Ride Hailing Platform

Software Architecture Design

Project Code: RHP

Document Code	**RHP_SAD**
Version	**1.1**
Effective date	**24/04/2026**

RECORD OF CHANGE

No	Effective Date	Version	Change Description	Reason	Creator	Reviewer	Approver
1	07/07/2025	0.1	Initial creation	Baseline for future changes	HaiNT188	LinhNMD, TrungTN6	
2	01/08/2025	0.2	Updated architecture diagram overview and Azure infrastructure architecture		HaiNT188	TrungTN6	
3	11/08/2025	0.3	Merged EventTracking module into CCRobotaxi service	To improve performance, simplify event processing, and streamline system architecture.	HaiNT188	TrungTN6	
4	14/08/2025	0.4	Merged updates from versions 0.2 and 0.3 into a unified document (version 0.4) and resolved customer feedback	Merged updates from versions 0.2 and 0.3 into a unified document (version 0.4) and resolved customer feedback	HaiNT188	TrungTN6	
5	27/08/2025	0.5	Update the customer feedback	Update the customer feedback	HaiNT188	TrungTN6	
6	29/09/2025	1.0	Update size and performance sections, and revise infrastructure, architecture diagram design to integrate with the Etisalat SMS provider, Mapbox	To improve system efficiency and scalability	HaiNT188	TrungTN6	
7	24/04/2026	1.1	Update SAD image and Deployment Image	To improve system efficiency and scalability	NamLT12	TrungTN6	

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Purpose

This document provides a comprehensive architectural overview of the **RHP (**short for **Ride Hailing Platform**), a ride-hailing system that uses autonomous vehicles (AVs) integrated with the **MOMENTA APIs**. The purpose of this document is to capture and communicate the key architectural decisions for the system, using a variety of views to illustrate system behavior, structure, performance, and integration.
It is primarily intended for use by:

- **Architects**: To understand and review the structure and decisions.
- **Developers**: To design and implement components accordingly.
- **Testers**: To verify system integrity and module interaction.
- **Stakeholders**: To understand capabilities and design trade-offs.

1.2 Scope

The document applies to the entire RHP **ride-hailing platform, encompassing:**

- Passenger and Operator mobile/web applications
- Backend services, orchestration, and data layers
- AV fleet coordination and MOMENTA system integration
- Real-time vehicle tracking and trip lifecycle management

This SAD covers both runtime and development-time views for the above components.

1.3 Definitions, Acronyms and Abbreviations

- AV: Autonomous Vehicle
- SAD: Software Architecture Document
- RHP: (Ride Hailing Platform)
- MOMENTA: External mobility provider
- API: Application Programming Interface
- E2E: End to End
- API: Application Programming Interface
- SLA: Service Level Assignment
- PC: Primary cases

1.4 References

- MOMENTA API Specifications
- IEEE 1471

1.5 Overview

This document is structured into multiple architecture views

- Logical View
- Process View
- Deployment View
- Implementation View
- Use-Case View

Additional sections describe system-level goals, data architecture, interface definitions, performance, quality attributes, and design trade-offs.

2 ARCHITECTURAL REPRESENTATION

The architecture of the **Ride-Hailing Platform (RHP)** is modeled using multiple views. The system integrates with the **MOMENTA Autonomous Vehicle platform** to support driverless ride dispatch, control, and live monitoring. Architectural views are chosen to represent functional, structural, behavioral, and deployment aspects of the system.

2.1 Use-Case View

This view represents the system's **functional requirements** from the perspective of key actors.

- Actors:
  - Passenger (via Flutter mobile app)
  - Fleet Operator (via Angular web dashboard)
  - Autonomous Vehicle (via MOMENTA AV API)
- Use Cases:
  - Request Ride
  - Assign & Monitor Vehicle
  - Track Trip
  - Emergency Override
  - Can view AV Video Stream in Fleet Management Portal
  - Complete Trip

2.2 Logical View

Describes the modular structure of the backend and domain models.

Core Services:

- TripService – Manages trip lifecycle and state machine
- DispatchService – Matches and assigns AVs
- CCRobotaxi – Interfaces with MOMENTA APIs
- UserService – Manages user data and authentication
- PaymentService – Handles payment processing
- NotificationService – Sends notifications to users

Domain Models:

- Trip, Vehicle, User, DispatchCommand, AVStatus, GeoZone

2.3 Process View

Models the system's **runtime behavior** and message/event flow.

- **Event Flow** (Pub/Sub via Service Bus)
  - TripRequested → DispatchService → MOMENTA AV assignment → TripAssigned
- WebSocket Communication:
  - Real-time AV telemetry and trip updates delivered via **Websocket**
  - WebRTC signaling messages for video streaming
- Concurrent State Machines:
  - Trip State: Requested → Assigned → InProgress → Completed
  - Vehicle State: Idle → EnRoute → Waiting → Driving → Parked

2.4 Deployment View

Shows how the application is deployed across Azure and integrated with external platforms.

Technology Stack:

- Frontend:
  - **Flutter** for Passenger App (iOS/Android)
  - **Angular** for Operator Dashboard (Web)
- Backend:
  - Spring Boot microservices deployed on **Azure App Services**
  - API Gateway using Azure API Management
- Real-Time:
  - **WebSocket** for bidirectional WebSocket communication
  - Groups: trip:{id}, user:{id}, operator:{fleetId}
- External Integration:
  - **MOMENTA AV API** for vehicle control (REST/WebSocket)
  - **MOMENTA Video Server** (WebRTC or RTSP for streaming)
- Storage:
  - Azure Blob Storage + CDN for assets, map files
- CI/CD:
  - Azure DevOps Pipelines

2.5 Implementation View

Details how code and infrastructure are organized for development and deployment.

- Repositories:
  - trip-service, dispatch-service, momenta-adapter, mobile-app, web-dashboard

- Languages:
  - Java (Spring Boot), Dart (Flutter), TypeScript (Angular)
- Project Layout:

```
├── backend/
│   ├── trip-service/
│   ├── dispatch-service/
├── clients/
│   ├── passenger-app/ (Flutter)
│   ├── operator-dashboard/ (Angular)
└── infra/
    ├── terraform/
    └── cicd/
```

- CI/CD Pipelines:
  - Flutter build ~~→ AppCenter / Firebase~~
    - Manual build for iOS → Upload to TestFlight (via Apple App Store Connect)
    - Manual build for Android → Upload to Google Play Console
  - Angular build → Azure Static Web Apps / Blob
  - Spring Boot → Azure App Services via Docker or JAR

