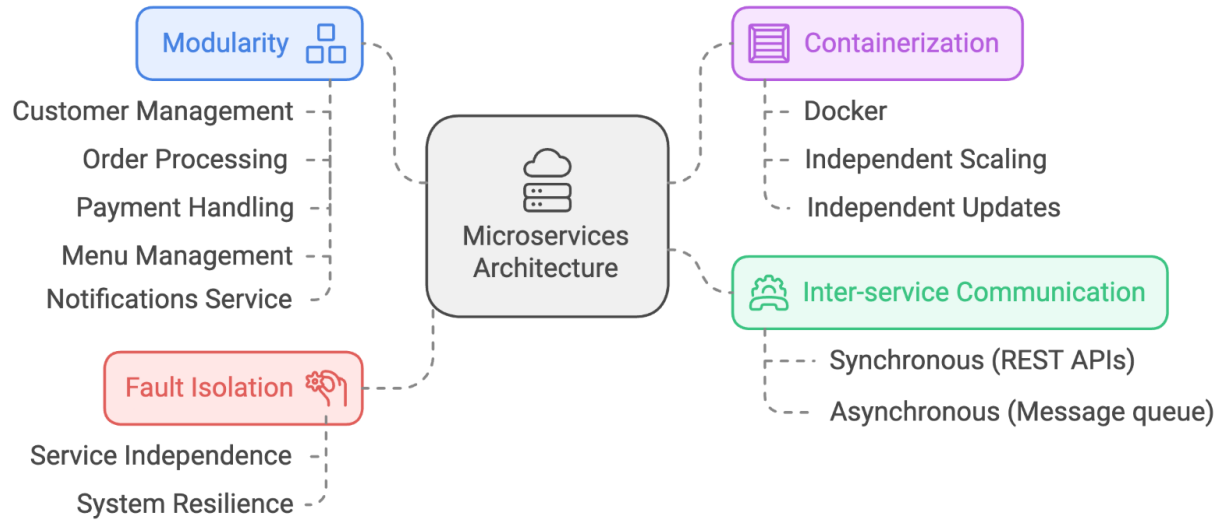
3.4 System Architecture Diagram

The system have five different services that communicate with each other's using synchronous and asynchronous methods. the following diagram will illustrate the system architecture.


Sync
Async
Menu Service
Sync
Order Servise
Async
Notification Service
Sync
Async
Payment Service (third-party-service)
Centralized API Gateway
handles routing and load balancing
Customer Service
MySql DB
Sync
Async

4. System Architecture

In this section, I will outlines the system architecture, describing the microservices design and the various components involved.



