# SYSTEM DESIGN

REPORT

NAME: GUBBALA HEMANTH KUMAR
REGISTRATION NO: 12409776
ROLL NO: 48

# FULL FOOD-DELIVERY AGGREGATOR SYSTEM REPORT

*Orders, Dispatch, Live Tracking, Real-Time Architecture*

## Executive Summary

**The Food-Delivery Aggregator System is a large-scale platform that connects customers, restaurants, and couriers to facilitate seamless ordering, dispatching, and real-time delivery tracking. The platform focuses on 24/7 availability, low-latency location updates, optimized driver assignment, and geographically scalable architecture using microservices**

The Food-Delivery Aggregator System is a comprehensive, large-scale, cloud-based platform designed to connect millions of customers, restaurants, and couriers across a geographically distributed environment. The primary aim of this system is to provide users with a seamless food-ordering experience by enabling menu exploration, secure ordering, transparent payment workflows, and real-time delivery tracking. In addition to serving customers, the system provides restaurants with tools to manage their menus, receive and process orders, and enhance operational efficiency during peak hours. For couriers, the platform provides route assistance, optimized delivery assignments, and earning insights.

This project addresses complex challenges such as low-latency location tracking, data consistency across microservices, intelligent dispatching algorithms for courier assignment, and high scalability during peak load times such as weekends, festivals, and lunch/dinner rush hours. With a strong focus on high availability, disaster resilience, security, and maintainability, this system utilizes microservices architecture, event-driven communication, and distributed caching to achieve exceptional performance.

This report outlines the entire ecosystem of the food delivery aggregator platform, covering all major architectural components, workflows, diagrams, algorithms, data models, API contracts, DevOps considerations, SLOs, scalability techniques, and engineering challenges. The report concludes with future enhancements and recommendations.

---

## Problem Definition

Food delivery platforms face several technical and operational challenges. Customers demand a highly responsive and feature-rich system where they can browse restaurants, place orders quickly, make secure payments, and track the status of deliveries in real-time. Restaurants require efficient tools for managing orders, updating menu items, and predicting preparation times. Couriers need accurate dispatching, route guidance, and timely notifications to complete deliveries efficiently.

The core problems the system must solve include:

### A. Real-Time Order Management

Handling thousands of concurrent orders, ensuring restaurant acceptance, preparation updates, and delivery sequencing with minimal delay.

**B. Intelligent Dispatching**

Selecting the best courier for each delivery requires evaluating dynamic factors such as realtime location, traffic patterns, courier load, historical performance, and distance to both restaurant and customer.

**C. Real-Time Tracking**

Delivering near-instant updates (<2 seconds) of courier GPS location to the customer map interface, while efficiently streaming millions of WebSocket messages.

**D. Scalability & Reliability**

Ensuring uninterrupted service even during peak load, implementing autoscaling, CDS caching, multi-region sharding, and fault tolerance.

**E. System Complexity**

The system spans multiple interacting modules: menus, orders, dispatch, payments, tracking, notifications, restaurants, couriers, and customers. Ensuring consistency and maintainability is a key challenge.

The final solution must address all these issues while ensuring smooth user experience, strong security, and operational efficiency.

---

## Stakeholder Analysis

A large number of internal and external entities interact with a food-delivery system. Understanding stakeholder needs ensures better system design.

**1. Customers**

- Browse menus and restaurants
- Add items to cart and place orders
- Track orders and couriers
- Receive notifications
- Secure payment processing
- Ability to raise issues or refunds

**2. Restaurants**

- Menu management (add/update/delete items)
- Inventory updates in real-time

- Accept, prepare, and complete orders
- Estimate preparation times
- Receive operational insights and daily reports

**3. Couriers**

- Receive order assignments
- Accept or reject dispatch requests
- Get optimized route guidance
- Send live GPS updates
- Track earnings, daily targets, and ratings

**4. Operations Team**

- Monitor system health
- Handle escalations
- Analyze fraud, delayed orders, courier issues
- Manage promotions and system configurations

**5. Business Stakeholders**

- Analyze financial performance
- Monetization strategy implementation
- Region-wise demand analysis
- Strategic decisions based on system dashboards

### Developers & DevOps Engineers

- Maintain uptime
- Manage scaling, monitoring, deployments
- Bug fixes, feature development

Each stakeholder influences critical features of the system.

---

## Functional Requirements

Below is a deeply expanded set of functional requirements.