3 ARCHITECTURAL GOALS AND CONSTRAINTS

3.1 Architectural Goals

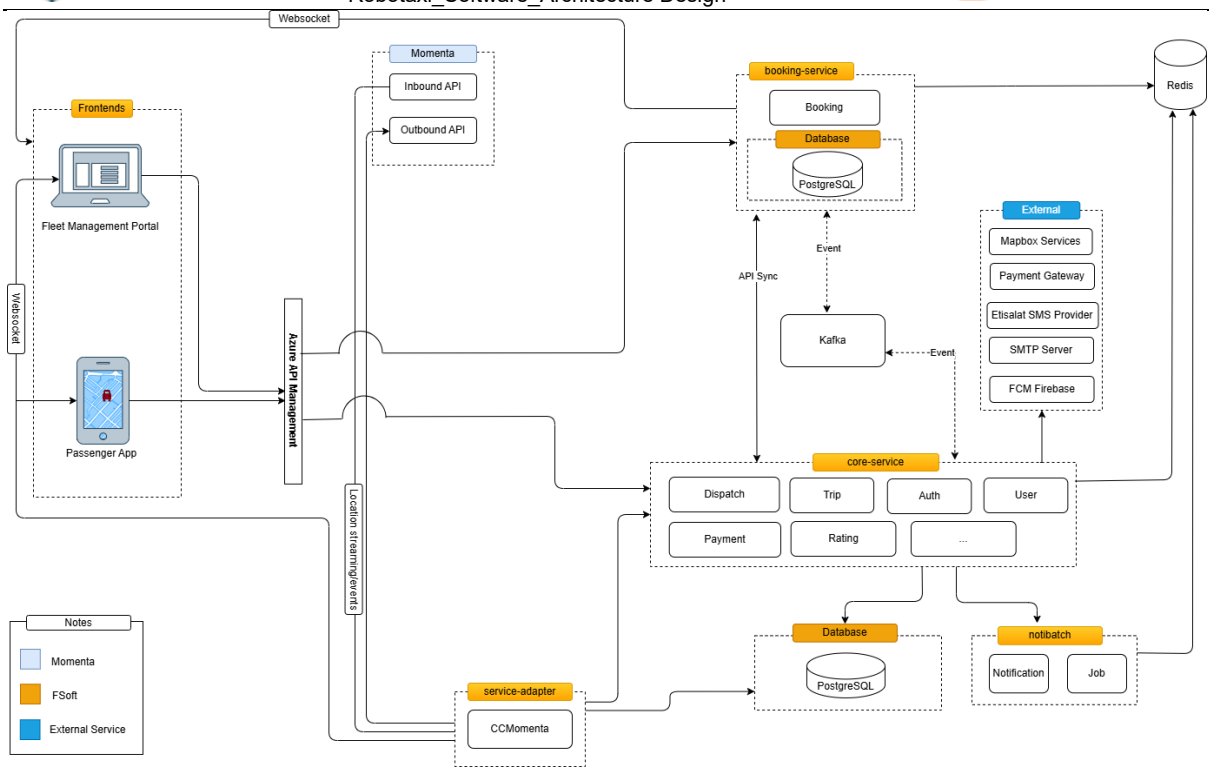
Goal	Description
Autonomous Vehicle Integration	Ensure seamless and secure interoperability with **MOMENTA APIs** for vehicle location tracking, trip control, and status updates.
Modular and Scalable Design	Design a modular architecture with the ability to scale horizontally to support real-time bookings and AV coordination under high user demand.
Real-time Communication	Enable low-latency, event-driven communication between backend services, passenger apps, and AVs using WebSocket and message queues.
High Availability	The system must provide **99.9%+ uptime**, with redundancy and auto-recovery strategies for critical services.
Secure Data Handling	Ensure data privacy, secure APIs, and encrypted communication between components, especially when handling personal and trip data.
Extensibility	Design the system to support plug-in integration for other AV vendors beyond MOMENTA, supporting future AV interoperability.
Geo-distribution	Support deployment across multiple regions, starting with PoC in UAE using cloud-native practices.

3.2 Architectural Constraints

Constraint	Description
Third-party Dependency (MOMENTA)	The system must conform to MOMENTA's interface specifications and update policies, including strict adherence to API request limits and formats.
Real-time Location Accuracy	GPS updates from AVs must be reflected to passengers with minimal delay (target: <10 seconds).
API Latency Budget	Round-trip latency from passenger request to AV command response must be <10 second for core operations.
Mobile-first Access	All passenger features are primarily accessed via mobile apps (iOS/Android), impacting UX and response constraints.
Data Compliance	All data operations must comply with local data protection laws (e.g., PDPL in UAE, GDPR in Europe if scaled).
Edge Computing Consideration	Some AV-side computations must run on embedded edge nodes with limited processing and memory capacity.
Versioning and Compatibility	Backend services must support backward-compatible API versions for mobile clients to allow gradual rollout of features.

4 SYSTEM ARCHITECTURE

Architecture Overview Diagram



5 USE-CASE VIEW

Primary Actors

Actor	Description
Passenger	End user who books and monitors AV rides using the mobile/web application.
AV (Autonomous Vehicle)	The vehicle executing the ride; integrated through MOMENTA APIs.
MOMENTA System	External AV control system for trip coordination, status updates, and telemetry.
Admin / Operator	Staff monitoring and managing system behavior, interventions, and configurations.

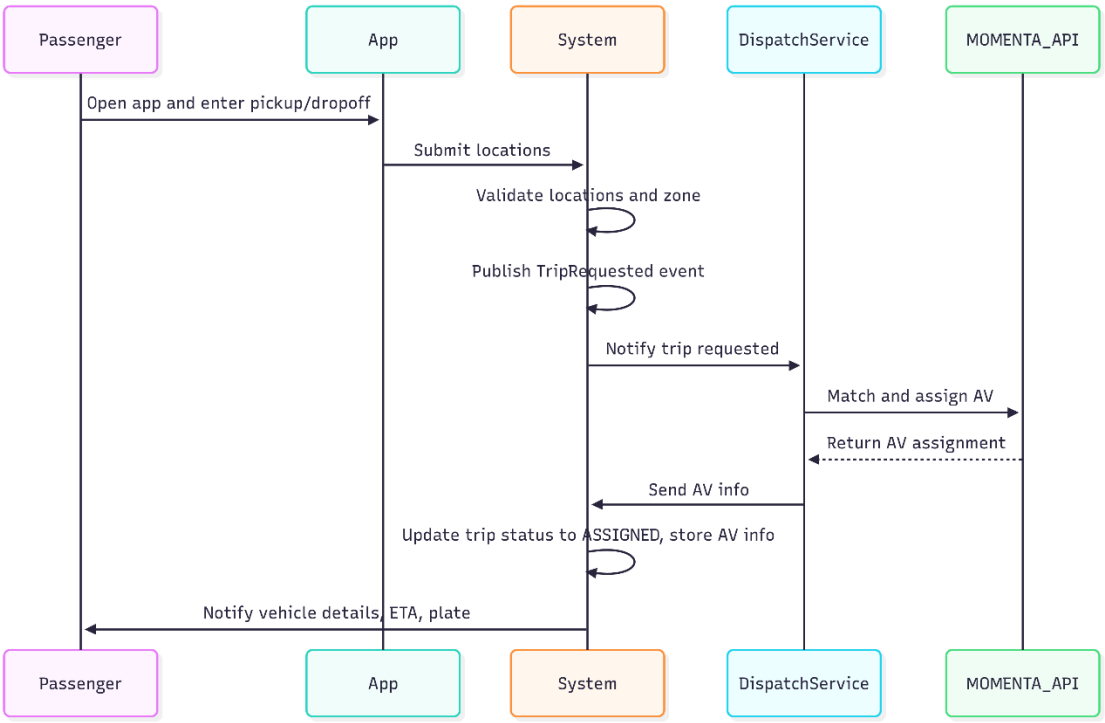
Primary Use Cases

PC-ID	Use Case Name	Description
PC-01	Book Ride	Passenger selects pickup/dropoff location and requests a ride. System matches with an available AV.
PC-02	Track AV	Passenger receives real-time vehicle location updates, route ETA, and AV status.
PC-03	Trip Execution	MOMENTA-controlled AV starts, progresses, and completes the trip while backend tracks each state.

PC-04	Cancel Ride	Passenger or system cancels ride before start. Cancellation is propagated to MOMENTA and other subsystems.
PC-05	Rate Trip & Payment	After completion, passenger can rate the ride and perform in-app payment.
PC-06	Operator Intervene	Admin interface allows operator to override or monitor trip status (e.g., for AV error or passenger issue).
PC-07	Monitor Fleet	Admin can track AVs in real-time, see ride history, AV health and telemetry logs.

