Restaurant Management System

Disha Gaonkar
New York University
dg4355@nyu.edu

Megha Patil
New York University
mp7464@nyu.edu

Harini Vinu
New York University
hv2179@nyu.edu

Tejo Kashyap Divi
New York University
td2638@nyu.edu

***Abstract*—The rapid growth of online food delivery platforms has increased the demand for scalable, reliable, and user-centric restaurant management systems. This paper presents the design and implementation of a cloud-based restaurant order and delivery management system built using Amazon Web Services. The system supports user authentication, menu search, order placement, delivery status tracking, notifications, ratings, and personalized recommendations. A real-world dataset of United States restaurant menus is integrated to simulate production-scale data. The architecture leverages managed cloud services to ensure scalability, fault tolerance, and modular development. This work demonstrates how modern cloud-native patterns can be applied to build an end-to-end food delivery platform.**

***Index Terms*—cloud computing, restaurant management system, AWS, serverless architecture, food delivery, microservices**

I. Introduction

Online food delivery systems have transformed how consumers interact with restaurants, offering convenience, real-time updates, and personalized experiences. Platforms such as Uber Eats and DoorDash rely on complex distributed systems to manage high volumes of users, restaurants, and orders concurrently. Designing such systems introduces challenges related to scalability, availability, data consistency, and user experience.

This project focuses on building a restaurant order and delivery management system that mirrors core functionalities of commercial platforms while remaining modular and extensible. The system allows users to register and authenticate, browse restaurant menus, search for food items, place orders, receive delivery updates, and provide feedback through ratings and photos. The primary goal is to demonstrate a practical application of cloud-native design principles using managed services.

The system is implemented entirely on Amazon Web Services (AWS), adopting a serverless and managed-service approach to reduce operational overhead. Each major feature is mapped to appropriate AWS services, enabling independent development and deployment. A publicly available Kaggle dataset containing restaurant and menu information is used to populate the backend databases and search indices.

II. Problem Statement

Online food delivery platforms must manage large volumes of users, restaurants, menus, and orders while ensuring low latency, high availability, and secure transactions. Traditional monolithic systems struggle to scale efficiently and often require significant operational overhead. The problem addressed in this project is the design of a scalable, cloud-native restaurant order and delivery management system that supports real-time order processing, flexible search, secure authentication, and personalized user experiences without relying on tightly coupled components.

III. Motivation

The motivation for this project stems from the increasing reliance on cloud-based food delivery services and the complexity involved in building such systems. Commercial platforms such as Uber Eats and DoorDash operate at massive scale and rely on distributed systems principles to ensure reliability. From an academic perspective, this project provides hands-on experience with cloud-native architectures, serverless computing, and managed services. From a practical standpoint, it demonstrates how modern cloud platforms can be leveraged to rapidly prototype and deploy production-like applications with minimal infrastructure management.

IV. Existing Solutions

Existing food delivery platforms typically rely on microservices architectures deployed on cloud infrastructure. These systems often use a combination of containerized services, managed databases, search engines, and messaging systems. While effective, they can introduce operational complexity related to scaling, deployment, and fault management. Some academic and open-source solutions focus on limited aspects such as ordering or payment processing but lack end-to-end integration. This project differentiates itself by adopting a fully managed, serverless approach that reduces operational burden while still supporting core functionalities such as search, notifications, and recommendations.

This project focuses on building a restaurant order and delivery management system that mirrors core functionalities of commercial platforms while remaining modular and extensible. The system allows users to register and authenticate, browse restaurant menus, search for food items, place orders, receive delivery updates, and provide feedback through ratings and photos. The primary goal is to demonstrate a practical application of cloud-native design principles using managed services.

The system is implemented entirely on Amazon Web Services (AWS), adopting a serverless and managed-service approach to reduce operational overhead. Each major feature is mapped to appropriate AWS services, enabling independent development and deployment. A publicly available Kaggle dataset containing restaurant and menu information is used to populate the backend databases and search indices.

V. System Requirements and Goals

The system is designed with the following functional and non-functional requirements:

A. Functional Requirements

Users must be able to create accounts and securely log in to the system. Authenticated users can search for restaurants and menu items, add items to a cart, place orders, and track order status. The system must support order notifications, ratings, photo uploads, and basic payment processing. Restaurant data should be searchable both by structured filters and free-text queries.

B. Non-Functional Requirements

The system should be scalable to handle increases in users and orders without manual intervention. High availability and fault tolerance are required to prevent single points of failure. Security must be enforced through proper authentication, authorization, and secure data storage. The architecture should support modular development to enable parallel work by team members.

