City-Scale Real-Time Transit Tracking & ETA Service

Submitted By

Name: Eshan Thakur

Roll No.: 23

Registration No.: 12410061

Course: B. Tech in Generative AI

Semester: 3

Session: 2024–2025

College: Lovely Professional University

---

1 — Requirements Pack

1.1 Stakeholders

Real-time transit systems involve multiple parties with different expectations. Identifying stakeholders clarifies requirements and helps define priorities.

Stakeholder	Description	Goals / Expectations
Commuters (Passengers)	Every day public transport users	Accurate arrival times, reliable tracking, alerts for delays
Transit Authority	Organization operating public transport	Monitor fleet, optimize performance, reduce delays
Drivers & Vehicle Operators	Bus/train staff responsible for movement	Simple reporting tools, route guidance, automatic updates
System Administrator	Manages platform access, data, and configurations	Stable secure system, tenant management, monitoring

Stakeholder	Description	Goals / Expectations
Developers & Engineers	Build and maintain system features	Scalable, modular code and easy integration
Data Analysts	Analyses usage and performance data	Insights on ridership, congestion, peak times
Third-Party Integrators	Navigation apps or government systems	API access to normalized transit data

---

1.2 User Stories

User stories capture requirements from different perspectives using simple natural language.

Role	User Story
Commuter	"As a user, I want to see the live location of my bus so I can avoid waiting at the stop."
Commuter	"As a user, I need ETA predictions so I can plan arrival time."
Admin	"As an admin, I want to configure multiple cities and fleets independently."
Driver	"As a driver, I want automatic GPS data updates without manual input."
System	"As a system, I should notify users when a route is delayed, cancelled, or diverted."

1.3 Functional Requirements

• Real- time GPS shadowing for all vehicles.

• Hunt functionality by stop, vehicle number, or route name.

• ETA computations grounded on live business, history, and speed.

• cautions for service interruptions, breakdowns, or route changes.

• Support for multilingual UI for availability.

• part- grounded access control for admin, drivers, and end druggies.

1.4 Non-Functional Requirements

Category	Requirement
Performance	Average latency <200ms for read requests
Availability	99.99% uptime SLA
Scalability	Horizontal scaling for millions of requests
Data Accuracy	ETA acceptance tolerance ± 1–2 minutes
Security	OAuth2, mTLS, encrypted data storage
Usability	Intuitive interface accessible to all ages
Portability	Cloud-ready and cross-platform mobile apps