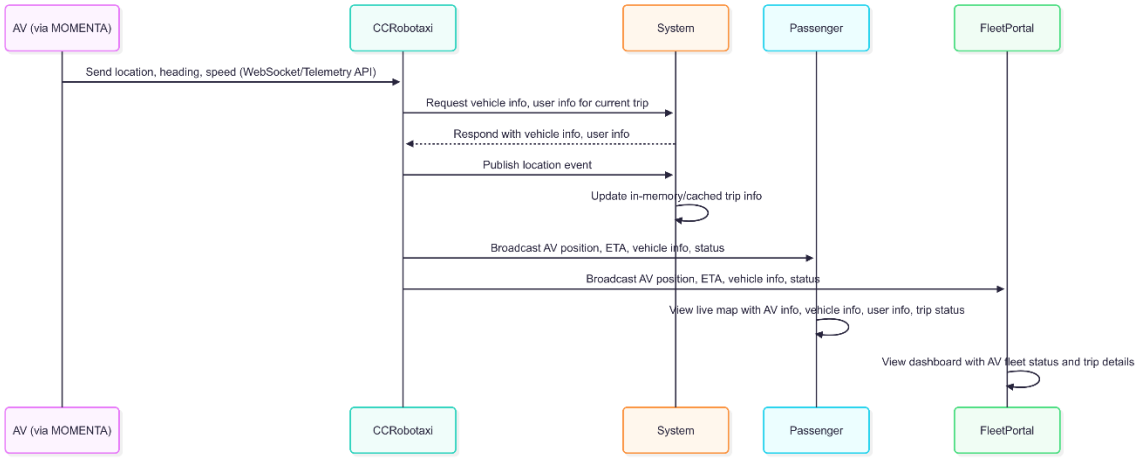
5.1 Use-Case Realizations

PC-01: **Book Ride**



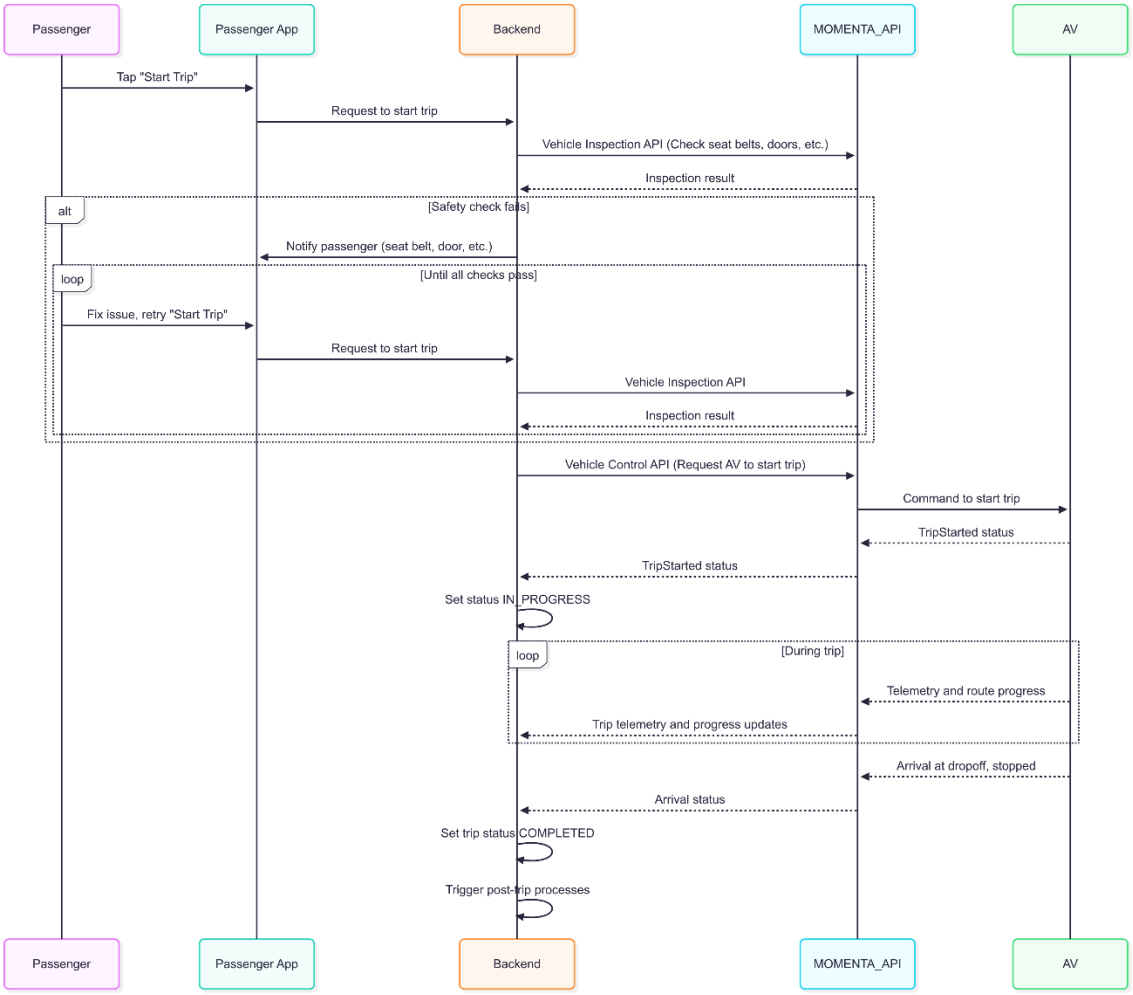
Step	Actor	System Action
1	Passenger	Opens app and enters pickup and dropoff locations
2	System	Validates input locations and zone availability
3	System	Publishes TripRequested event
4	DispatchService	Matches and assigns a suitable AV via MOMENTA API
5	System	Updates trip status to ASSIGNED, stores AV info
6	System	Notifies Passenger with vehicle details, ETA, and AV plate

PC-02: **Track AV**



Step	Actor	System Action
1	AV (via MOMENTA)	Sends location, heading, speed to CCRobotaxi via WebSocket/Telemetry API
2	CCRobotaxi	Requests vehicle info and user info for current trip from System
3	System	Responds to CCRobotaxi with vehicle info and user info
4	CCRobotaxi	Publishes location event to System
5	System	Updates in-memory or cached trip info
6	CCRobotaxi	Broadcasts AV position, ETA, vehicle info, and status to Passenger
7	CCRobotaxi	Broadcasts AV position, ETA, vehicle info, and status to FleetPortal
8	Passenger	Views live map with AV info, vehicle info, user info, and trip status
9	FleetPortal	Views dashboard with AV fleet status and trip details