4.1. Microservices Architecture Overview

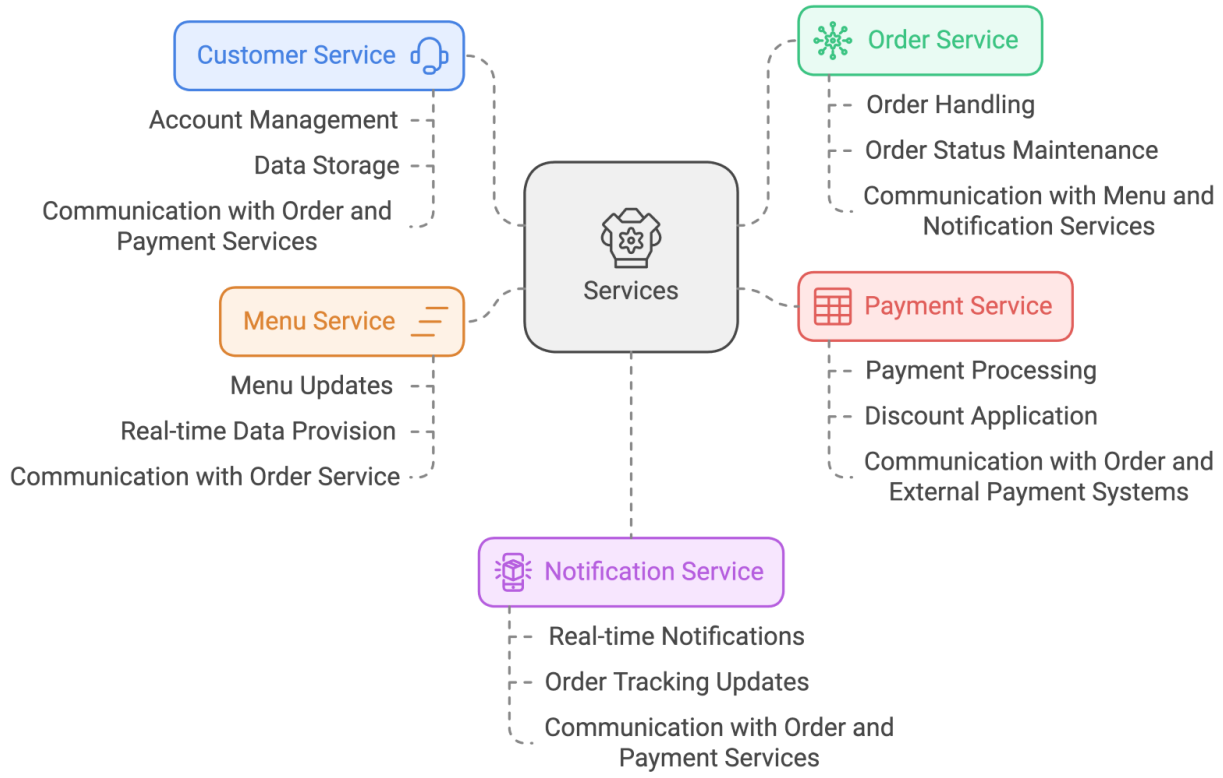
The system design consists of a microservices architecture where each service properly encapsulates a business function and acts as its own, an independently working unit with respect to scaling & monitoring. Services communicate synchronously through REST APIs, and/or asynchronously using message queues providing flexibility and scalability. The main objectives of such a design are:

- **Modularity:** Each microservice does one well-defined functionality of the system (e.g., menu management).
- **Inter-Service Communication:** Services communicate with each other via sync (REST API calls) and async mechanisms like messaging queues based on the requirements of the operation.
- **Fault Isolation:** Since every service runs independently of each other, issues with one specific service will not break down the entire system.

All system components will be containerized via Docker, which allows each service to run independently in their containers scale or be updated at demands.

4.2. Service Components

The following diagram should illustrate the overall services:



5.2.1. Customer Service

Responsibility	This service handles customer account management, including user registration, login, and profile updates. It stores customer details like name, contact information, and order history.
Communication	It communicates with the order and payment services to retrieve and manage customer-specific data related to orders.

5.2.2. Order Service

Responsibility	The core service for handling customer orders, including pizza selection, toppings, drink options, and special requests. It processes order submissions and maintains order status.
Communication	It communicates synchronously with the Menu Service to retrieve available toppings and drinks and asynchronously with the Notification Service to provide order status updates.

5.2.3. Payment Service

Responsibility	This service manages all payment-related activities, including calculating the total bill, applying discounts, and processing payments through external payment gateways.
Communication	It communicates synchronously with the Order Service to confirm the total cost and asynchronously with external payment systems to complete transactions.

5.2.4. Menu Service (Toppings and Drinks Management)

Responsibility	This service manages the restaurant's menu, allowing admins to update available pizzas, toppings, drinks, and prices. It provides real-time menu data to the Order Service.
Communication	It interacts synchronously with the Order Service to ensure up-to-date menu information is available during the ordering process.

