Food-Delivery Aggregator - Requirements Pack

Submitted By:
Name: Manjot Kaur
Roll Number: 08
Registration Number: 12412384
Course: B.Tech (Generative AI)
Semester: 3
Session: 2024–2025
College: Lovely Professional University

Overview

Build a marketplace that ingests menus, takes orders, assigns drivers, and streams status updates. Key goals: availability (24/7), real-time ETA updates, scalability across geographies, and clear domain boundaries.

Audience

Engineering leads / architects (primary), Product managers (secondary).

---

1) Stakeholder analysis & prioritization

Stakeholder	Interests	Priority
Customers	Fast ordering, accurate ETAs, reliable deliveries	High
Restaurants	Accurate menu/catalog, timely orders, commission clarity	High

Couriers	Fair assignments, clear routing, earnings visibility	High
Ops / Support	Monitoring, incident response, capacity control	Medium
Payments	Fraud prevention, settlement speed	Medium
Legal / Compliance	Data residency, privacy, terms	High

Key takeaways: prioritize customer UX, restaurant onboarding flows, courier reliability, and operational observability.

---

2) Requirements

Functional

- Menu ingestion & caching (restaurant, categories, prices, availability)
- Cart & checkout (taxes, tips, promos)
- Order lifecycle: placed → accepted (restaurant) → prepared → assigned (courier) → pickup → enroute → delivered → closed
- Dispatch / matching: assign nearest/available courier using policy (latency, ETA, fairness)
- Live tracking: courier geolocation streaming to customer & restaurant
- Notifications: push, SMS, email for status changes
- Payments: charge customer, hold/settle to restaurant, fees
- Admin: dashboards, overrides, restaurant onboarding, courier management
- Reporting & auditing (orders, payouts, disputes)

Non-functional

- Availability: 99.9% (regionally) during operating hours; 99.95% critical paths

- Real-time performance: p95 location update propagation < 1s; p95 order placement < 300ms
- Scale targets: support >10k concurrent active deliveries; burst handling 5–10×
- Security: PCI/DSS for payments, GDPR for personal data

Technical

- Real-time channels: WebSocket or Server-Sent Events for location/status
- Use third-party Maps (routing/ETA) with caching of route results
- Geo-sharding for regional scaling and data residency
- Event-driven (message bus) for decoupled services