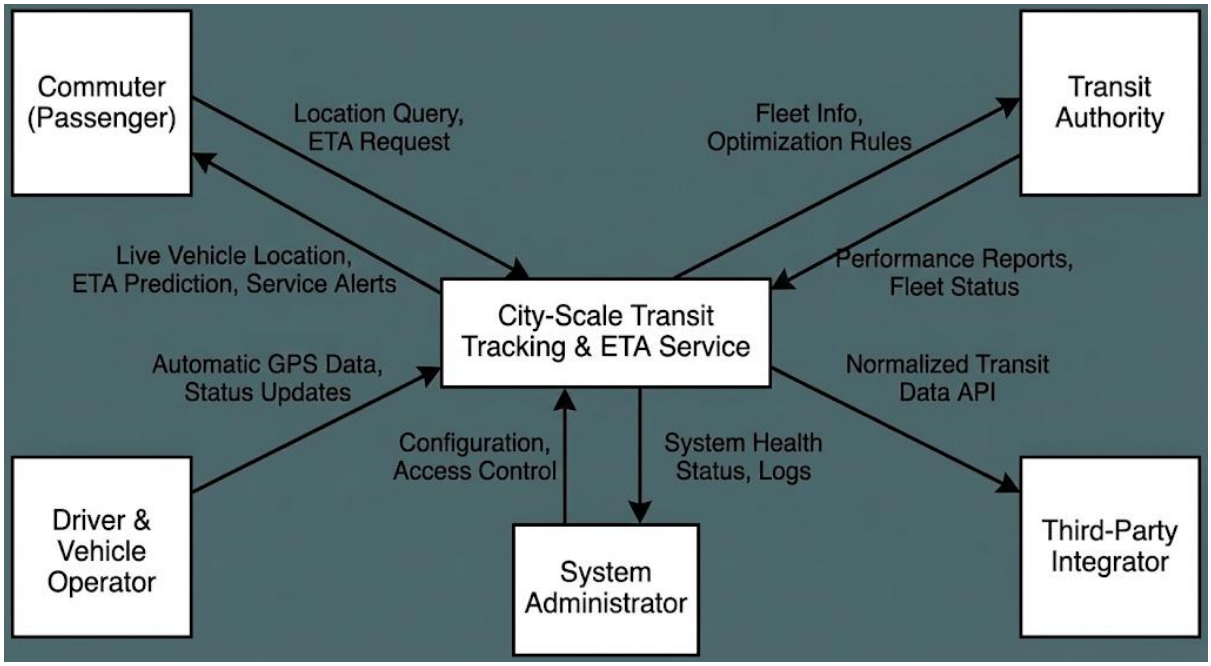
1.5 Constraints

- GPS signal loss in tunnels or rural areas.
- Network latency can impact telemetry.
- Budget restrictions for IoT hardware across fleets.
- Regional compliance (NDHM, GDPR optional).

1.6 Assumptions

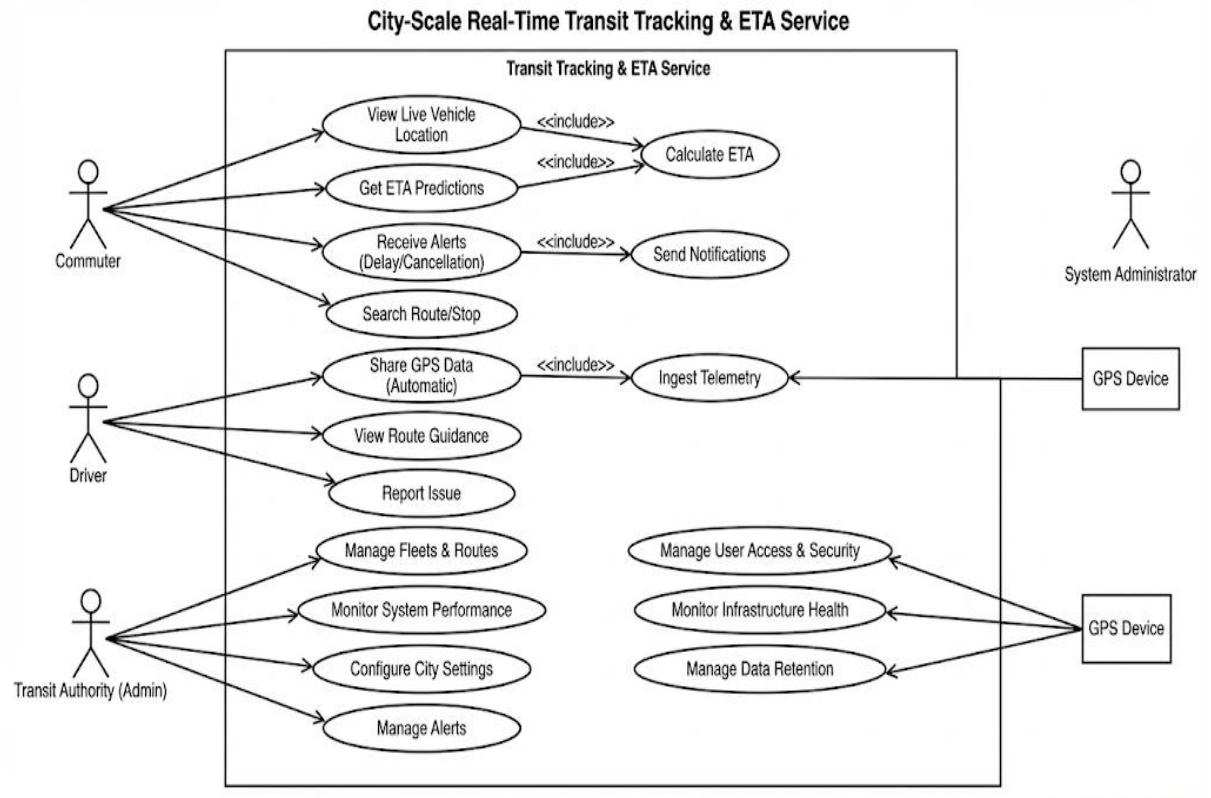
- All vehicles are fitted with supported GPS devices.
- Internet connectivity is available in transit environment.
- All cities share standardized GTFS format.