5.2.5. Notification Service (Order Status and Time Estimation)

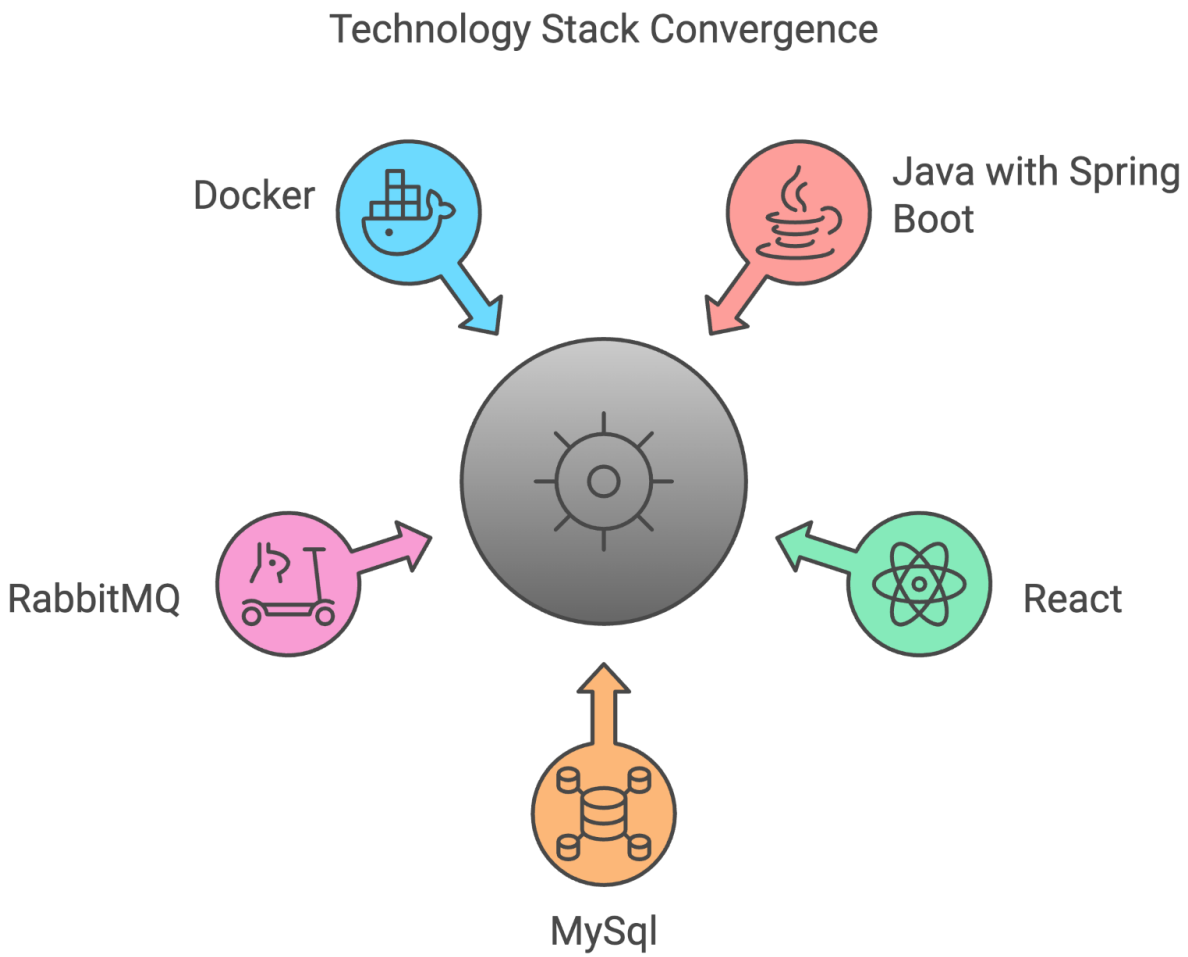
Responsibility	This service sends real-time notifications to customers regarding order status and estimated preparation time. It also provides updates for order tracking and delivery status.
Communication	It receives asynchronous messages from the Order and Payment Services to trigger notifications for status changes (e.g., "Order Received", "Order Prepared").

4.3. Communication Between Services (Sync and Async)

Inter-service communication in this system is managed using both synchronous and asynchronous methods:

- **Synchronous Communication:** There is direct, synchronous communication between Services (Customer, Order, and Menu) using RESTful API's.
- **Asynchronous Communication:** Notification, and Payment services run in asynchronous communication using a message queue. An example is how the Order Service, asynchronously informs the Notification Service about the estimated preparation time to update customers when their order has been placed.

5. Technology Stack



Java with Spring Boot will be used for back-end development, and to guarantee flexible microservices and a responsive user interface I will utilize Reactjs. I will use MySQL as our relational database to store the data. RabbitMQ allows asynchronous messaging between microservices hence efficient in handling order updates and notifications. All services will be containerized using Docker, to ensure consistency and portability across environments.