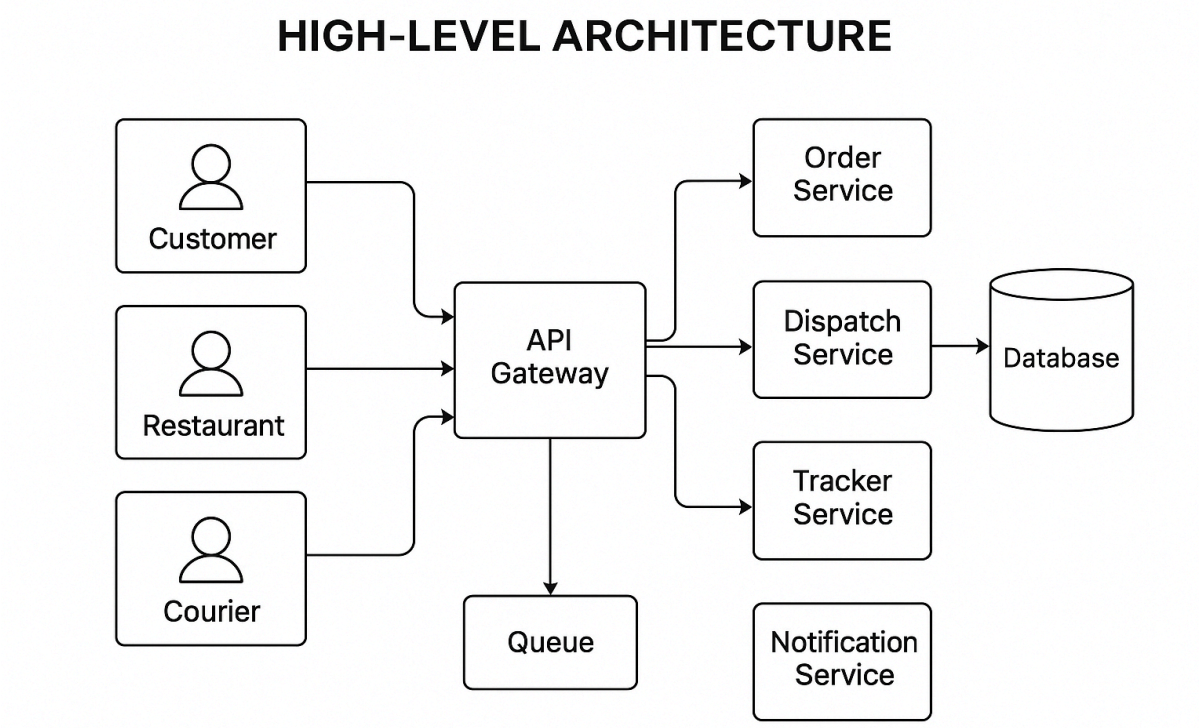
---

3) Constraints & assumptions

- Use third-party maps/routing APIs (e.g., Google Maps, Mapbox)
- Eventual consistency acceptable for ETAs and route recalculations
- Courier devices have intermittent connectivity
- Restaurants own menu truth but system caches for performance
- Payment provider SLA ≥ 99.9%

