Project Documentation

Project Name: Scalable Logistics and Fleet Management Platform

Objective:

The objective of this project is to build a highly scalable and efficient logistics platform that can handle vehicle booking, real-time tracking, dynamic pricing, and automated driver assignments. It aims to streamline the logistics processes for both users and drivers, ensuring timely deliveries, optimized routes, and enhanced user experiences.

Introduction:

This platform is designed to offer a seamless logistics experience, integrating real-time tracking, dynamic pricing models (e.g., surge pricing during peak hours or bad weather), and automated driver allocation. Inspired by successful models like Uber India, this project uses a cutting-edge tech stack to handle high volumes of data and traffic, ensuring scalability and performance for large-scale usage.

The platform supports multiple vehicle types such as two-wheelers, mini trucks, and heavy trucks. It includes dashboards for users, drivers, and admins to manage bookings, fleets, payments, and analytics. Critical features such as live GPS tracking, driver proximity-based assignments, and efficient database management with sharding enable the system to handle real-time requests with low latency and high reliability.

Key Features:

Vehicle Booking: Users can book two-wheelers, mini trucks, or heavy trucks based on their needs.
Real-time GPS Tracking: Live tracking for both users and drivers, integrated with Google Maps API.
Dynamic Pricing: A flexible pricing model similar to Uber's, which adjusts rates based on vehicle type, location, and external factors like traffic and time of day.

Driver Assignment: Automated driver assignment based on proximity and availability, leveraging real-time algorithms to assign drivers within a 10km radius of the user's pinpointed pickup location.
Admin Dashboard: Admins can add drivers, update vehicle information, view performance metrics, and manage bookings.
Surge Pricing: Implemented during peak hours, bad weather, or in high-demand areas, to ensure fair pricing for users and extra incentives for drivers.
Role-based Dashboards: Separate dashboards for users, drivers, and admins to manage specific tasks efficiently.

Technology Stack:

- **Frontend:**

React.js
Firebase Authentication
Google Maps API
CSS (with custom styles)

- **Backend:**

Node.js
Express.js
MongoDB (with Sharding)
Redis (for caching)
JWT (for authentication)
bcrypt.js (for password hashing)

- **Additional Tools:**

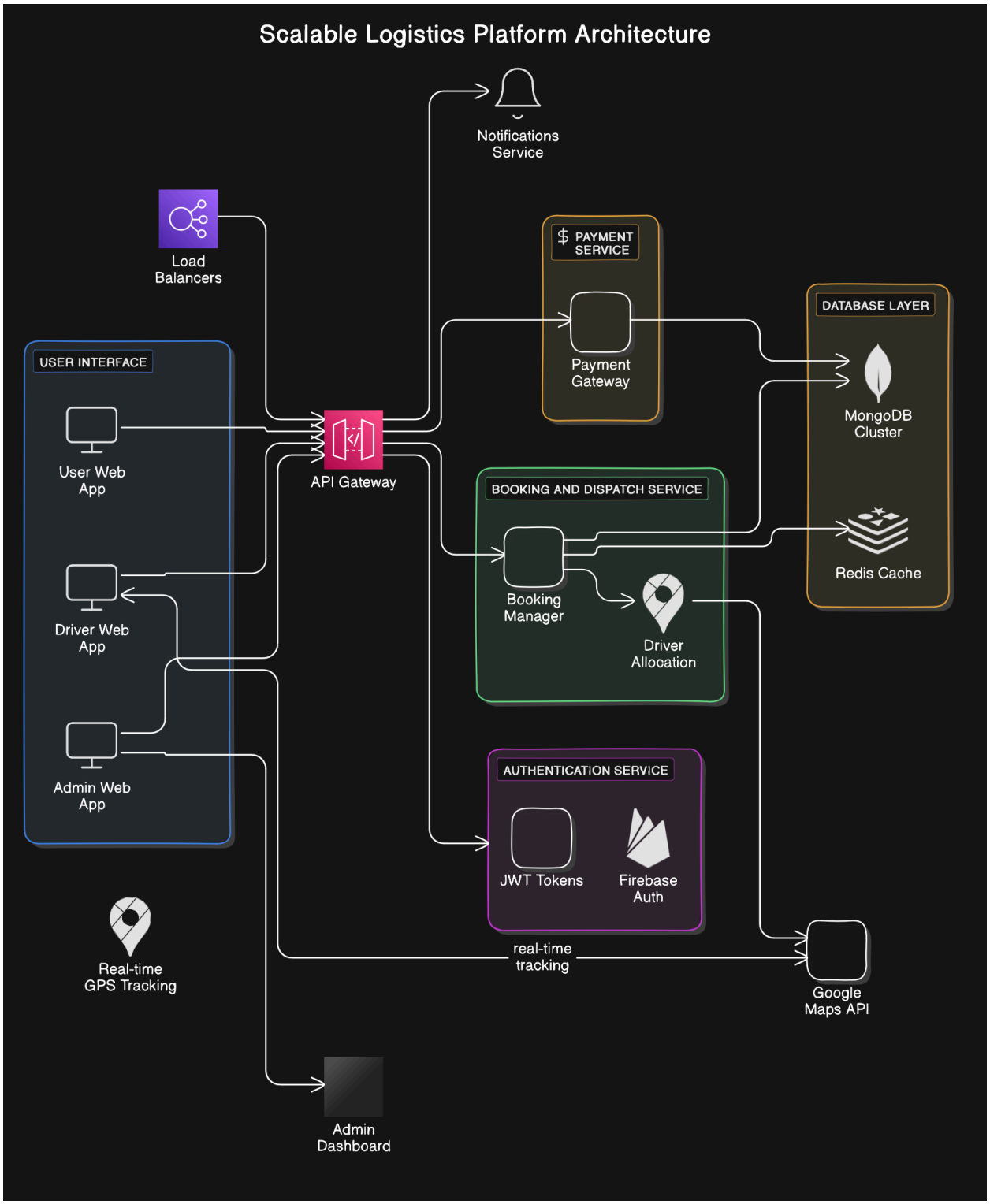
Docker
Nginx (for load balancing)
Nodemon (for development)
Mongoose (for MongoDB object modeling)

