# CallSphere Assignment: Multi-Agent Logistics & Delivery Platform (OpenAI-Only, Chat + Voice)

## 1) Objective

Design and implement an end-to-end **logistics & last-mile delivery** platform with: - A **Customer dashboard** that supports **both** a **chat agent** and a **browser voice agent**. - An **Admin/Ops dashboard** for real-time monitoring, issue handling, escalation workflows, and **editable metric configuration**.

The system must be production-minded (clean architecture, RBAC, auditability) while remaining feasible as an assignment prototype.

---

## 2) Required Tech Stack

- **Backend:** NestJS + Prisma + PostgreSQL
- **Frontend:** React + TypeScript + TailwindCSS
- **Realtime:** WebSockets (NestJS Gateway + frontend WS client)
- **AI (OpenAI only):**
- **OpenAI Agents** for multi-agent orchestration (router + specialist agents)
- **OpenAI Realtime API** for browser **voice** (STT/TTS) and streaming responses

**Constraints:** - Use **OpenAI infrastructure only** for AI/voice (no other LLM providers, no self-hosted models). - The product must support **both** a chat agent **and** a voice agent (not one or the other).

---

## 3) Users, Roles, and RBAC

Implement role-based access control (RBAC) across REST and WebSockets.

### Roles

- **customer**
- Track own shipments (or public tracking by trackingNumber if you choose).
- Create issues and request delivery changes for own shipments.
- **driver**
- View assigned route(s) and stops.
- Update stop status (completed/failed) for assigned shipments.
- **dispatcher**
- View/update shipments and issues within assigned region.
- Manage route adjustments (within policy).
- Read metrics.
- **manager**
- All dispatcher capabilities plus escalation acknowledgments.
- Read metrics; optionally edit certain thresholds if allowed.
- **admin**
- Full access.

- Can edit **metric definitions, thresholds, dashboard layouts, and escalation ladders**.

### RBAC Requirements

- Enforce RBAC in **NestJS Guards** on all REST endpoints.
- Enforce RBAC for **WebSocket channels** (only authorized users can subscribe to route/issue/escalation feeds).
- Frontend must hide/disable UI controls based on role.

---

## 4) Product Scope

### 4.1 Customer Capabilities (Chat + Voice)

1. **Track Shipment**
2. "Where is my package ABC123?"
3. Returns status, last scan, and ETA.
4. **Report Delivery Issue**
5. Damaged, missing, wrong address, missed delivery, delay, other.
6. Creates a DeliveryIssue and notifies Ops in real time.
7. **Request Delivery Change**
8. Reschedule delivery window.
9. Update delivery instructions.
10. Change address (only when allowed by policy).

### 4.2 Ops/Admin Capabilities

1. **Routes & Stops**
2. View route list and route detail (stops, statuses).
3. Driver-level view (what a driver sees).
4. **Issues & Escalations**
5. Issue queue (filter by severity/status/type/region).
6. Escalation ladder execution and acknowledgment.
7. **Metrics Dashboard + Editing**
8. Overview KPIs (on-time rate, first-attempt success, open issues, SLA-risk count).
9. Admin can **edit metric definitions, targets, and thresholds**.
10. Store metric snapshots for performance.

---

## 5) Architecture

### 5.1 Backend (NestJS)

**Modules (suggested):** - `auth`, `users` - `shipments`, `shipment-scans` - `routes`, `route-stops` - `delivery-issues` - `escalations`, `acknowledgments` - `metrics` (compute + snapshots) - `dashboard-config` - `agent-sessions` (chat/voice sessions) - `settings` (policy controls) - `ws` (WebSocket gateway)

**Backend responsibilities:** - Owns the database and domain logic. - Exposes REST APIs (frontend + OpenAI tool calls). - Emits WebSocket events for live updates. - Performs background jobs (SLA risk scanning, metric snapshots).

### 5.2 AI Layer (OpenAI Agents + Realtime)

All agent logic runs through OpenAI: - Define a **router agent** and multiple **specialist agents**. - Provide tools so agents can safely call backend APIs.

**Chat path:** - User message → OpenAI Agent session → tool calls → response text.

**Voice path:** - Browser audio → OpenAI Realtime (STT) → agent routing/tools → response text + OpenAI TTS playback.

### 5.3 Realtime (WebSockets)

Backend publishes real-time events: - Shipment scan updates - Issue created/updated - Escalation triggered/advanced/acknowledged - Optional: metric snapshot updates

**Suggested channels:** - `shipments:<trackingNumber>` - `routes:<routeCode>` - `issues` - `escalations` - `metrics:overview`

---

## 6) Core Data Model (Domain)

Implement at least these tables (Prisma models) with appropriate indexes.