1.7 Context Diagram



This diagram will show how external entities such as users, vehicles, and administrators interact with the core system using APIs, telemetry ingestion services, and notification engines.

1.8 Use-Case Diagram



2 — Architecture

2.1 Architecture Style Comparison

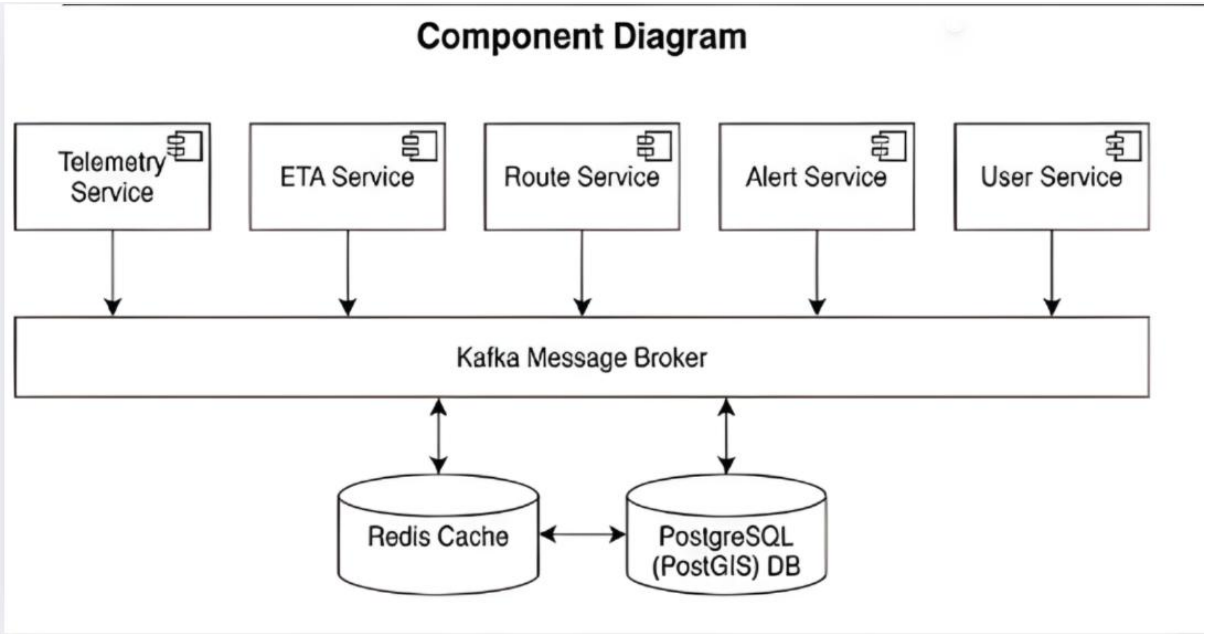
To choose the best architecture, we evaluated three approaches:

Criteria	Monolith	Microservices	Serverless
Scalability	❌ Difficult	✅ Highly scalable	⚠️ Good but costly
Deployment Speed	✓ Easy	⚠️ Needs orchestration	✓ Easy but cold start issues
Fault Isolation	❌ Weak	✅ Excellent	⚠️ Medium
Real-Time Streaming Suitability	❌ Poor	✅ Ideal	⚠️ Limited
Multi-Tenant Support	❌ Limited	✅ Best choice	⚠️ Possible

