Software Architecture and Design Specification

Project: Food Delivery Service System

Authors: Jahnavi Srinag, Chirag, Darshan Gowda BM, Isha

Approvals

Role	Name	Signature / Date
Project Guide	Arpitha K	_____
Architect	Jahnavi Srinag	_____
QA Lead	Darshan Gowda BM	_____
Product Owner	Isha	_____
Developer	Chirag	_____

1. Introduction

1.1 Purpose

This document presents the Software Architecture and Design (SAD) for the Food Delivery Service System. It provides a comprehensive overview of the system's structural design, data flow, APIs, and module interactions. The purpose is to ensure alignment between SRS requirements and implementation while addressing scalability, reliability, and security objectives.

1.2 Scope

The Food Delivery Service System enables customers to order food online, restaurants to manage menus and orders, delivery agents to track and deliver orders, and administrators to manage analytics and reports. It integrates with third-party APIs for payments and notifications but excludes inventory and logistics management.

1.3 Audience

This document is intended for developers, QA engineers, system administrators, project managers, and security auditors.

1.4 Definitions

API – Application Programming Interface
MFA – Multi-Factor Authentication
RBAC – Role-Based Access Control
ETA – Estimated Time of Arrival
PCI-DSS – Payment Card Industry Data Security Standard
GDPR – General Data Protection Regulation
JWT – JSON Web Token

2. Document Overview

2.1 How to Use this Document

This document defines the Software Architecture and Design for the Food Delivery Service System. It provides architectural patterns, UML diagrams, component design, and API specifications for development and validation purposes.

2.2 Related Documents

• SRS v1.0 – Food Delivery System Requirements Specification
• STP v1.0 – Food Delivery System Test Plan
• RTM – Requirements Traceability Matrix
• IEEE 42010 – Standard for Architecture Description
• OWASP Application Security Guidelines

3. Architecture

3.1 Goals & Constraints

Goals:

- Secure data handling for customers, restaurants, and payments.
- ≥99.9% uptime and ≤2-second response time.
- Scalability for up to 50,000 concurrent users.
- High modularity and maintainability.

Constraints:

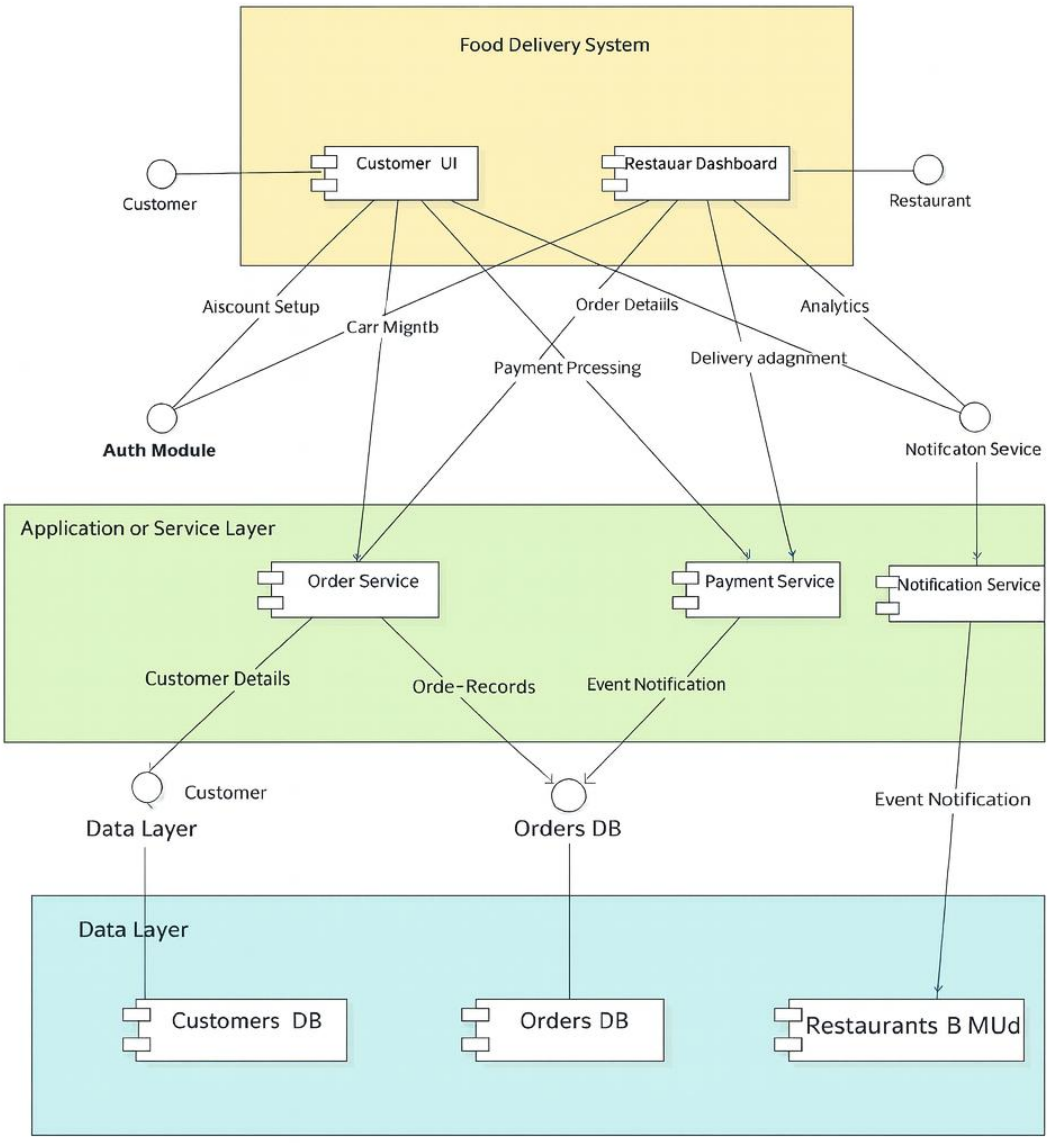
- PCI-DSS and GDPR compliance mandatory.
- Must support Chrome, Firefox, Edge.
- Multi-language support (English + regional languages).
- Use of open-source technology stack and AWS cloud infrastructure.