**Customer App Requirements**

1. User registration and login
2. Location-based restaurant discovery
3. Menu browsing with categories
4. Cart creation and modification
5. Coupon application and discount engine
6. Secure payment options (UPI, card, wallet, COD)
7. Order tracking with step-by-step updates
8. Real-time courier map tracking
9. Customer support / chat bot
10. Ratings and feedback system
11. Past orders and reorder feature

**Restaurant Dashboard Requirements**

1. Restaurant registration, verification
2. Menu creation and editing
3. Manage item availability
4. Order acceptance workflow
5. Track courier arrival
6. Manage preparation times
7. Daily reports & analytics
8. Auto-accept settings

**Courier App Requirements**

1. Registration and verification
2. Online/offline status
3. Order assignment alerts
4. Accept/reject workflow
5. Navigation (Google Maps, OSRM API)

6. Continuous GPS tracking
7. Delivery status updates
8. Earnings summary

**Backend Service Requirements**

1. Order lifecycle management
2. Dispatch algorithm execution
3. Real-time tracking
4. Payment processing
5. Notification management
6. Menu caching
7. ETA prediction
8. Fraud detection

---

## Non-Functional Requirements (NFRs)

A production-grade food delivery system requires strict attention to NFRs.

**Performance**

- API latency: p95 <200ms
- WebSocket updates: <2 seconds
- Payment completion: <3 seconds
- Delivery ETA accuracy: ±3 minutes

**Scalability**

- Horizontal autoscaling for microservices
- Multi-region cloud infrastructure
- Geo-sharded databases

**Security**

- HTTPS everywhere
- PCI-DSS compliant payment flow

- RBAC enforcement
- DDoS protection
- Anti-fraud monitoring

**Reliability**

- 99.9% uptime
- Redundant services
- Retry policies and backoff
- Circuit breaker patterns

**Maintainability**

- Clean and modular microservices
- CI/CD
- Automated test coverage
- Logging, monitoring, alerting

---

## System Architecture Overview

The architecture follows:

**Microservices Architecture**

Each domain (e.g., Orders, Dispatch, Menu) has its own microservice. This ensures independent deployment, scalability, and maintainability.

**Event-Driven Architecture**

Using Kafka/Pulsar ensures asynchronous communication:

Events such as:

- ORDER_PLACED
- ORDER_ACCEPTED
- ORDER_READY
- COURIER_ASSIGNED
- COURIER_PICKED
- ORDER_DELIVERED

are broadcast through event streams.

**API Gateway**

Handles:

- Rate limiting
- Routing
- Token validation
- Request aggregation

**Databases**

- PostgreSQL for relational data
- Redis for caching
- MongoDB or DynamoDB for event logs

**CDN**

Used for static content and cached menu snapshots.

**WebSocket Layer**

High-throughput, low-latency push updates for tracking.

---

## Context Diagram (Expanded)

```
+----------------------+
|      Customer       |
+-----------+-----------+
            |
            | Browse, Order, Track
            v
      +-----+-----+
      |  API GW   |
      +-----+-----+
            |
       ----------------------------------------------------
       |         |          |          |        |
       v         v          v          v        v
   +--------------+ +------------+ +-----------+ +----------+ +--------------+
   | Order Svc    | | Menu Svc   | | Payment   | | Dispatch | | Tracking Svc |
   |              | |            | | Service   | | Service  | | (WebSocket)  |
   +--------------+ +------------+ +-----------+ +----------+ +--------------+
       |                                 |
|                                  |           v
v
  +----------------+                      +-----------------+
  | Restaurant App |                      | Courier App     |
  +----------------+                      +-----------------+
```

---

## Use Case Diagram (Expanded)

Actors:

- Customer
- Restaurant User
- Courier
- Admin

Use Cases:

- UC01: Register/Login
- UC02: Browse Restaurants
- UC03: Place Order
- UC04: Accept Order
- UC05: Assign Courier
- UC06: Track Delivery

- UC07: Update Menu
- UC08: Analyze Performance
- UC09: Payment Processing

---

## Dispatcher Design: Matching Algorithm (Highly Expanded)

### Why Intelligent Matching is Required

Assigning the wrong courier leads to:

- Late delivery
- Bad customer experience
- Traffic delays
- High operational cost

### Inputs to Algorithm

- Courier real-time location
- Courier experience level
- Traffic density (via Maps API)
- Distance to restaurant
- Distance to customer
- Courier workload
- Courier vehicle type (bike, scooter, cycle)
- Expected restaurant preparation time

### Scoring Model (Expanded)

The algorithm computes a weighted score:

Score =

  0.30 * Distance_Restaurant +

  0.20 * Distance_Customer +

  0.15 * Load_Factor +

  0.10 * Courier_Rating +

0.10 * Traffic_Index +

0.10 * Historical_Acceptance +

0.05 * Vehicle_Speed_Category

Lower score → better candidate.