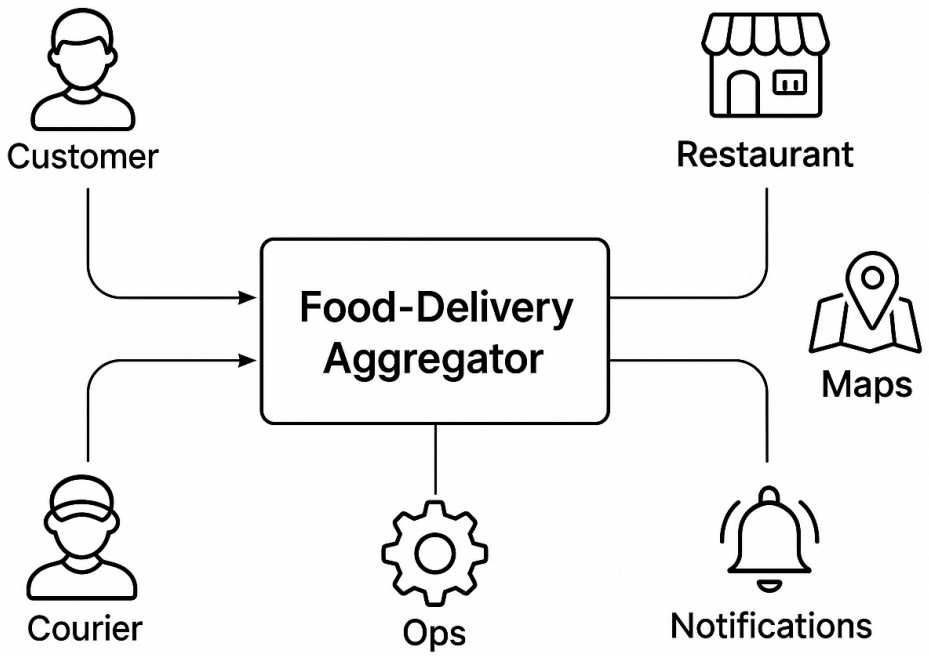
---

4) High-level architecture

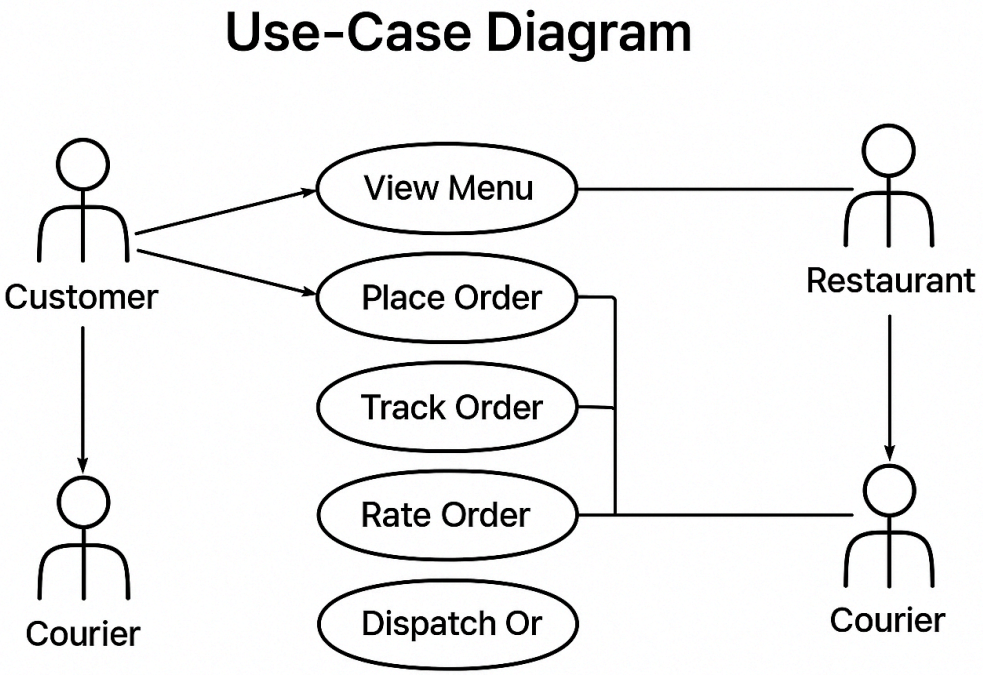


---

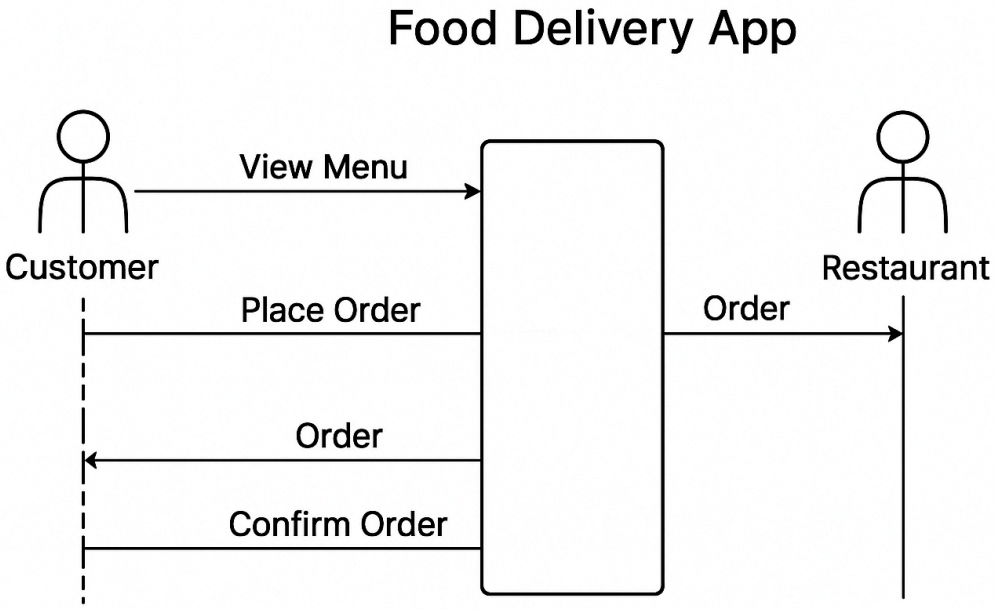
5) System Context



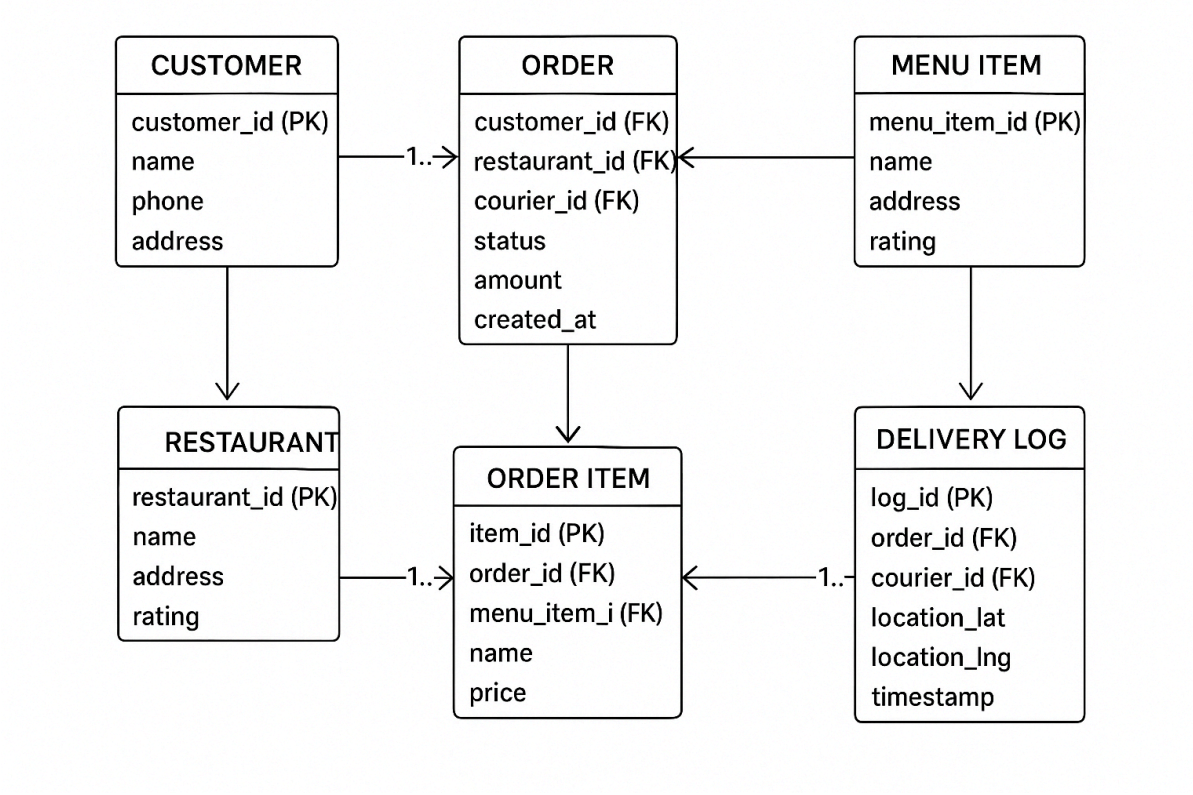
6) Use Case Daigram



7) SEQUENCE DIAGRAM



8) ER / Data Model Daigram



10) Core sequence diagrams

A. Place order

1. Customer sends POST /orders
2. API gateway → Ordering Service validates cart, reserves inventory (if needed)
3. Payment auth (optional) → Payments Service
4. Order persisted (SQL), event OrderCreated published
5. Notification & Restaurant push

B. Dispatch & assignment

1. OrderCreated → Dispatch Service subscribes

2. Dispatch queries available couriers (Location store / redis)
3. Compute candidates + policy → assign courier (write assignment to DB)
4. Publish OrderAssigned event → Real-time gateway notifies courier & customer