3.2 Stakeholders & Concerns

Customers – Require a fast, secure, and user-friendly ordering experience.
Restaurants – Need real-time order updates and analytics.
Delivery Agents – Require efficient route tracking and order management.
Administrators – Need dashboards for monitoring and reports.
Developers/QA – Require CI/CD support and modular codebase.

3.3 Component (UML) Diagram

3.3 Component (UML) Diagram



3.4 Component Descriptions

Component	Responsibility
Customer Module	Handles registration, order placement, and order tracking.
Restaurant Module	Manages menus, order approvals, and restaurant analytics.
Delivery Module	Tracks delivery agents, assigns orders, and updates delivery status.
Payment Service	Processes secure payments using PCI-DSS compliant gateways.
Notification Service	Sends SMS/email notifications for order and delivery updates.
Admin Dashboard	Provides reporting, analytics, and system control.
Database Layer	Stores user, order, and delivery data securely.

3.5 Chosen Architecture Pattern and Rationale

Pattern: Layered Architecture (Presentation → Application → Data)
Rationale: This approach ensures clear separation of concerns, easy debugging, and modular expansion. Microservices were considered but ruled out for reduced complexity in the current scope.

3.6 Technology Stack & Data Stores

Frontend: ReactJS / Flutter
Backend: Java Spring Boot
Database: MySQL / MongoDB
Cloud: AWS EC2, RDS, S3
Security: OAuth2, TLS 1.3, JWT
Payment Integration: Razorpay / PayPal APIs

3.7 Risks & Mitigations

Risk	Mitigation
High traffic load	Use AWS ELB, caching, and horizontal scaling.
Data breach	Encrypt sensitive data and enforce MFA.

API downtime	Implement redundancy and fallback mechanisms.
Database failure	Enable regular backups and replication.
DDoS attacks	Deploy AWS WAF and set API rate limits.

3.8 Traceability to Requirements

Requirement ID	Architectural Element
FR-001	Authentication & Authorization Module
FR-004	Cart & Order Service
FR-009	Payment Service
FR-012	Delivery Module
FR-015	Admin Dashboard
NFR-003	Scalability Infrastructure

3.9 Security Architecture

Threat	Mitigation
Spoofing	Implement MFA and signed JWTs.
Tampering	Enforce HTTPS, input validation, and digital signatures.
Repudiation	Enable audit logs with timestamps.
Information Disclosure	Use AES-256 encryption and TLS 1.3.
Denial of Service	Apply rate limiting and scalable infra.
Elevation of Privilege	Apply RBAC and least privilege.