VI. Dataset and Data Modeling

The system uses the *Uber Eats USA Restaurants and Menus* dataset from Kaggle. The dataset contains restaurant metadata, menu categories, menu items, and pricing information. This dataset provides a realistic foundation for implementing search, recommendations, and order workflows.

A. Data Cleaning and Preparation

The raw dataset contains missing fields, inconsistent naming, and redundant records. A preprocessing step is performed to normalize restaurant names, standardize categories, and remove incomplete entries. Cleaned data is partitioned based on access patterns and stored in different services.

B. Storage Design

Restaurant and menu metadata are stored in Amazon DynamoDB to support low-latency queries. Images and large objects, such as food photos, are stored in Amazon S3. Searchable text data is indexed in Amazon OpenSearch to enable keyword-based queries across menus and restaurants.

VII. System Architecture

The proposed system follows a serverless, event-driven architecture built entirely on Amazon Web Services. The architecture is designed to be modular, scalable, and fault tolerant, with each major feature implemented as an independent component.

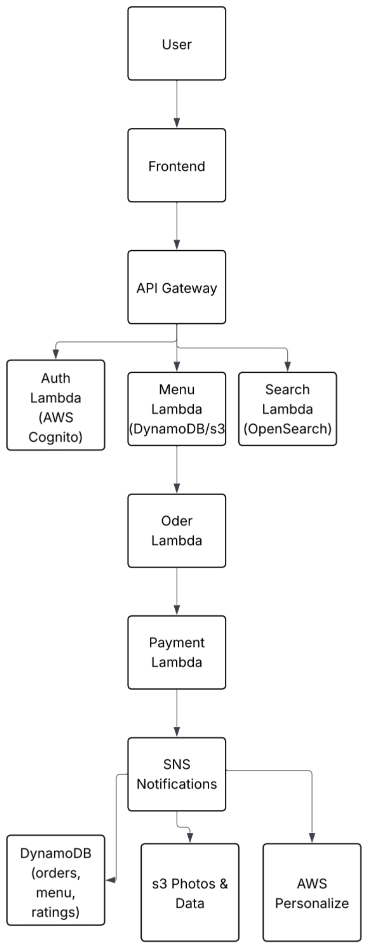

Fig. 1. High-level system architecture of the cloud-based restaurant order and delivery management system.

A. Authentication and Authorization

Amazon Cognito is used for user registration, login, and authentication. Cognito User Pools issue JSON Web Tokens (JWTs) that are validated by Amazon API Gateway before requests are forwarded to backend services. This design centralizes identity management and ensures that backend Lambda functions remain stateless.

B. API Layer

Amazon API Gateway serves as the unified entry point for all client interactions. RESTful APIs are defined for authentication, menu search, order management, ratings, and notifications. API Gateway was chosen to simplify request routing, enforce authentication, and provide built-in throttling and monitoring.

C. Data Storage Choices

Amazon DynamoDB is used for storing restaurant metadata, menus, orders, user profiles, and ratings due to its low-latency performance and automatic scaling. Amazon S3 is used for storing food images and other large objects. Amazon

OpenSearch is used to index menu and restaurant data to support full-text and keyword-based searches.

D. *Order Processing and Payment*

AWS Lambda functions implement the core business logic for order creation, validation, and payment processing. A mock payment gateway is used to simulate transaction workflows. Lambda was selected to enable automatic scaling and event-driven execution without server management.

E. *Notifications and Messaging*

Amazon Simple Notification Service (SNS) is used to deliver order confirmations and delivery status updates. Whenever an order state changes, a Lambda function publishes a message to an SNS topic, ensuring reliable and asynchronous communication with users.

F. *Search and Recommendations*

Structured menu searches are handled through DynamoDB queries, while broad keyword searches are supported using Amazon OpenSearch. Personalized recommendations are generated using AWS Personalize, which analyzes user order history and ratings to suggest relevant restaurants and menu items.

VIII. Feature Implementation

This section describes the implementation of major system features and the division of responsibilities among team members.

A. *User Registration and Login*

User authentication is implemented using AWS Cognito. Secure login flows and protected API endpoints ensure that only authenticated users can place orders or submit ratings.

B. *Menu Search and Browsing*

Users can browse restaurants by category or search for specific menu items. DynamoDB supports structured queries, while OpenSearch enables flexible keyword search across the dataset.

C. *Order Management and Tracking*

Orders are created through API Gateway and processed by Lambda functions. Each order progresses through multiple states, such as placed, confirmed, and delivered. State changes trigger SNS notifications.