C. Live tracking

1. Courier sends periodic location updates via WebSocket
2. Location Service writes to in-memory store (Redis) + publishes LocationUpdate
3. Routing/ETA recalculates ETA asynchronously; updates propagated to customers via socket

---

11) Dispatch design – matching algorithm

- Candidate selection: nearest N couriers within radius & available status
- Scoring factors: ETA-to-pickup, courier load (concurrent orders), fairness score, courier acceptance rate, surge pricing incentives
- Matching modes: greedy (fast), batch (optimize multiple orders), hybrid (time-windowed).
- Failure handling: fallback radius expansion, requeue, manual override.

---

12) Data model

- Order(id, user_id, restaurant_id, status, total, payment_id, created_at)
- OrderItem(order_item_id, order_id, menu_item_id, qty, price)
- Menu(restaurant_id, menu_json, version, cached_at)
- Courier(id, status, location(lat,lon), updated_at, current_load)
- Assignment(order_id, courier_id, assigned_at, accepted_at)

- RouteSnapshot(order_id, courier_id, eta_pickup, eta_delivery, polyline)

Indexes: orders(restaurant_id, created_at), courier(location spatial index), assignments(courier_id, status)

---

13) API

- POST /v1/orders — create order (body: cart, restaurantId, paymentMethod)
- GET /v1/orders/{id} — get order status
- POST /v1/couriers/{id}/location — courier location heartbeat (or ws)
- POST /v1/dispatch/{orderId}/assign — manual assign override (admin)
- POST /v1/websocket/connect — upgrade to WS for live updates

Auth: OAuth2 / JWT tokens scoped by role (customer/restaurant/courier/admin)

---

14) Caching & CDN

- Menus served via CDN with short TTL + versioning on updates
- Redis for hot reads: courier presence, slot caches, candidate caches
- Route result caching (start,end,time) to reduce map API calls

---

15) Consistency choices & tradeoffs

- Strong consistency: order state transitions & payments (ACID, SQL)
- Eventual consistency: ETA calculations, route updates, analytics

- Reason: user-facing financial flows need correctness; ETA/route are best-effort and recomputed

---

16) Resiliency

- Retries with exponential backoff for external APIs
- Circuit breakers for Maps & Payment providers
- Idempotency keys for order placement and payment capture
- Dead-letter queue for failed events

---

17) Observability & Ops

- Structured logs (JSON), distributed traces (OpenTelemetry), metrics (Prometheus)
- Alerts: order creation error spike, dispatch failures, map API latency, location ingestion lag
- Dashboards for active deliveries, average ETA accuracy, courier utilization

---

18) Capacity planning & rough sizing

- Target: 10k concurrent active deliveries; assume 2–5 location updates/sec per courier
- Location update QPS: 10k * 3 = 30k writes/sec to ingest layer
- Use partitioned ingestion horizontally (shard by geo region)
- Kafka topics partitioned by region with 100+ partitions for throughput
- Cache tier sized for hot courier states (Redis cluster with memory to hold ~100k keys)

Back-of-envelopes included in slides.

---

19) SLOs & Quality targets

- Availability: 99.9% (region); critical flows (order placement, assignment) 99.95%
- Latency: p95 order placement < 300ms; p95 location update propagation < 1s
- Error budget: 0.05% monthly for critical flows

---

20) Maintenance & runbook

- Blue-green deploys; schema migrations with data backfills
- Daily integrity checks for orders vs payouts
- On-call rotation & incident playbooks (lost courier, double-charge, payment failures)

---

21) Trade-offs summary

- SQL for strong order correctness vs NoSQL for telemetry — chosen hybrid
- Third-party maps reduce effort but increase external dependency; mitigate with caching & fallback
- Real-time strictness (1s) can be expensive — accept eventual consistency for analytics

---