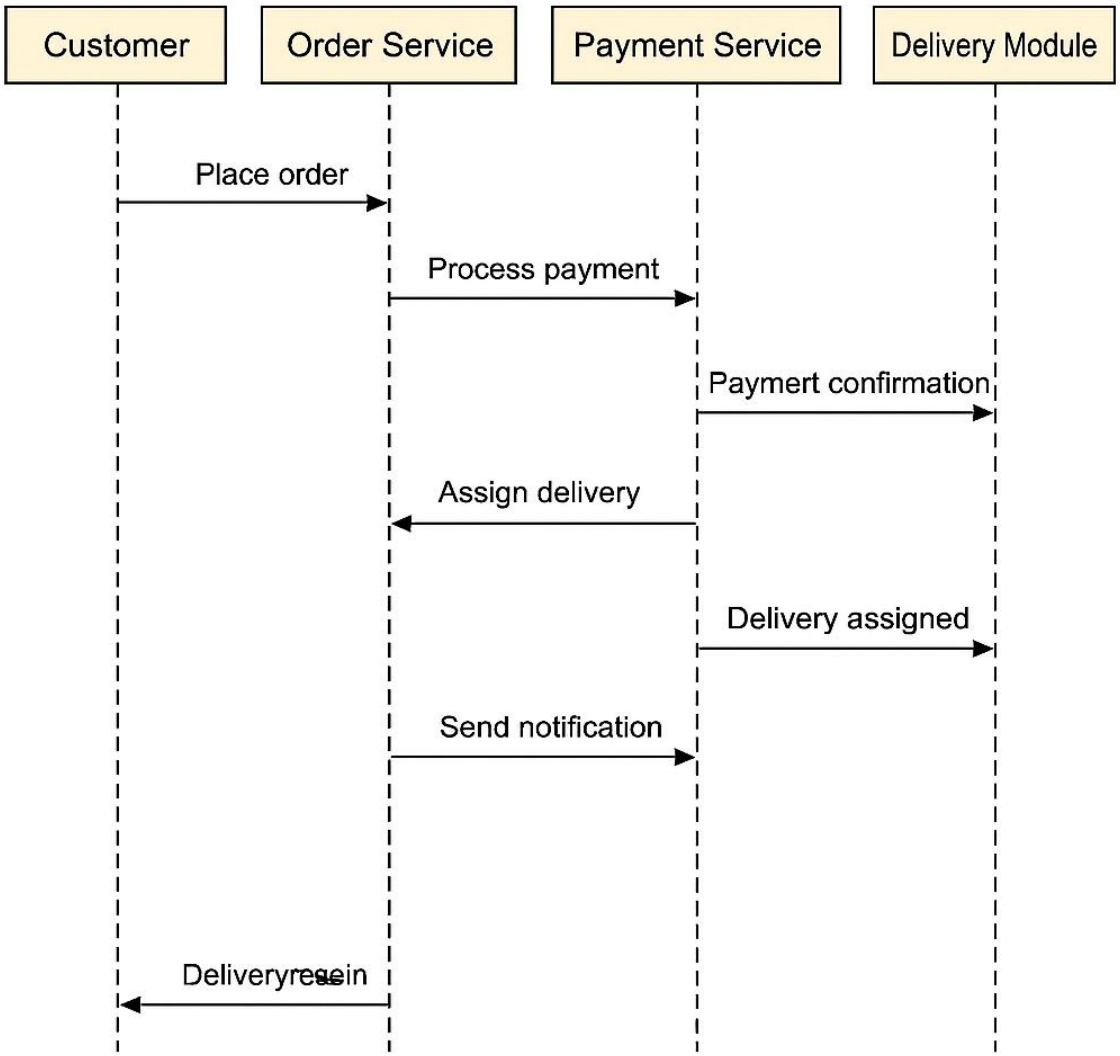
4. Design

4.1 Design Overview

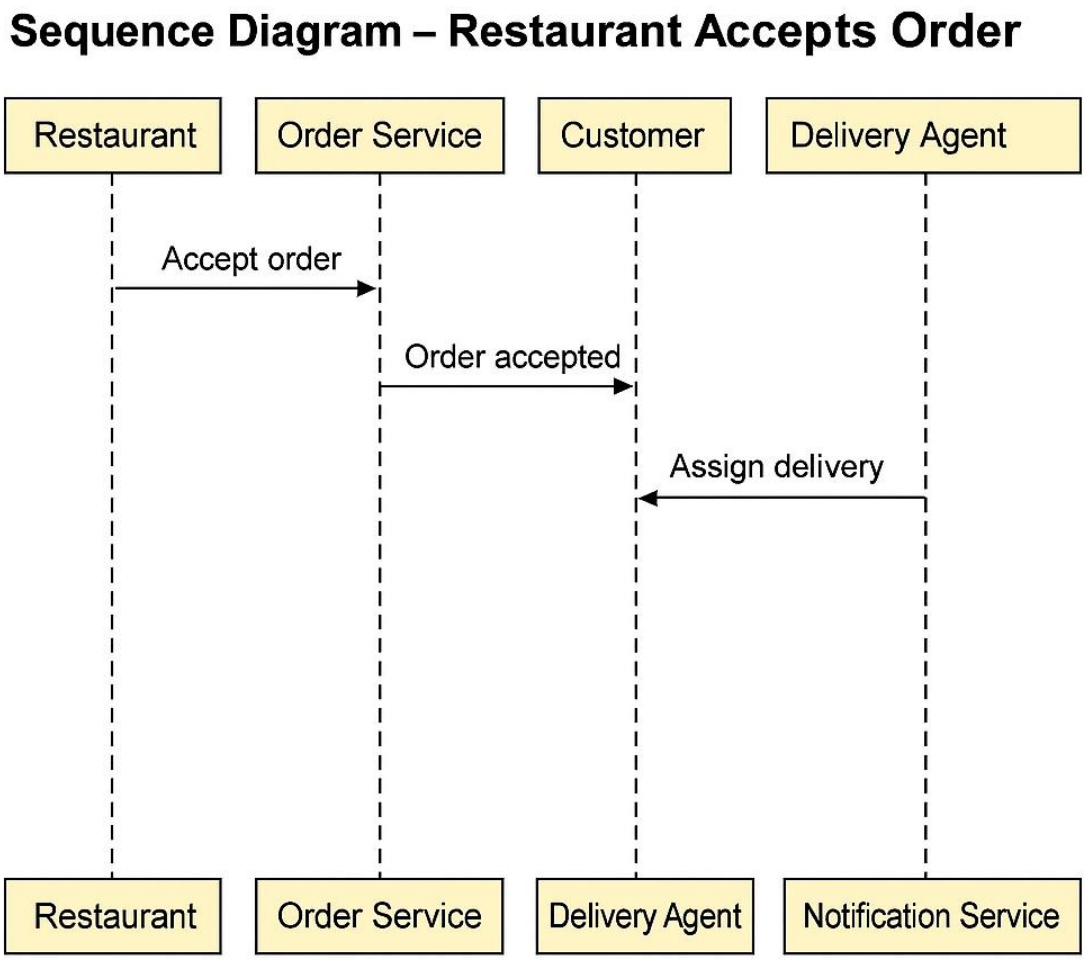
The system follows a modular, layered design. Each service communicates via REST APIs secured with JWT. The architecture promotes scalability, flexibility, and maintainability.

4.2 UML Sequence Diagrams

1. Customer Places Order Flow



2. Restaurant Accepts Order Flow



4.3 API Design

Endpoint: /api/v1/order/place (POST)
Request: {customerId, restaurantId, items[], paymentMethod, address}
Response: {orderId, status, ETA}

Endpoint: /api/v1/delivery/update (PUT)
Request: {deliveryId, status, location}
Response: {message: 'Delivery updated successfully'}

4.4 Error Handling, Logging & Monitoring

Standardized error codes are returned for all APIs. Sensitive information is excluded from logs. Monitoring is performed through Grafana and AWS CloudWatch for real-time health tracking.

4.5 UX Design

The system interface follows a responsive, mobile-first approach supporting both light and dark themes. Accessibility is ensured through WCAG 2.1 AA compliance and intuitive navigation design.

4.6 Open Issues & Next Steps

- Finalize CI/CD pipelines.
- Integrate AI-driven food recommendations.
- Add chatbot-based order placement.
- Conduct security penetration testing.

5. Appendices

5.1 Glossary

API, JWT, RBAC, MFA, PCI-DSS, GDPR, REST, HTTPS

5.2 Acronyms

API, UI, DB, JWT, MFA, RBAC, PCI-DSS, TLS

5.3 References

- Food Delivery System SRS v1.0
- Food Delivery System Test Plan v1.0
- IEEE 42010: Architecture Description
- OWASP ASVS
- NIST SP 800-160