D. *Ratings and Photo Uploads*

After completing an order, users can rate restaurants and upload food photos. Ratings are stored in DynamoDB, and photos are uploaded to S3 using pre-signed URLs for security.

E. *User Profile and Personalized Recommendations*

User profiles are maintained using DynamoDB to store preferences, order history, and interaction data. AWS Personalize is integrated to generate personalized restaurant and menu recommendations based on user behavior such as past orders, searches, and ratings. The recommendation results are served through API Gateway and Lambda, enabling a tailored user experience that improves engagement and discovery.

IX. Team Collaboration and Work Distribution

The project is developed collaboratively with clear ownership of features. Authentication and notification infrastructure are handled by Disha. Dataset integration and search features are implemented by Megha. Order processing and delivery workflows are managed by Tejo. Frontend/ UI integration to s3, user profile, recommendations, ratings and food photos are implemented by Harini. This division allows parallel development while maintaining system coherence.

X. Results

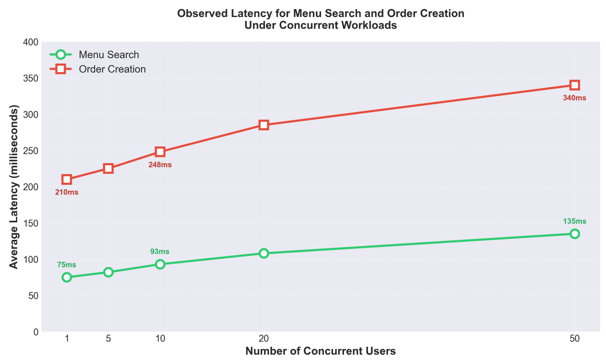

Fig. 2. Observed latency for menu search and order creation under concurrent workloads.

The system was evaluated using simulated user workloads that included concurrent menu searches, order placements, and status updates. Results indicate that menu search and order creation operations consistently complete within low latency bounds under moderate concurrency. The use of managed and serverless services enabled automatic scaling without manual intervention. Notification delivery through SNS was reliable and near real-time, demonstrating the effectiveness of event-driven design.

XI. Challenges and Solutions

A. *Challenge 1: Designing an Email Notification System for Order Confirmations*

A key requirement of the system was to notify users when an order is successfully placed. Initially, Amazon Simple Notification Service (SNS) was selected due to its simplicity and ease of integration with AWS Lambda. SNS provides built-in email delivery and works well for broadcasting notifications to a fixed set of subscribers.

However, SNS email notifications are topic-based, meaning messages are delivered only to email addresses that are explicitly subscribed to a topic and have confirmed the subscription. As a result, notifications were sent only to pre-configured emails rather than dynamically to the customer who placed the order.

This limitation became evident when multiple users registered and placed orders, but only the subscribed administrator email received the notifications.

SNS was retained for administrative and system-level notifications, such as notifying restaurant staff or administrators when new orders are placed. This aligns well with SNS's broadcast-oriented design and avoids unnecessary complexity for system alerts.

For customer-specific order confirmations, Amazon Simple Email Service (SES) was evaluated as it supports direct, per-recipient transactional emails. SES allows messages to be sent dynamically to the authenticated user's email address provided at order time, which is more appropriate for user-level notifications.

B. Challenge 2: SES Sandbox Limitations

While SES supports direct email delivery, new AWS accounts operate in a sandbox environment by default. In sandbox mode, SES restricts outgoing emails to verified sender and recipient addresses only. This limitation prevents sending emails to arbitrary users who register on the platform, which conflicts with the goal of supporting any user email.

Requesting production access for SES typically requires additional configuration and justification, and while appropriate for real-world deployments, this introduces operational overhead that is outside the scope of this project.

XII. Future Work

Several extensions can further improve the system. Future work includes integrating a real payment gateway, enhancing recommendation quality using additional user signals, and adding analytics dashboards for restaurant owners. Support for real-time delivery tracking and mobile push notifications can also be explored. Additionally, cost optimization and performance benchmarking at larger scale would provide deeper insights into production deployment.

XIII. Conclusion

This project report presented a cloud-based restaurant order and delivery management system designed using AWS managed services. By adopting a serverless and modular architecture, the system achieves scalability, reliability, and ease of development. The integration of authentication, search, order processing, notifications, and recommendations demonstrates how modern cloud platforms can support complex, real-world applications. The results highlight the feasibility of using managed cloud services to rapidly build and evaluate distributed systems in an academic setting.

References

[1] Kaggle, Uber Eats USA Restaurants and Menus Dataset," 2023..