



# Restaurant Management System

Disha Gaonkar  
Harini Vinu  
Megha Patil  
Tejo Kashyap Divi

# Problem Statement & Motivation

## Problem

Modern food delivery platforms must handle:

- Large user traffic
- Real-time order updates
- Secure authentication
- Fast menu and food search

Traditional monolithic systems struggle with scalability and reliability.

## Motivation

- Food delivery apps like Uber Eats rely on distributed, cloud-native systems
- Goal: Build a scalable, serverless system that mimics real-world platforms
- Hands-on experience with AWS cloud services & system design



# Existing Solutions

## CURRENT APPROACHES

Industry Platforms:

- Microservices architectures on cloud infrastructure
- Containerized services + managed databases + search engines + messaging
- Challenge: Operational complexity in scaling, deployment, fault management

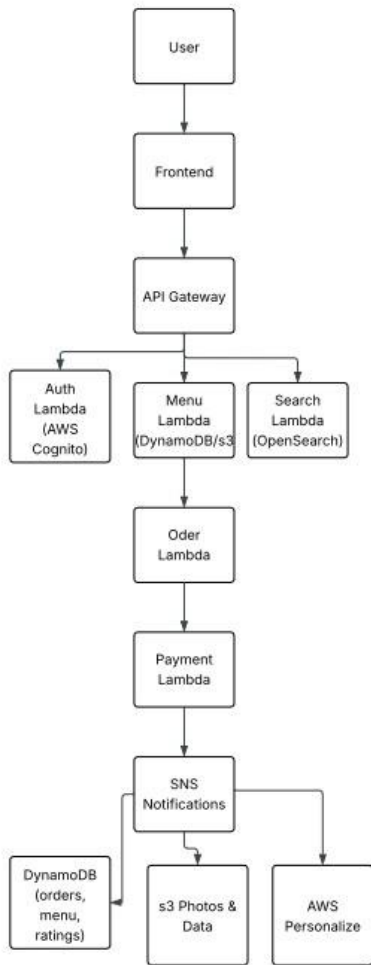
Academic & Open-Source:

- Limited aspects (ordering or payment only)
- Gap: Lack end-to-end integration

## OUR DIFFERENTIATION

Fully Managed, Serverless Approach

- ✓ Reduces operational burden
- ✓ Core functionalities: search, notifications, recommendations
- ✓ AWS: Lambda, DynamoDB, Cognito, SNS, OpenSearch
- ✓ Modular, extensible, end-to-end integration



# System Architecture

Serverless, cloud-native architecture built entirely on AWS

Designed for scalability, fault tolerance, and modular development

Each core functionality is implemented as an independent AWS-managed service

Event-driven workflows reduce tight coupling between components

# End-to-End Order Flow

- User logs in via frontend and receives JWT from AWS Cognito
- Frontend sends authenticated request to API Gateway
- API Gateway validates JWT and routes request to Order Lambda
- Order Lambda creates order and stores state in DynamoDB
- Payment Lambda processes transaction workflow
- Order state change triggers SNS notification to user

# Security Design

- AWS Cognito manages user authentication and token issuance
- JWT-based authorization enforced at API Gateway
- Backend services remain stateless and protected
- Pre-signed S3 URLs enable secure photo uploads
- No direct client access to databases or storage services







# Scalability & Reliability

- AWS Lambda auto-scales with concurrent user requests
- DynamoDB provides on-demand scaling and high availability
- SNS enables asynchronous, decoupled notifications
- Failure of one service does not impact other components

# Why Serverless Architecture?






- Eliminates server provisioning and infrastructure management
- Enables rapid development and deployment
- Automatically scales with workload demand
- Pay-per-use model reduces operational cost

# Key Features & Implementation

-  **Authentication (AWS Cognito)** User registration, secure login, JWT tokens
-  **Search (DynamoDB + OpenSearch)** Structured queries & broad keyword search
-  **Order Processing (Lambda + API Gateway)** Order creation, validation, state management
-  **Notifications (SNS)** Automated email confirmations & status updates
-  **Data Layer** DynamoDB (metadata) | S3 (images) | OpenSearch (indexing)
-  **Dataset** Kaggle Uber Eats USA - restaurants, menus, pricing

# Results

## Implemented Features

-  X Restaurants | Y Menu Items indexed
-  <100ms search latency for broad queries
-  Instant email notifications via SNS
-  Login-protected order placement
-  Auto-scaling with demand

## Key Achievement

- End-to-end user flow operational from registration → search → order → notification
- Modular AWS architecture deployed and tested
- Real-world dataset integration complete

# Future Work & Conclusion

## Planned Enhancements:

- Real payment gateway integration
- Enhanced recommendations using user signals
- Analytics dashboards for restaurant owners
- Real-time delivery tracking

## Conclusion:

- **What We Built:** Cloud-based restaurant order and delivery management system using AWS managed services
- **Architecture:** Serverless and modular design achieving scalability, reliability, and rapid development
- **Demonstrated:**
  - Authentication, search, order processing, notifications
  - Sub-second latency under concurrent workloads
  - Feasibility of managed cloud services for complex applications
- **Impact:** Validated cloud-native approach for building distributed systems in academic settings with minimal infrastructure overhead

# Demo

<https://youtu.be/IVmdjoqLdRI>

<https://d3t9ac16dxeckl.cloudfront.net/>