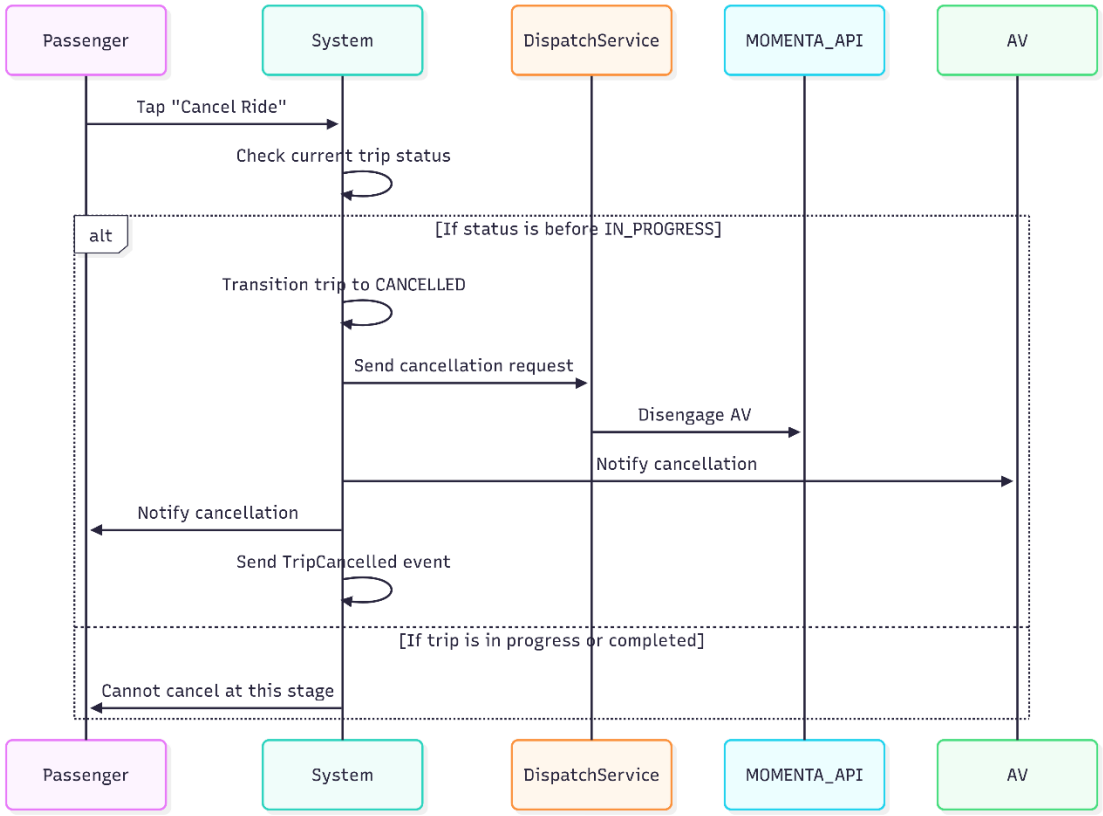
PC-03: **Trip Execution**



Step	Actor	System Action
1	Passenger	Taps "Start Trip" in Passenger App
2	Passenger App	Sends request to Backend to start trip
3	Backend	Calls MOMENTA_API Vehicle Inspection API (checks seat belts, doors, etc.)
4	MOMENTA_API	Returns inspection result to Backend
5	Backend	If safety check fails, notifies Passenger via Passenger App
6	Passenger	Fixes issue, retries "Start Trip"
7	Passenger App	Sends new request to Backend to start trip (repeats inspection steps until pass)
8	Backend	Calls MOMENTA_API Vehicle Control API to request AV to start trip
9	MOMENTA_API	Sends command to AV to start trip
10	AV	Reports TripStarted status to MOMENTA_API
11	MOMENTA_API	Passes TripStarted status to Backend
12	Backend	Sets trip status to IN_PROGRESS
13	AV	Sends telemetry and route progress to MOMENTA_API (during trip)

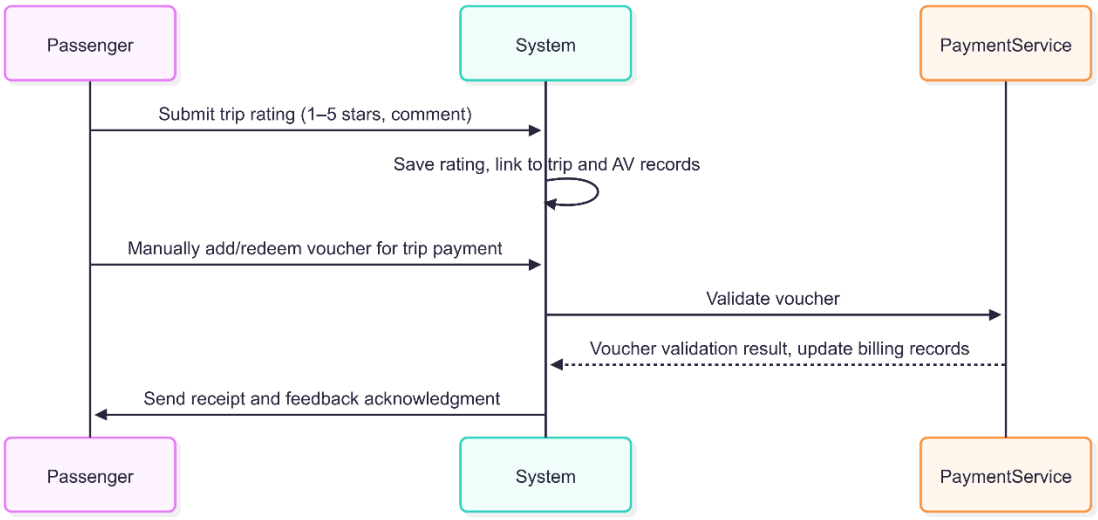
Step	Actor	System Action
14	MOMENTA_API	Forwards trip telemetry and progress updates to Backend (during trip)
15	AV	Reports arrival at dropoff and stops to MOMENTA_API
16	MOMENTA_API	Passes arrival status to Backend
17	Backend	Sets trip status to COMPLETED
18	Backend	Triggers post-trip processes

PC-04: **Cancel Ride**



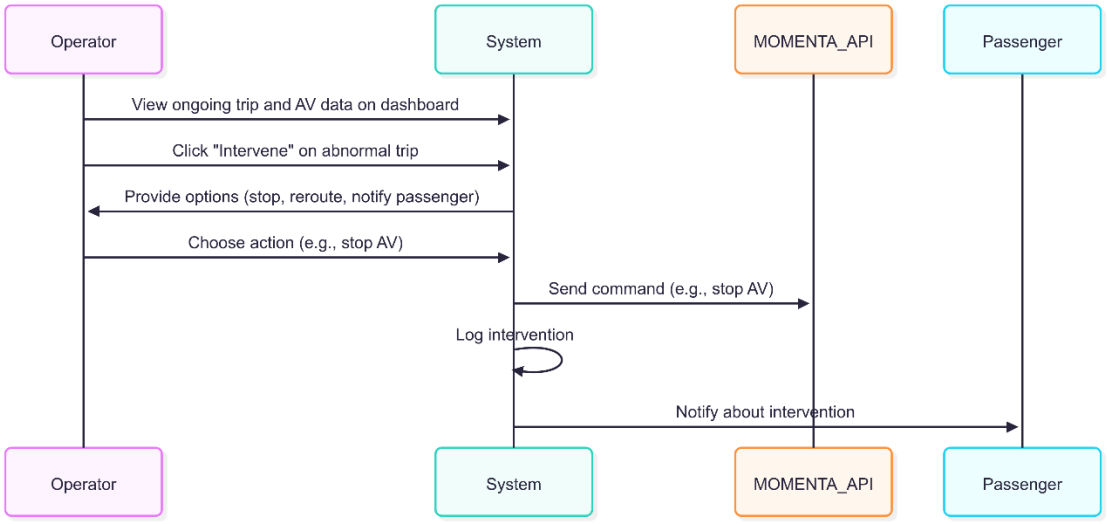
Step	Actor	System Action
1	Passenger	Taps “Cancel Ride” before AV arrives
2	System	Checks current trip status (must be before IN_PROGRESS)
3	System	Transitions trip to CANCELLED
4	DispatchService	Sends cancellation to MOMENTA API to disengage AV
5	System	Notifies AV and Passenger
6	System	Sends TripCancelled event

PC-05: **Rate Trip & Payment**



Step	Actor	System Action
1	Passenger	Opens trip summary and submits rating (1–5 stars, comment)
2	System	Saves rating and links to trip and AV records
3	Passenger	Manually adds/redeems voucher for trip payment (~~wallet/credit card~~)
4	PaymentService	Validates voucher and updates billing records
5	System	Sends receipt and feedback acknowledgment

