Online Food Restaurant System Architecture

Technical Architecture Documentation Version 1.0

Table of Contents

1. System Overview

1.1 Purpose

This document outlines the technical architecture for an online food restaurant ordering system, designed to handle multiple restaurants, customers, and delivery partners.

1.2 System Goals

- High availability (99.9% uptime)
- Scalable architecture
- Secure payment processing
- Real-time order tracking
- Efficient order management

2. Architecture Components

2.1 Client Layer

Web Application

- React.js frontend
- Progressive Web App (PWA) capabilities
- Responsive design for all devices

Mobile Applications

- Native iOS (Swift)
- Native Android (Kotlin)
- Shared business logic layer

Restaurant Dashboard

- Admin panel for restaurant management
- Real-time order monitoring

- Inventory management interface

2.2 API Gateway

- Kong/AWS API Gateway
- Rate limiting
- Request routing
- SSL termination
- API documentation (Swagger/OpenAPI)

2.3 Core Services

Authentication Service

- User registration/login
- JWT token management
- Role-based access control
- OAuth2 integration

Order Service

- Order processing
- Status management
- Order history
- Real-time updates

Menu Service

- Menu management
- Pricing
- Availability control
- Category management

Payment Service

- Payment processing
- Refund handling
- Payment gateway integration
- Transaction history

Restaurant Service

- Restaurant profile management
- Working hours
- Location management
- Rating and reviews

Delivery Service

- Delivery partner assignment
- Route optimization
- Real-time tracking
- Delivery status updates

User Service

- Profile management
- Address management
- Preferences
- Order history

2.4 Database Layer

Primary Databases

- PostgreSQL
  - User data
  - Transaction records
  - Order information
- MongoDB
  - Menu items
  - Restaurant profiles
  - Reviews and ratings

Cache Layer

- Redis
  - Session management
  - Frequent queries
  - Real-time data

2.5 Message Queue

- Apache Kafka
  - Order events
  - Notifications
  - Service communication

3. Technical Stack

3.1 Backend Technologies

- Node.js/Express.js for microservices
- Python/FastAPI for data processing
- Go for performance-critical services

3.2 Frontend Technologies

- React.js
- Redux for state management
- Material-UI components
- WebSocket for real-time updates

3.3 DevOps Tools

- Docker
- Kubernetes
- Jenkins/GitHub Actions
- ELK Stack for logging

4. Security Considerations

4.1 Authentication & Authorization

- JWT-based authentication
- Role-based access control
- OAuth2 for social login
- API key management

4.2 Data Security

- End-to-end encryption
- Data masking
- Regular security audits
- PCI DSS compliance

4.3 Infrastructure Security

- WAF implementation
- DDoS protection
- Regular penetration testing
- Security monitoring

5. Deployment Strategy

5.1 Container Orchestration

- Kubernetes clusters
- Auto-scaling policies
- Load balancing
- Health checks

5.2 CI/CD Pipeline

- Automated testing

- Code quality checks
- Deployment automation
- Rollback procedures

5.3 Environment Management

- Development
- Staging
- Production
- Disaster recovery

6. Scalability Considerations

6.1 Horizontal Scaling

- Microservices architecture
- Stateless services
- Database sharding
- Load balancing

6.2 Performance Optimization

- CDN integration
- Cache strategies
- Database indexing
- Query optimization

6.3 Monitoring & Analytics

- Real-time monitoring
- Performance metrics
- User analytics
- Error tracking

7. Appendix

7.1 System Requirements

- Minimum 99.9% uptime
- Maximum 2-second response time
- Support for 100,000+ concurrent users
- Data backup every 6 hours

7.2 Integration Points

- Payment gateways (Stripe/PayPal)
- Maps API (Google Maps)
- SMS/Email services

- Social media APIs

Note: This architecture document serves as a high-level guide and should be adapted based on specific business requirements and constraints.