System Design - End Term - Yerlan Dias

Github repo link

TASK2.

Functional requirements

1. Demand forecasting and analysis the real-time passenger data and spatial trends(peak hour hotspots)
2. Dynamic allocation of buses (both self-driving and traditional)
3. Ingesting/processing of live traffic data
4. Autonomous Fleet monitoring - map position, health and diagnostics
5. Route scheduling based on real-time data
6. Vehicle status reporter - anomalies, breaks etc.
7. Passenger information system - live ETAs, bus capacity status, and route changes on mobile/web apps

8. Energy and charging management for autonomous vehicles

Non-functional requirements:

1. < 200ms for most API interactions (real-time routing, vehicle updates, dashboard interactions).
   ~500ms acceptable for less frequent operations (e.g., user registration, profile updates)
2. 250 GB x 3 HDDs per node (replicated for fault tolerance)
   16 GB RAM per server instance to support in-memory caching (Redis), stream processing windows, and ML inferences.
3. Must support 2000+ concurrent connections

4. Each core service (optimizer, dispatcher, etc.) runs on 8 vCPUs minimum
5. System uptime >= 99.9%, operate during peak hours
6. Fleet decision within 10 seconds of new data
7. Encryption between vehicles communication
8. Authentication(OAuth2, JWT)
9. Modular architecture to allow plugging in new vehicle types
10. Interoperability: compatible with multiple AV vendors
11. Data quality - quality data ingestion pipelines with deduplication and validation

The system integrates real-time data from passengers, vehicles, and external sources to make dynamic decisions about vehicle allocation, routing, and scheduling.

Data Ingestion Layer - Purpose: Collect data from multiple sources and funnel it into the processing backbone. Sources:

1. Rider mobile apps (trip requests, GPS, feedback)
2. AV telemetry systems (vehicle location, status, faults)
3. City data feeds (traffic, weather, events)

Tech Stack: Kafka (event streaming), REST API Gateway (secured with TLS and OAuth2), optional MQTT for lightweight telemetry.

2. Stream Processing & Event Handling Purpose: Handle and process incoming data streams in near-real-time. Key Tasks:

1. Parse and enrich data (e.g., geolocation to zone mapping)
2. Apply filters and transformations
3. Detect patterns and anomalies

Tech Stack: Apache Flink or Spark Streaming Pattern Used: Observer (reactive processing), Event Sourcing (Kafka log storage)

3. Demand Prediction EnginePurpose: Forecast short-term demand to guide fleet optimization. Inputs:

1. Historical ridership data
2. Current trip requests
3. Traffic and event data

Techniques: Time-series ML models (e.g., LSTM, Prophet), clustering of high-demand zones.

Pattern Used: CQRS (separation of training vs query path for predictions)

4. Fleet Optimization & SchedulingPurpose: Make decisions about which vehicles go where, and when. Functions:

1. Route re-planning based on demand and congestion
2. Adjust schedules dynamically
3. Balance load across the fleet (autonomous and traditional)

Design Features:

1. Zone-based agent controllers
2. Prioritization of high-demand corridors
3. Safety and certification-aware planning for AVs

Patterns Used: Strategy, CQRS, Bulkhead (to isolate failures across control zones)

5. Autonomous Vehicle Control Interface Purpose: Communicate with the self-driving buses safely and effectively.

Tasks:

1. Send routing instructions
2. Receive health & status telemetry
3. Monitor AV behavior and emergency alerts

Security: Circuit breakers, API key enforcement, TLS encryption Pattern Used: Adapter, Circuit Breaker

6. Vehicle Monitoring & Alerting

Purpose: Track the state of every vehicle in real time. Functions:

1. Visualize vehicle location and status
2. Trigger alerts (e.g., emergency stop, low battery)
3. Log data for post-mortem analysis

Pattern Used: Observer, Pub/Sub

7. Data Storage

1. Hot Path (real-time): Redis or Cassandra for fast lookups (vehicle position, ETAs)
2. Cold Path (historical): Data Lake (e.g., S3 + Parquet or BigQuery) for reports, training

   Security: Encryption at rest, access control

Compliance: Data anonymization, GDPR tagging

8. Control Dashboard & Visualization

For Dispatchers:

1. Map-based interface showing live vehicle positions
2. Alerts for crowding, delays, or faults
3. Tools to override routes if needed

For Riders (Mobile App):

1. Live arrival times
2. Vehicle occupancy indicators
3. Trip booking and feedback

Pattern Used: Backend-for-Frontend (custom APIs for each UI)

9. Security & Compliance Layer (Cross-Cutting)

Enforced Throughout:

1. TLS encryption in transit
2. OAuth2 + API key access
3. Role-based access control
4. GDPR compliance for all user data
5. Safety enforcement for certified AV behaviors

Safety considerations and how the architecture resolves them

1. Real-Time AV Monitoring & Telemetry

- Concern: Autonomous vehicles must be continuously monitored to ensure operational safety and detect issues like system faults or environmental hazards.
- Solution: The architecture includes a dedicated Vehicle Monitoring Service that receives telemetry data (e.g., GPS, LIDAR, system health) from each AV. It uses real-time alerting and anomaly detection to notify the Fleet Control Center of potential issues.
- Trade-off: Processing large volumes of data can increase system load.
- Mitigation: Uses scalable stream processing frameworks (e.g., Apache Flink) and only forwards alerts, not raw data, for real-time decisions.

2. Fail-safe AV Control Interface

- Concern: AVs must not accept unsafe commands or operate outside safety boundaries.
- Solution: The AV Command API Layer includes TLS encryption, circuit breakers, authentication (OAuth2/API keys), and strict command validation. Only pre-approved commands (reroute, stop, park) are allowed.
- Trade-off: Adds latency to control command execution.
- Mitigation: Critical commands are routed through edge nodes for faster execution; central control used for non-urgent operations.

3. Route Optimization with Safety Constraints

- Concern: Routes must avoid restricted areas (e.g., schools, construction) and account for changing traffic conditions.
- Solution: The Route Optimization Engine integrates live traffic APIs and map overlays (e.g., geofences, safety zones). It applies route constraints to avoid high-risk areas.
- Trade-off: May produce less optimal routes in terms of time or efficiency.
- Mitigation: Configurable policies allow trade-offs between speed and safety; human override is possible with audit logging.

4. Human-in-the-Loop Controls

- Concern: The system should not rely entirely on autonomy in critical or ambiguous situations.

- Solution: The Control Center Dashboard provides real-time status of AVs and allows manual overrides and decision support tools for dispatchers.
- Trade-off: Manual intervention can slow down reaction time.
- Mitigation: Used only when anomaly thresholds are crossed; otherwise, the system operates autonomously.

5. Data Security & Privacy

- Concern: Real-time tracking and user data involve sensitive information (GDPR, privacy laws).
- Solution: The Security & Compliance Layer enforces TLS encryption, role-based access control, and GDPR-compliant data handling (e.g., anonymization, user consent).
- Trade-off: Adds complexity to data pipelines and storage.
- Mitigation: Separates raw and aggregate data paths, applies pseudonymization at ingestion, and ensures auditability.