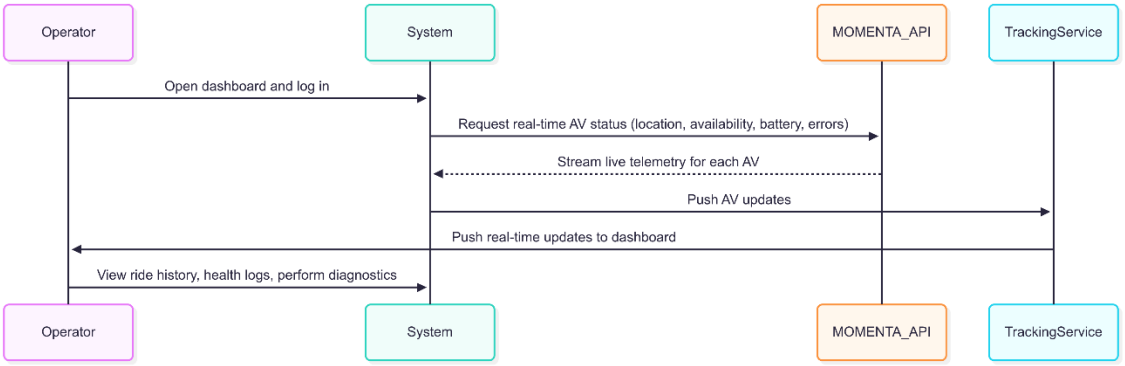
PC-06: **Operator Intervene**



Step	Actor	System Action
1	Operator	Views ongoing trip and AV data on dashboard
2	Operator	Clicks “Intervene” on trip in abnormal state
3	System	Provides options: stop vehicle, reroute, notify passenger
4	Operator	Chooses action (e.g., stop AV)

5	System	Sends command via MOMENTA API
6	System	Logs intervention, notifies passenger and AV

PC-07: **Monitor Fleet**



Step	Actor	System Action
1	Operator	Opens dashboard and logs in
2	System	Loads real-time AV status: location, availability, battery, errors
3	MOMENTA API	Streams live telemetry for each AV
4	Web PubSub	Pushes updates to Operator dashboard
5	Operator	Views ride history, health logs, and performs diagnostics

6 LOGICAL VIEW

6.1 Overview

The logical architecture is divided into the following high-level subsystems:

Subsystem	Description
Trip Management	Handles ride lifecycle, state transitions, and trip data persistence.
Dispatch Service	Matches ride requests with available AVs based on ETA and location.
AV Integration Layer	Wraps MOMENTA APIs for trip assignment, control, and AV telemetry.
Notification Service	Sends updates to passengers and operators via push or WebSocket channels.
User Services	Manages user profiles, preferences, and authentication.
Admin Services	Supports fleet monitoring, trip override, and manual AV assignment.
Data Layer	Manages persistent storage of trips, user actions, vehicle logs, and analytics.

Each subsystem is implemented as an independent module with well-defined interfaces, supporting future extensibility (e.g., integrating new AV providers).

6.2 Architecturally Significant Design Packages

Design Package	Description / Responsibility	Architectural Significance
TripService	Manages the full lifecycle of a trip (requested → assigned → in-progress → completed/canceled)	Central to core business logic; governs trip state machine and coordinates across Dispatch, AV, and Payment services
DispatchService	Matches passengers to AVs, selects optimal AV using MOMENTA APIs, and manages AV assignments	Integrates external AV API (MOMENTA), requires low latency, retries, and event-driven design
CCRobotaxi	Abstracts MOMENTA AV APIs (REST & WebSocket) for vehicle control, telemetry, and state updates	Encapsulates AV platform integration; isolates external dependency logic; needs API compatibility and fault isolation
UserService	Manages authentication, user identity (passenger/operator), roles, and profile data	AuthN/AuthZ is security-critical; user context used across other services
PaymentService	Handles trip payments, refunds, voucher redemption, and invoice generation during the POC phase.	Provides a secure and reliable payment workflow for core ride scenarios in POC
RatingService	Collects and stores passenger ratings and feedback	Supports user feedback features; uses standard data storage and processing
NotificationService	Sends real-time trip/AV events via push (FCM), SMS, WebSockets, or email	Cross-cutting real-time communication; impacts user/operator experience; high volume/low latency
FleetManagementService	Provides APIs and backend logic for the Fleet Management Portal (operator dashboard)	Supports operational control, AV intervention, fleet overview, and compliance; integrates with MOMENTA and core flows
RideHistoryService	Stores and retrieves past trip records, ratings, and incident reports	Required for analytics, compliance, and reporting; interacts with user and trip data
AuditLoggingService	Captures and persists actions/events for traceability (e.g., operator intervention, AV commands)	Crucial for compliance, debugging, and regulatory logs; must be immutable and secure
EventBusAdapter	Wraps access to Kafka/Event Bus for publishing and subscribing to domain events	Enables asynchronous service communication; improves system decoupling and failure isolation

CameraService	Handles proxy/signaling for MOMENTA AV video streams (e.g., RTSP, FLV, WebRTC)	Enables live monitoring and video features; needs secure, low-latency streaming; optional but vital for operations
TelemetryService	Collects and reports system/application metrics, logs, and health data	Enables monitoring, alerting, and operational insight; supports reliability and SRE practices
AnalyticsEngine	Processes, aggregates, and analyzes operational/business event data	Supports business intelligence, KPI tracking, and data-driven decision making

6.3 Interfaces And Connections

Service / Module	**Exposed Interface / Event**	**Communication Type**	**Protocol / Technology**	**Consumers / Dependencies**	**Notes**
TripService	POST /trip/book, GET /trip/{id}	REST API	HTTP (Spring Boot)	Passenger App, Fleet Management Portal	Handles trip lifecycle; sends and consumes domain events
	Event: TripRequested, TripCompleted, TripCancelled	Pub/Sub Event	Event Bus (Kafka)	DispatchService, PaymentService, NotificationService	Core event producer in ride lifecycle; sends ride-related events
	Consumes: TripAssigned, TripStarted, TripCancelled	Event Subscription	Event Bus (Kafka)	DispatchService, CCRobotaxi	Trip state machine update based on external triggers
DispatchService	POST /dispatch/assign	Internal API	HTTP	TripService (internal)	Matches trips to AVs via MOMENTA
	MOMENTA API: POST /order_event_notification, GET /order_details	External REST	HTTPS	MOMENTA	Dispatches AVs based on availability

Service / Module	Exposed Interface / Event	Communication Type	Protocol / Technology	Consumers / Dependencies	Notes
	Event: TripAssigned, DispatchFailed	Pub/Sub Event	Event Bus (Kafka)	TripService, NotificationService	Sends AV assignment events
	Consumes: TripRequested	Event Subscription	Event Bus (Kafka)	TripService	Starts dispatch flow on trip request
CCRobotaxi	MOMENTA Inbound API, Outbound API	External	MOMENTA Protocol	MOMENTA AV Platform	Streams location, battery, and AV status
	POST /vehicle/{id}/stop, /reroute, etc.	External REST	HTTPS	MOMENTA	AV command forwarding
	Event: TripStarted, TripCompleted, VehicleLocationUpdated, AVErrorDetected	Pub/Sub Event	Event Bus (Kafka)	TripService, TrackingService, FleetManagementService	Reports trip and AV state in real time; sends AV-related events
	WebSocket	WebSocket	WebSocket	Passenger App, Fleet Management Portal, Core Services	Ingests/processes AV live location and status updates; distributes via WebSocket
UserService	POST /auth/login, GET /user/{id}	REST API	HTTP	Passenger App, Fleet Management Portal	User identity, roles, and session auth
	Token validation for API Gateway	JWT / OAuth2	API Gateway	All services (via headers)	Validates roles for secure access
NotificationService	POST /notify/push, POST /notify/ws	Internal API	HTTP	All services (TripService, DispatchService, etc.)	Unified notification delivery
	FCM push for mobile, WebSocket for web	Push / WebSocket	Firebase, WebSocket	Passenger App, Fleet Management Portal	Cross-channel messaging bridge