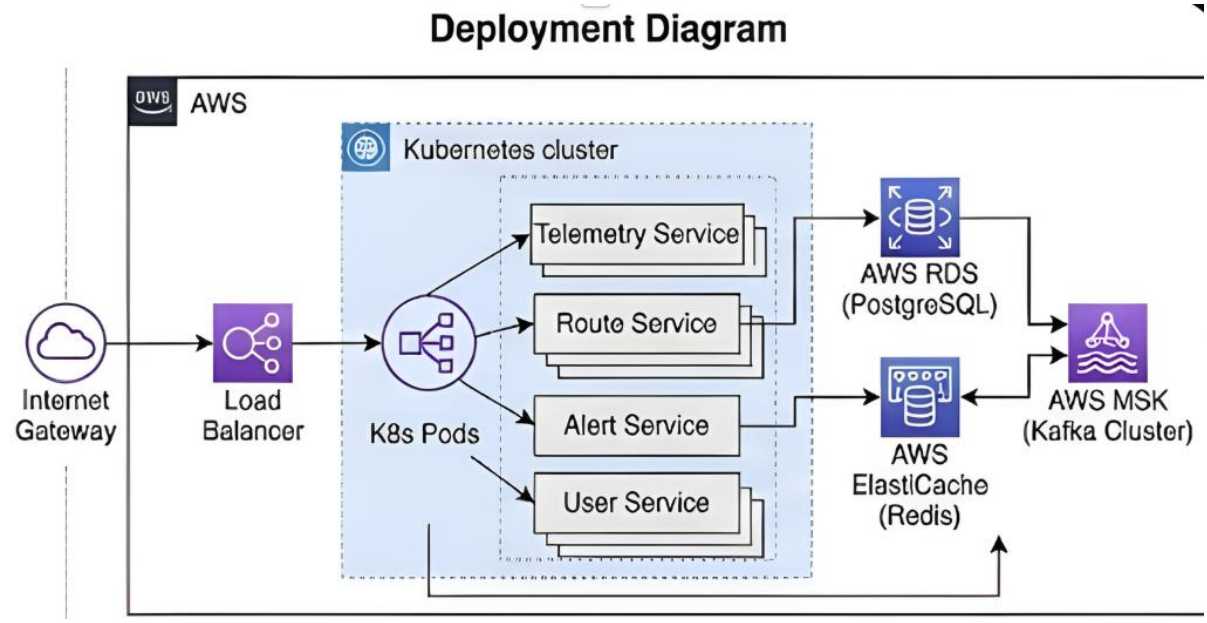
➞ Final Choice: Microservices + Event-Driven Architecture

Reason:
Microservices support modular growth, multi-tenant deployment, independent scaling, and real-time ingestion using Kafka.

2.2 Component Diagram



2.3 Deployment Diagram



---

3 — Design Patterns

3.1 Creational Patterns

Pattern	Purpose	Application
Factory Method	Create flexible ETA models	Different algorithms for bus vs metro
Singleton	Ensure only one shared instance	Redis and database connection pooling

---

3.2 Structural Patterns

Pattern	Purpose
Adapter	Converts inconsistent GPS vendor data formats into a unified schema
Facade	Simplifies complex internal service calls into one API endpoint

---

3.3 Behavioural Patterns

Pattern	Purpose
Strategy	Allows interchangeable ETA calculation algorithms
Observer	Push notifications based on telemetry events