Why We Chose This Technology Stack:

1. Scalability: MongoDB with sharding ensures the platform scales horizontally as data grows.

2. Real-time Processing: Node.js and Redis are designed for handling asynchronous requests and real-time processing, crucial for live tracking.
3. Load Balancing: NGINX load balancing ensures high availability and smooth performance, even during peak loads.
4. Dynamic Pricing and Surge Algorithms: Drawing inspiration from Uber's pricing model, our surge pricing model dynamically adjusts fares based on real-time factors, such as demand, location, and traffic conditions. The system detects peak demand using API inputs (like weather or traffic) and automatically adjusts prices.

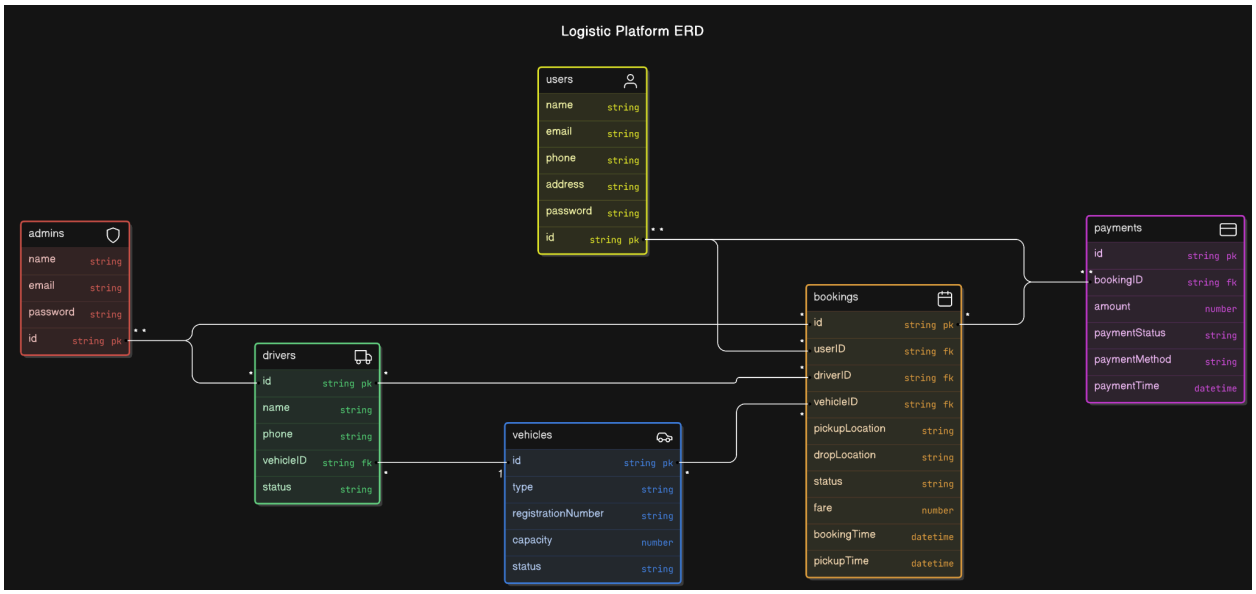
High-Level Architecture Diagram:



This diagram outlines the structure of the system and its key components, which include:

Load Balancers: Ensure even traffic distribution across the backend servers, preventing any single point of failure.
MongoDB (Sharded Clusters): The database is horizontally scaled using sharding to distribute data across clusters.
Redis Caching: Positioned between the database and the application, Redis stores frequently accessed data to reduce query times.
Real-time GPS Tracking Mechanisms: The system integrates with the Google Maps API, allowing users and drivers to receive continuous updates on vehicle locations.
Microservices: Each service (e.g., booking, tracking, payments) is decoupled from others, allowing for independent scaling.

Entity-Relationship (ER) Diagram:



The ER diagram defines relationships between key entities such as:

Users: Contains user details such as name, contact, and booking history.
Drivers: Includes driver details such as availability, assigned vehicle, and rating.
Bookings: Contains information about the booking details, including pickup and drop locations, fare, and vehicle assigned.

Vehicles: Stores details of the fleet (two-wheelers, mini trucks, trucks).
Payments: Handles payment records, including fare breakdowns and surge charges.

Task Sheet Questions

1. **Major design decisions and trade-offs, especially related to scalability and high-performance handling of real-time data**:
   We opted for MongoDB sharding to handle scalability. The trade-off here is increased complexity in data management, but the benefit is significant in terms of performance and scalability.
   Redis caching allows for quick data retrieval but increases memory usage. The trade-off between speed and resource consumption was carefully balanced to ensure optimal performance.
2. **How the system is capable of managing the specified high-volume traffic**:
   By using NGINX and horizontal scaling, the platform can handle large spikes in traffic. Each server is isolated, meaning if one fails, the others continue to operate seamlessly.

3. **How you've implemented load balancing and distributed data handling:**
   Load balancing ensures that even during periods of high demand (e.g., during festivals or high-traffic periods), the platform remains stable without dropping requests or experiencing downtime.
   For real-time data (such as GPS tracking and driver assignment), we rely on Node.js's event-driven architecture, which efficiently handles multiple concurrent requests.
   A trade-off here is that Node.js may face performance issues with heavy computational loads, but since the application is I/O heavy (GPS tracking and bookings), it performs well under these conditions.