Service / Module	Exposed Interface / Event	Communication Type	Protocol / Technology	Consumers / Dependencies	Notes
	Event: TripUpdateNotification, PaymentNotification	Pub/Sub Event	Event Bus (Kafka)	All services	Sends and receives notification-related events
PaymentService (POC)	POST /pay/charge, GET /pay/status/{tripId}	REST API	HTTP	TripService	Handles trip payment and refund using voucher system in POC phase
	Voucher redemption API	Internal API	HTTP	TripService	Validates and redeems vouchers; external payment provider not integrated in POC phase
	PaymentSuccess, PaymentFailed (direct response)	API Response	HTTP	NotificationService, RideHistoryService	Sends payment status directly via API response
FleetManagementService	GET /fleet/status, POST /intervene/{tripId}	REST API	HTTP	Fleet Management Portal	Control center API for interventions, fleet status, and AV control
	Event: InterventionIssued, InterventionLogged	Pub/Sub Event	Event Bus (Kafka)	CCRobotaxi, AuditLoggingService	Sends override commands and logs them
RideHistoryService	GET /ride/{userId}, POST /ride/save	REST API	HTTP	Fleet Management Portal, TripService	Stores and queries past trip records
	Event: TripCompleted, PaymentSuccess	Event Subscription	Event Bus (Kafka)	TripService, PaymentService	Updates history after trip/payment events; sends

Service / Module	Exposed Interface / Event	Communication Type	Protocol / Technology	Consumers / Dependencies	Notes
					ride completion events
AuditLoggingService	POST /audit/log	Internal API	HTTP	FleetManagementService, TripService	Persists secure logs of all critical actions; sends AV error and intervention logs
EventBusAdapter	publish(eventName, payload)	Internal Library	Event Bus (Kafka)	All services	Wraps message publishing and topic configuration; sends events to Kafka
CameraService	POST /camera/signal, GET /camera/relay/{tripId}	REST / WebRTC Signaling	WebSocket, HTTPS	Fleet Management Portal, AV (via MOMENTA)	Handles signaling/proxy for MOMENTA video stream (RTSP, WebRTC, etc.); sends camera events
TelemetryService	Collects system metrics, logs, health checks	Internal metrics/logs	Prometheus, ELK, etc.	All services	Enables monitoring, alerting, and operational insight; sends system health data
AnalyticsEngine	Processes event streams and historical data	Internal data processing	Batch/Stream Processing	All services, Data Warehouse	Supports business intelligence, KPI tracking, and reporting;

Service / Module	Exposed Interface / Event	Communication Type	Protocol / Technology	Consumers / Dependencies	Notes
					sends analytics results

7 PROCESS VIEW

7.1 Key Process Groups

Process Group	Description
Trip Orchestration Process	Manages full trip lifecycle, from request to completion, using asynchronous event flow and state transitions.
AV Communication Process	Sends commands and receives updates from MOMENTA's system in real time.
Notification Process	Handles real-time updates pushed to passenger and operator interfaces via WebSocket and push.
Event Dispatcher Process	Listens to the Event Bus and routes domain events to interested services.
Operator Monitoring Process	Runs in background, pulling AV status, vehicle health, and analytics for the Admin Portal.

7.2 Communication Mechanisms

Mechanism	Description
HTTP/REST	Used for external synchronous API calls (e.g., assigning trip to MOMENTA).
Webhooks	MOMENTA pushes vehicle updates to system via pre-registered HTTP endpoints.
Event Bus (Kafka)	Internal event propagation between services (e.g., trip state changes, AV position updates).
WebSocket	Enables real-time bidirectional messaging between backend and frontend clients (e.g., live AV tracking).
gRPC (optional)	Could be used for high-performance internal communication (future enhancement).

7.3 Process Interaction Example – Trip Flow

Here's an example of how multiple processes interact for a single ride:

1. **Passenger** submits a booking request → API Gateway routes to Trip Service.
2. Trip Service sends ~~emits~~ TripRequested event → picked up by Dispatch Service.
3. Dispatch Service → calls CCRobotaxi → MOMENTA AssignTrip API.
4. MOMENTA responds → TripAssigned event emitted.
5. Passenger is notified via **Notification Process**.
6. When AV arrives and trip starts, **MOMENTA** pushes TripStarted → CCRobotaxi handles → Trip Service updates state → notifies Passenger and Admin.
7. At trip end, same mechanism applies → TripCompleted → archived and analytics triggered.

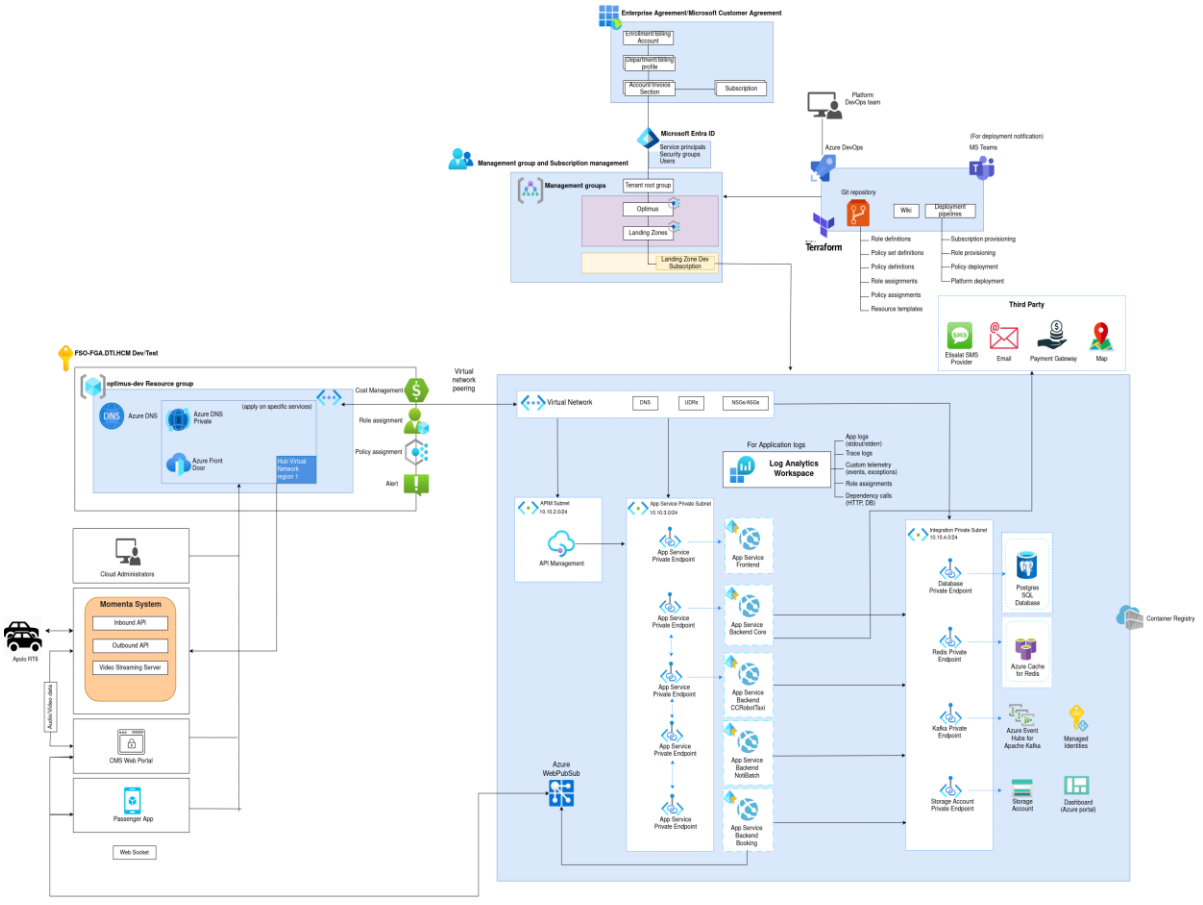
7.4 Concurrency and Isolation Considerations

- Each ride is treated as an **isolated logical unit** with its own lifecycle state and event stream.
- Services are **stateless** and horizontally scalable; session data is stored in Redis or DB.
- **Idempotency** is ensured in the event bus to tolerate retries and reprocessing (especially for external inputs).
- AV Adapter uses a **retry mechanism** with exponential backoff for failed MOMENTA API calls.