### 6.1 `User`

- `id`, `name`, `email`, `role`, `isActive`, `createdAt`, `updatedAt`

### 6.2 `Shipment`

- `id`
- `trackingNumber` (unique, indexed)
- `orderId` (optional)
- `customerId` (FK → User)
- `fromAddress`, `toAddress`
- `currentStatus` (enum)
- `serviceLevel` (enum)
- `promisedDeliveryDate`
- `lastScanAt`, `lastScanLocation`
- `isVip` (bool)
- `slaRiskScore` (float 0–1)
- `createdAt`, `updatedAt`

### 6.3 `ShipmentScan`

- `id`, `shipmentId` (indexed), `scanType`, `location`, `timestamp` (indexed), `notes`

### 6.4 `Vehicle`

- `id`, `vehicleCode` (unique), `capacityVolume`, `capacityWeight`, `homeBase`

### 6.5 `Driver`

- `id`, `driverCode` (unique), `userId`, `assignedVehicleId`, `homeBase`

### 6.6 `Route`

- `id`, `routeCode` (unique), `date` (indexed), `driverId`, `vehicleId`, `region`

### 6.7 `RouteStop`

- `id`, `routeId` (indexed), `shipmentId` (indexed), `sequenceNumber`, `plannedEta`, `actualArrival`, `status`

### 6.8 `DeliveryIssue`

- `id`, `shipmentId` (indexed), `reportedByUserId`
- `issueType`, `description`
- `aiSeverityScore` (float 0–1)
- `status`, `resolutionNotes`
- `createdAt`, `updatedAt`

### 6.9 `EscalationContact`

- `id`, `userId`, `position`, `contactType`, `timeoutSeconds`, `isActive`

### 6.10 `EscalationLog`

- `id`, `shipmentId`, `deliveryIssueId` (optional)
- `contactId`, `attemptNumber`
- `eventType`, `payload` (JSON)
- `ackReceived`, `ackMethod`, `acknowledgedAt`, `createdAt`

### 6.11 `Acknowledgment`

- `id`, `shipmentId`, `deliveryIssueId` (optional), `userId`, `method`, `notes`, `createdAt`

---

## 7) AI Agents (OpenAI)

Define the following agents (names are suggestions):

1. **LogisticsRouterAgent**
2. Classifies intent and routes to a specialist agent.
3. **ShipmentTrackingAgent**

4. Resolves tracking number, fetches shipment + latest scan, returns status & ETA.

5. **DeliveryIssueAgent**

6. Collects issue details, classifies issueType, assigns aiSeverityScore, creates issue.

7. **DeliveryChangeAgent**

8. Validates and applies delivery changes based on shipment status/policies.

9. **LogisticsEscalationAgent**

10. Triggers escalation for high severity, VIP, or SLA-risk shipments.
11. Advances ladder until ACK is recorded.

12. **LogisticsAnalyticsAgent**

13. Answers ops questions using metrics endpoints/snapshots.

**Tooling requirement:** - Agents must use backend tools to read/write data (no hallucinated updates).

---

## 8) Dashboards (Both Modalities Required)

### 8.1 Customer Dashboard (Chat + Voice)

Build a customer page that includes: - Tracking input + shipment list. - Shipment detail panel (timeline of scans, current status, ETA, open issues). - **Chat + Voice Agent widget**: - Text input + send. - Mic start/stop. - Live transcription for voice. - Unified conversation history across chat and voice.

Must support: - Track shipment - Create issue - Request change

### 8.2 Admin/Ops Dashboard (Metrics + Editing)

Build an internal dashboard with: - **Overview KPIs** (cards + trends) - **Shipments** (filters, detail) - **Routes** (list + route detail) - **Issues** (queue, severity, SLA risk) - **Escalations** (history + acknowledgment actions) - **Metrics Admin** (admin-only): - Edit metric definitions, targets, thresholds - Toggle visibility on dashboards - Manage dashboard layouts (role defaults)

---

## 9) Database Schema Additions (Single Place)

All dashboard + agent-specific schema extensions must live only here.

