Explanation Document for On-Demand Logistics Platform

1 Introduction

The on-demand logistics platform is designed to provide scalable, real-time transportation services, connecting users with a fleet of drivers for moving goods. With 50 million users and 100,000 drivers, the system is built to handle 10,000 concurrent requests per second while ensuring seamless coordination, booking, real-time tracking, and payment processing. This document outlines the major design decisions, trade-offs, and technical implementation details that enable the platform to achieve the required performance and scalability.

2 Major Design Decisions and Trade-offs

2.1 Scalability with Microservices Architecture

A microservices architecture was chosen to ensure scalability and maintainability. Each service (e.g., Booking, Driver Management, Payment) can scale independently, avoiding the bottlenecks of a monolithic system and improving maintainability.

Trade-off: Increased complexity in managing multiple services, including orchestration and communication between them.

2.2 Database Choice – MongoDB with Sharding

MongoDB was selected for its ability to handle semi-structured data and support geospatial indexing, which is critical for driver-user matching. The use of sharding ensures performance at scale by distributing data across multiple servers.

Trade-off: Managing shard keys and data partitioning adds complexity and requires careful design.

2.3 Real-Time GPS Tracking with Socket.io

Socket.io is used to facilitate real-time, low-latency communication between users and drivers, allowing users to monitor driver locations in real time.

Trade-off: Adds infrastructure complexity, particularly with millions of active connections, but significantly improves the user experience.

2.4 Workflow Management with Zeebe Workflow Engine

Zeebe is used to orchestrate processes such as bookings and driver assignments, ensuring robust management of task flows and reliability.

Trade-off: Adds deployment complexity but ensures consistent and reliable workflow execution, reducing errors.

2.5 Caching Layer with Redis

Redis is used as an in-memory cache to store frequently accessed data, reducing latency and offloading the primary database.

Trade-off: Requires careful memory management to prevent eviction of critical data during high load.

3 Handling High-Volume Traffic

3.1 Load Balancer – Nginx

Nginx is used to distribute incoming requests evenly across application servers, ensuring no single server becomes overloaded and enabling efficient handling of high traffic.

3.2 Container Orchestration with Kubernetes

Kubernetes allows for automatic scaling of containerized services, providing horizontal scalability. Auto-Scaling Groups further ensure the availability of service instances during peak traffic.

3.3 Message Broker – Apache Kafka

Kafka is used for handling asynchronous communication between services, such as booking updates and driver notifications, decoupling services and optimizing response times.

3.4 Sharded MongoDB

Sharding in MongoDB allows the database to handle large data volumes efficiently by distributing the load across multiple servers. Geospatial indexing is also used for proximity searches, optimizing driver-user matching.

3.5 Real-Time Data Handling

Socket.io and WebSockets ensure efficient handling of thousands of concurrent connections for real-time GPS tracking and booking updates.

4 Load Balancing and Distributed Data Handling

4.1 Nginx Load Balancer

Nginx distributes client requests across multiple application servers to ensure load distribution and mitigate DDoS attacks.

4.2 API Gateway – Kong

Kong acts as an API Gateway, routing requests from users and drivers to appropriate services, and implementing rate limiting and authentication to prevent misuse.

4.3 Kubernetes and Auto-Scaling

The platform uses Kubernetes to orchestrate containerized services, allowing for dynamic scaling based on real-time traffic demands.

4.4 Redis Caching

Redis is used for caching frequently accessed data, reducing the load on MongoDB and ensuring fast response times for real-time data requests.

5 Security Measures

HTTPS is enforced for secure communication, and JWT Tokens are used for authentication and authorization of users, drivers, and admins.