8 DEPLOYMENT VIEW

8.1 Target Deployment Environment

- **Mobile Clients**: Passenger apps deployed to iOS/Android and Fleet Admin deployed to Web
- **3rd-Party Services**: MOMENTA Cloud, Push Notification Providers, Mapbox.



8.3 Deployment Scenarios

Environment	Purpose	Notes
Development (DEV)	Feature development and unit testing	Owned by FPT; Isolated namespace, local DB, MOMENTA APIs mocked or stubbed
Testing (TEST)	System integration and functional testing	Owned by FPT; connected to MOMENTA sandbox; automated tests; CI/CD triggered deployments

Staging (UAT)	User acceptance testing and PoC validation	Owned by Kintsugi; production-like setup; pre-release sign-off; MOMENTA sandbox
Production (PROD)	Live deployment with AV fleet and real users	Owned by Kintsugi; high-availability, full monitoring, autoscaling, real MOMENTA integration

8.4 Network and Security Considerations

- **Encrypted traffic** via HTTPS and WebSocket Secure (WSS) across all communication lines.
- **Firewalls and VPC Peering** restrict exposure to MOMENTA IP ranges.
- **Authentication** using OAuth2/JWT for frontend-backend and backend-AV communication.

9 IMPLEMENTATION VIEW

9.1 Overview

The backend system is structured into 5 main layers:

1. **Controller Layer** – Exposes APIs (REST)
2. **Service Layer** – Contains business logic
3. **Repository Layer** (MyBatis) – Handles data access
4. **Integration Layer** – Communicates with external systems
5. **Shared Layer** – Reusable DTOs, utils, and error handling

Technologies Used

Area	Technology
Backend Framework	Java 21, Spring Boot, WebFlux
Persistence	MyBatis, Azure SQL
Messaging	**Kafka** (event-driven architecture)
Realtime & Push	**Google Firebase** (cloud messaging, analytics)
Maps & Location	**Mapbox API** (routing, reverse geocoding)
Notifications	SMS, Email, Real-time websocket.
API Documentation	Swagger/OpenAPI
Realtime UI	WebSocket
CI/CD	Azure Pipelines

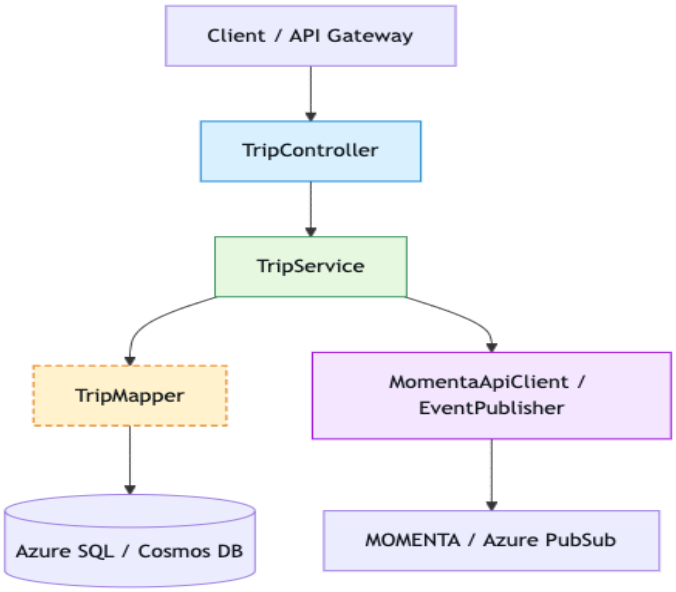
Layer Responsibilities

Layer	Description
Controller Layer	REST API endpoints using Spring MVC + Swagger annotations
Service Layer	Core logic, state transitions, Kafka event publishing
Repository Layer	SQL access via MyBatis mappers
Integration Layer	MOMENTA API/WebSocket, Kafka, Firebase, Mapbox API
Shared Layer	DTOs, exceptions, response wrappers, utilities

Layer Rules

- Controllers → call Services
- Services → call Repositories + Integration
- Integration handles external APIs and messaging
- Shared code is used by all other layers

Component Diagram



9.2 Layers

9.2.1. Presentation Layer

- **Description**: Entry point for HTTP/REST and WebSocket requests.
- Typical Modules:
  - TripController.java
  - NotificationGateway.java
- **Responsibilities**: Validate input, invoke business services, return responses.

9.2.2. Service Layer

- **Description**: Contains core application logic and orchestrates domain workflows.
- Modules:
  - TripService
  - DispatchService
  - TripStateMachine
- **Responsibilities**: Manage trip lifecycles, dispatch AVs, trigger events.

9.2.3. Integration Layer

- **Description**: Adapters to communicate with external systems (e.g., MOMENTA).

- Modules:
  - CCRobotaxi
  - MapboxService
- **Responsibilities**: Convert internal requests to external formats, handle retries and auth.

9.2.4. Data Access Layer

- **Description**: Manages persistent storage operations.
- Modules:
  - TripRepository
  - AVStatusRepository
- **Tech Stack**: Mybatis + PostgreSQL + Redis
- **Responsibilities**: CRUD operations, indexing, transaction boundaries.

9.2.5. Domain Model Layer

- **Description**: Defines business entities and state models.
- Entities:
  - Trip, Passenger, AV
  - TripStatus Enum
- **Responsibilities**: Rich domain modeling, encapsulate rules and constraints.

9.2.6. Shared Libraries

- **Description**: Cross-cutting concerns reused across layers.
- Modules:
  - EventBusPublisher
  - LoggerUtil
  - AuthContextResolver
  - CryptoUtil
  - RolesUtil
  - Slf4j + AOP for auto logging
  - Swagger
- **Responsibilities**: Messaging, logging, authentication, monitoring.

Code Repository Structure (Sample)

/rhp-backend

```
├── api-gateway/
├── trip-service/
├── dispatch-service/
├── CCRobotaxi/
├── notification-service/
├── shared-lib/
```

```
├── data-access/
├── docker/
└── helm-charts/
```

Build and Deployment Tools

Build Tools

Tool	Purpose
Maven	Build backend microservices (Spring Boot)
Flutter CLI	Build and compile the passenger mobile app
Angular CLI	Compile the operator web console
MyBatis	SQL mapping and persistence layer
Swagger/OpenAPI	Auto-generate REST API documentation and stubs

CI/CD Pipeline

Tool / Platform	Role
Azure Pipelines	Deploy to environments (DEV, TEST, UAT, PROD)
Docker	Containerization of backend services
Azure Container Registry (ACR)	Store Docker images
Firebase Console	Manage notifications, analytics, crash logs

Deployment Targets

Environment	Purpose	Deployment Target
DEV	Feature development	Azure App Services (Staging)
TEST	Functional & integration testing	Azure App Services (Test slot)
UAT	User acceptance testing	Azure App Services (UAT slot)
PROD	Production for live users/AV fleet	Azure App Services (Prod slot)

Configuration Management

- Environment-specific properties managed via **Spring Profiles**
- Secrets and API keys stored securely in **Azure Key Vault**
- Build variables injected through CI/CD pipelines

Monitoring and Logging

Tool	Description
Azure Monitor	Metrics, health checks, alerts
Firebase Analytics	User and mobile app analytics
Application Insights / ELK *(optional)*	Centralized logging and tracing

10 DATA VIEW (OPTIONAL)

Data Architecture Overview

The system uses a **polyglot persistence model**, with a combination of:

- **Relational Database (PostgreSQL)** for structured data like trips, users, and transactions.

- **In-memory Cache (Redis)** for transient data like session tokens, WebSocket subscriptions, and dispatch buffers.
- **AZURE Blob Storage** for logs, snapshots, avatar, images, AV diagnostics, and backup.