### 9.1 `AgentSession`

- `id` (UUID, PK)
- `userId` (FK → User, nullable for guest sessions)

- `role` (from `User.role` or `customer_guest`)
- `channel` (enum: `chat`, `voice`)
- `linkedShipmentId` (FK → Shipment, nullable)
- `openAiSessionId` (string; OpenAI session identifier)
- `startedAt`
- `endedAt` (nullable)
- `status` (enum: `active`, `completed`, `error`)
- `lastAgentName` (e.g., `ShipmentTrackingAgent`)
- `transcript` (JSON or text)
- `outcome` (JSON; e.g., issueCreated, changeRequested, escalationTriggered)

### 9.2 `MetricDefinition`

- `id` (UUID, PK)
- `key` (unique; e.g., `on_time_delivery_rate`)
- `name`
- `description`
- `aggregationType` (enum: `ratio`, `count`, `avg`)
- `dimension` (enum/string: `global`, `region`, `route`, `driver`)
- `targetValue` (numeric)
- `warningThreshold` (numeric, optional)
- `criticalThreshold` (numeric, optional)
- `ownerRole` (usually `admin`)
- `isVisibleOnDashboard` (bool)
- `createdAt`, `updatedAt`

### 9.3 `MetricSnapshot`

- `id` (UUID, PK)
- `metricId` (FK → MetricDefinition)
- `value` (numeric)
- `timeRangeStart`
- `timeRangeEnd`
- `computedAt`
- `breakdown` (JSON; e.g., per region)

### 9.4 `DashboardConfig`

- `id` (UUID, PK)
- `ownerType` (enum: `role`, `user`)
- `ownerRole` (nullable)
- `ownerUserId` (nullable; FK → User)
- `layout` (JSON; widgets + arrangement)
- `createdAt`, `updatedAt`

---

## 10) WebSocket Events (Minimum Set)

Define and implement events (names are suggestions): - `shipment.scan.created` - `shipment.status.updated` - `issue.created` - `issue.updated` - `escalation.triggered` - `escalation.advanced` - `escalation.acknowledged` - `metrics.snapshot.created` (optional)

All events must be: - Authorized (RBAC) - Logged (at least in application logs)

---

## 11) Implementation Plan (Milestones)

### Milestone 1 — Database & Seed Data

- Implement Prisma models + migrations.
- Add seed script:
- 30–50 shipments with scan history
- Routes + route stops
- A few issues + escalation contacts
- Metric definitions + dashboard config defaults

### Milestone 2 — Backend REST + RBAC

- CRUD/queries for shipments, routes, issues.
- Escalation endpoints (trigger, advance, acknowledge).
- Metrics endpoints:
- `GET /metrics/overview`
- `GET /metrics/definitions` (admin edit)
- `POST /metrics/definitions` / `PATCH ...` (admin only)

### Milestone 3 — WebSockets

- Gateway + event publishing on:
- new scans
- issue updates
- escalation updates
- metric snapshot updates (optional)

### Milestone 4 — OpenAI Agents + Realtime (Chat + Voice)

- Define agents in OpenAI Agents.
- Define tools that map to backend operations.
- Implement backend “AI Orchestrator”:
- Starts/maintains chat sessions
- Starts/maintains Realtime voice sessions
- Logs AgentSession records
- Ensure chat and voice share the same routing + tool calls.

### Milestone 5 — Frontend Dashboards

- Customer dashboard with unified chat + voice widget.
- Admin/Ops dashboard with:

- Shipments, routes, issues, escalations
- Metrics overview + metrics admin editor
- WebSocket subscription handling.

**Milestone 6 — Demo + Documentation**

- README with run steps and demo scripts.
- 3 demo flows: 1) Track shipment → create issue (chat) 2) Request delivery change (voice) 3) SLA risk → escalation → manager ACK (dashboard)

---

## 12) Deliverables

1. **Backend repository** (NestJS + Prisma + Postgres)
2. **Frontend repository** (React + TS + Tailwind)
3. **OpenAI agent configuration** (documented prompts + tools)
4. **Seed data script** + DB migration files
5. **README** with setup + demo scripts

---

## 13) Evaluation Criteria

- Correctness of core flows (tracking, issues, changes, escalations).
- Clean architecture (domain boundaries, services, modules).
- RBAC correctness across REST + WebSockets.
- OpenAI-only compliance (Agents + Realtime for voice; no external AI).
- Professional UX for both customer and admin dashboards.
- Observability: logs + useful error handling.