**Dispatch Workflow (Detailed)**

1. Order marked as READY
2. Dispatcher identifies couriers within 3km radius
3. Compute score for each courier
4. Send dispatch request to top-ranked courier
5. If courier rejects → next courier selected
6. If all reject → expand radius, retry
7. Courier accepts → route guidance provided

---

## Data Model (Highly Expanded)

**Orders Table**

- order_id (UUID, PK)
- customer_id (UUID)
- restaurant_id (UUID)
- courier_id (UUID, null allowed)
- cart_items (JSONB)
- total_amount (FLOAT)
- taxes (FLOAT)
- status (enum: placed, accepted, preparing, ready, picked, delivered)
- payment_status (enum)
- created_at
- updated_at

**Restaurants**

- id
- name
- address
- cuisine type
- rating
- operating_hours
- is_open

**Menu Items**

- item_id
- restaurant_id
- name
- description
- price
- availability **Courier Location Table**

Stores latest GPS reading.

**Events Table (for Audit Logging)**

---

## API Contracts (Expanded)

**POST /orders/create**

Handles full order placement with validation, payment link generation, and event publishing.

**GET /orders/{id}**

Returns detailed order view including events, ETA, courier location (if assigned).

**POST /dispatch/assign**

Triggers matching algorithm.

**PUT /courier/status**

Updates courier online/offline status.

**POST /restaurant/menu/update**

Triggers CDN purge + Redis invalidation.

(Dozens more endpoints included in full system.)

---

## Menu Caching & CDN (Deep Explanation)

Menu reads are 80–90% of total API calls.
Thus, a multi-layer caching strategy is used:

**L1 Cache: CDN**

- Caches static menu snapshots
- TTL = 5 minutes
- Best for non-dynamic content

**L2 Cache: Redis**

- Stores items in hash/key-value structure
- Ultra-fast reads
- Invalidation via Pub/Sub events

**L3 Cache: Database**

- Permanent storage

This reduces load on backend, improves speed, and reduces cost.

---

## Rate Limiting & Backoff Strategies (Expanded)

**Rate Limits**

- Customer: 100 requests/min
- Courier: 300 requests/min (mostly GPS updates)
- Restaurant: 200 requests/min

**Backoff Strategies**

- Exponential Backoff for failed dispatch
- Fibonacci backoff for retrying failed notifications
- Circuit breaker patterns to avoid cascaded failures

---

## SLOs & Error Budgets (Expanded)

### SLO Targets

- Availability → 99.9%
- Tracking Latency → <2 sec
- Order API Latency → p95 <200ms
- Payment Failure Rate → <0.3%

### Error Budget

- Monthly downtime allowed: **43.2 minutes**

If 80% is consumed early → freeze deployments.

---

## Scalability: Geo-Sharding, Load Balancing & Hot Regions

### Geo-Sharding

Every region (North, South, East, West India) receives:

- Local databases
- Regional dispatch engines
- Regional WebSocket clusters
- Local cache clusters

This reduces latency drastically.

### Hot Regions

Examples:

- Hyderabad IT Corridor (HITEC City)
- Bangalore Koramangala/Whitefield
- Mumbai Andheri/Bandra

Autoscaling based on:

- Orders per minute
- Courier online count

- Restaurant density

---

**16. Maintainability, Security & Engineering Challenges**

**Maintainability**

- Modular microservices
- Code ownership model
- Proper documentation
- Versioned APIs

**Security**

- OAuth2 + JWT
- TLS for all traffic
- PCI-DSS payment compliance
- Access logs + anomaly detection

**Operational Challenges**

- Scaling WebSocket servers
- Handling payment disputes
- Accurate ETA prediction
- Managing delivery surge during rains/festivals

---

## Conclusion

The Food Delivery Aggregator Platform is a complex, mission-critical system that requires careful engineering across multiple domains such as scalable cloud infrastructure, real-time tracking, accurate dispatch algorithms, secure payments, intuitive user interfaces, and resilient backend services. By leveraging microservices architecture, event-driven communication, caching strategies, and geo-sharded infrastructure, this platform can handle millions of users with high reliability and low latency.

As food delivery demand continues to grow, future improvements may include AI-powered delivery optimization, drone-based delivery, autonomous routing, real-time ML-driven ETA corrections, and enhanced predictive analytics for restaurants. With its strong architectural foundation, the system is well-positioned to evolve with the future of on-demand delivery.