Data Retention and Archival Strategy

Data Type	Retention	Notes
Trips & Payments	3 years	Required for legal/audit purposes
Location Trace	6 months	Then archived to cold storage
Event Logs	12 months	For analytics/debugging
Notifications	1 month	Short-lived logs only
AV Telemetry	3 months	With exception for error traces

Compliance

- Data access is logged and audited.

11 SIZE AND PERFORMANCE

Expected Workload

Metric	Achievable Target	Notes
AVs Managed	500 AVs	2–3 zones, ~170–250 AVs per backend node
Concurrent Users	2000 users	Stateless backend with horizontal scaling (App Services, Redis, JWT)
Peak Ride Requests	~1 request/sec	Kafka + autoscaled APIs handle this reliably (even on peak traffic)
Daily Completed Trips	10,000–12,500 trips/day	Assumes ~20–25 trips/vehicle/day, which is high but realistic in dense city zones
Location Updates	10,800,000 updates/day	Every 4 sec/AV: 500 × 21,600 updates/AV/day
Notifications Sent	12,000–25,000/day	Scaled by trip count

Performance Requirements

Requirement	Target	Detailed Justification & Architecture Impact
Ride Booking Latency	≤ 1000 ms total (from request to dispatch)	- API layer: ≤ 200 ms - Kafka enqueue: ≤ 100 ms - Dispatch (Redis lookup + AV selection): ≤ 500 ms (async) - Matching logic: Redis confirmation ≤ 200 ms - System tuned to reliably deliver sub-1s responses
WebSocket AV Location Update Latency	≤ **4 seconds** end-to-end	- Backend receives and processes AV location updates every 4 seconds - Backend → WebSocket: ≤ 1s - WebSocket delivery: ≤ 800 ms (includes client jitter)

		- 500 vehicles × 21,600 updates/AV/day = 10,800,000 updates/day, easily handled at scale
API Response Time	< 1500 ms (99% requests)	- Book/cancel/status endpoints - Stateless Spring Boot controllers - Redis cache for frequent lookups - No DB blocking writes - Connection pooling, lazy init - Scale easily for 2,000 concurrent users
Database Write Throughput	≥ 3,000 – 4,000 writes/sec (across all tables)	- **Trip events** (create, assign, complete): ~1,000 TPS - **Telemetry logs**: batch insert every N seconds - Use MyBatis batching, partitioned write-heavy schema - Azure SQL with provisioned IO + auto-indexing
System Uptime (Monthly SLA)	≥ **99.5%** (allows ~3.6 hrs downtime/ month)	- **Azure Zone Redundancy + Front Door failover** - Container auto-restart on AKS/AppService - Daily rolling updates with **zero-downtime deployment** (blue-green) - Acceptable for PoC or early-scale fleet
WebSocket Connection Capacity	≥ 3,000 concurrent/ region	- 500 AVs + 2,000 users - 1,000–1,500 connections per node - Delta publishing, sharding not required at this scale
Cold Start Time for Services	< 10 seconds	- Spring Boot cold start: 5–10s (standard) - Optimize using: • lazy bean loading • fast classpath scanning • pre-warmed container pools on Azure - Initial TLS handshake adds ~100–200ms for HTTPS services

12 QUALITY

Reliability

- **Redundancy** is built in at every critical layer: service replicas, multi-AZ databases, and high-availability messaging.
- **Circuit breakers and retries** are implemented for MOMENTA API integration to tolerate external service instability.
- **Service degradation** paths allow fallback responses in case of partial failures.

Scalability

- **Horizontally scalable microservices** enable elastic response to increasing load.
- **Load balancing and autoscaling** strategies are applied based on real-time system metrics.
- Backend services, WebSockets, and databases are tuned to support millions of daily requests and updates.

Security

- **OAuth2.0/JWT**-based access control for mobile and operator apps.
- **Mutual TLS** between services and encrypted communication (HTTPS, WSS).
- **Role-based access control (RBAC)** in Admin Portal to limit operator permissions.
- **Input validation, rate limiting**, and **audit logging** guard against abuse and injection attacks.
- **Secrets and credentials** are managed securely via vaults or cloud-native secret stores.

Extensibility

- **Plug-in architecture** for AV integration (e.g., MOMENTA, other vendors in future).
- Modular design allows adding new services (e.g., Loyalty, Promotions) with minimal impact.
- Event-driven architecture enables asynchronous, non-blocking extension points.

Maintainability

- Consistent layered codebase and naming conventions.
- Clean separation of controller, service, and repository layers for testability.
- Code linting, static analysis, and automated unit/integration tests in CI/CD pipeline.
- **Documentation** generated automatically from annotations (e.g., OpenAPI/Swagger for REST APIs).

Portability

- All components are containerized using Docker.
- Can run on Azure, other clouds, or on-prem if needed.
- Uses open technologies (Spring Boot, PostgreSQL, Kafka).

Observability

- Dashboards configured for:
    - API latency, error rate, uptime
    - Trip state distributions
    - MOMENTA integration errors
    - AV telemetry quality

13 MAKE, BUY OR RE-USE CONSIDERATION

Component Evaluation Table

No	Component	Make	Buy	Re-use	Notes
1	TripService	☑			Governs trip lifecycle and state machine; coordinates Dispatch, AV, Payment, and history
2	DispatchService	☑			Matches passengers to AVs; integrates MOMENTA APIs; event-driven, low latency
3	CCRobotaxi	☑			Abstracts MOMENTA AV APIs (REST/WebSocket); vehicle control, telemetry, status
4	UserService	☑			AuthN/AuthZ via JWT; manages user identity, roles, profiles
5	PaymentService	☑			POC phase: processes trip payments and refunds using voucher system; no external payment provider integration
6	NotificationService	☑			Push (FCM), SMS, WebSockets, email; high volume, real-time
7	FleetManagementService	☑			Backend APIs/logic for fleet control, operator actions, compliance
8	Fleet Management Portal	☑			Web frontend for operators; visualizes fleet, powered by FleetManagementService
9	RideHistoryService	☑			Stores/retrieves trip records, ratings, incident reports
10	AuditLoggingService	☑			Immutable, secure event/action logs for compliance and debugging
11	EventBusAdapter	☑			Wraps Kafka/Event Bus for async communication, decoupling
12	CameraService	☑			Proxy for MOMENTA AV video streams (RTSP, FLV, WebRTC); secure, low-latency
13	TelemetryService	☑			Collects/reports metrics, logs, health data; supports monitoring/SRE
14	AnalyticsEngine	☑			Processes and aggregates event/business data for BI/KPI
15	API Gateway		☑		Azure API Management (APIM)
16	Monitoring Stack			☑	Azure Insights/Monitoring
17	CI/CD Pipeline			☑	Uses corporate DevSecOps best practices & templates
18	Database & Storage Layer	☑			PostgreSQL + custom schema for trip, user, history, logs, etc.
19	Passenger Mobile App	☑			iOS/Android app for booking, tracking, notifications

No	Component	Make	Buy	Re-use	Notes
20	Regulatory & Safety Compliance	☑			Data privacy, audit logs, incident reporting, insurance, user consent; vehicle-side via MOMENTA
21	Disaster Recovery/Backup			☑	Automated backups, failover
22	Rate Limiting/Abuse Prevention	☑			API protection
23	Customer Support/Helpdesk		☑		Zendesk, Intercom, or custom
24	Map & Routing Service		☑		Mapbox, or custom

Summary of Decisions

- **Make**: Critical application logic, including trip lifecycle, AV dispatching, user management — ensuring deep control, flexibility, and domain knowledge ownership.
- **Buy**: Commodity services like API gateway, mobile push services, and AV platform access are purchased to reduce complexity and focus on core value.
- **Re-use**: DevOps pipelines, observability tools, and shared utilities are reused across FPT projects to ensure consistency and reduce effort.