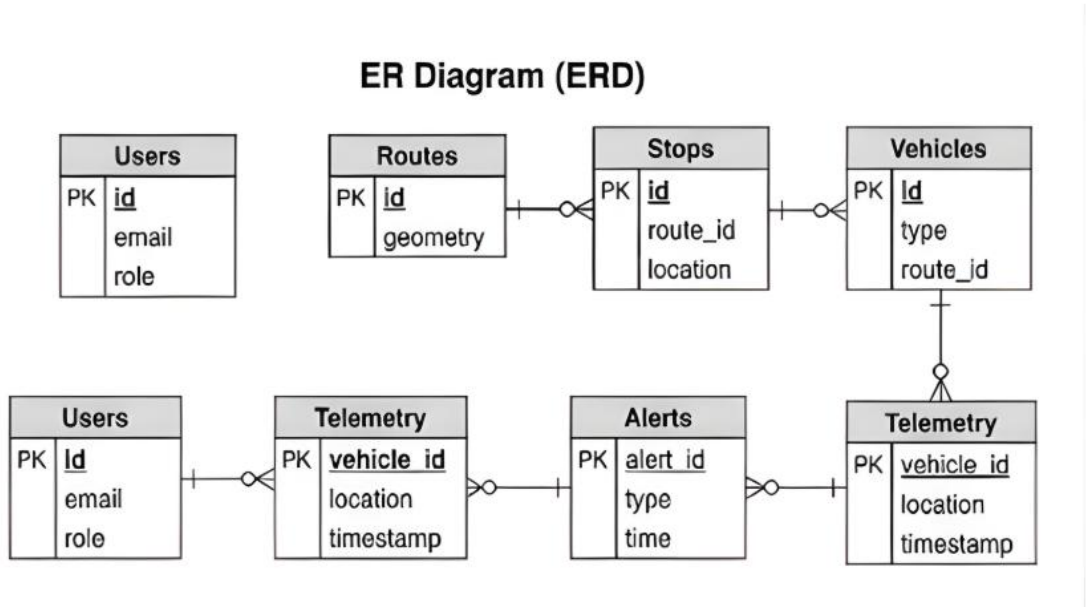
3.4 Anti-Patterns Avoided

- **God Object:** Logic distributed among services.
- **Hardcoding:** Configuration stored in DB and environment variables.
- **Spaghetti API calls:** Standard service discovery + messaging.

---

4 — Database Design

4.1 ER Diagram



4.2 Schema Tables

Table	Key Fields	Purpose
users	id, email, role	Authentication
routes	id, geometry	Static route path
stops	id, route_id, location	Stop geodata
vehicles	id, type, route_id	Fleet metadata
telemetry	vehicle_id, location, timestamp	Real-time tracking
alerts	alert_id, type, time	Notifications

4.3 Normalization vs Denormalization

- Master data (users, vehicles, routes) normalized to reduce redundancy.
- Real-time telemetry denormalized for fast read access and caching.

4.4 Indexing Strategy

- B-tree index for ID-based queries.
- Spatial index via **PostGIS** for route queries and nearest stop calculations.

4.5 Partitioning & Sharding

- Telemetry time-partitioned (daily/hourly tables).
- Multi-tenant sharding per city to avoid cross-region conflicts.

---

5 — Performance & Scale

- Redis caching for live status.
- CDN edge caching for route maps.
- Kubernetes auto-scaling on CPU and request metrics.
- Kafka prevents overload via consumer group backpressure.
- Asynchronous tasks for analytics and ETA modeling.

---

6 — Security & Reliability

- Threat modeling includes GPS spoofing, SQL injection, MITM.
- OAuth2 + JWT for user authentication.
- RBAC for admin/driver/user access.
- Automatic failover and blue-green deployment for service continuity.

Section 7 — API Specification

Endpoint	Method	Purpose
/routes/{city}	GET	Retrieve route list
/vehicle/{id}	GET	Live location
/vehicle/{id}/eta	GET	ETA estimation
/alerts	POST	Create alert

- Error codes include 400, 401, 404, 429, and 500.
- Idempotency key required for POST operations.
- API versioning format: /v1/...

---

8 — Observability

Observability ensures system transparency and operational health.

Feature	Tool
Logging	ELK (Elasticsearch + Logstash + Kibana)
Metrics	Prometheus
Tracing	Jaeger + OpenTelemetry
Runbook Events	On-call automation with PagerDuty

---

9 — Tech Stack Justification

Layer	Options Evaluated	Final Pick	Justification
Backend	NodeJS, Python, Go, Java	**Go**	Fast, memory-efficient
DB	MySQL, MongoDB, PostgreSQL	**PostGIS PostgreSQL**	Best for geo spatial queries
Cache	Redis, Memcached	**Redis**	<10ms cache reads
Broker	Kafka, RabbitMQ	**Kafka**	Fault tolerant high throughput
Cloud	AWS, Azure, GCP	**AWS**	Rich managed services

Conclusion

The City-Scale Transit Tracking and ETA Service delivers real-time visibility, reduced waiting times, smarter fleet management, and improved commuter satisfaction using modern distributed systems principles, reliable messaging, geospatial intelligence, and predictive modeling.
