DriveHive - Technical Design Document

CONTENTS

Introduction

This design document explains the structure and details of a ride-sharing platform where drivers can create rides, and customers can easily find and book them. The platform is designed to provide a smooth user experience, focusing on ride time, cost, and nearby or similar ride options. It includes secure user login and stores past ride data for each user.

This document is a guide for developers, stakeholders, and contributors, covering the platform's architecture, features, technology, and design choices.

System Overview

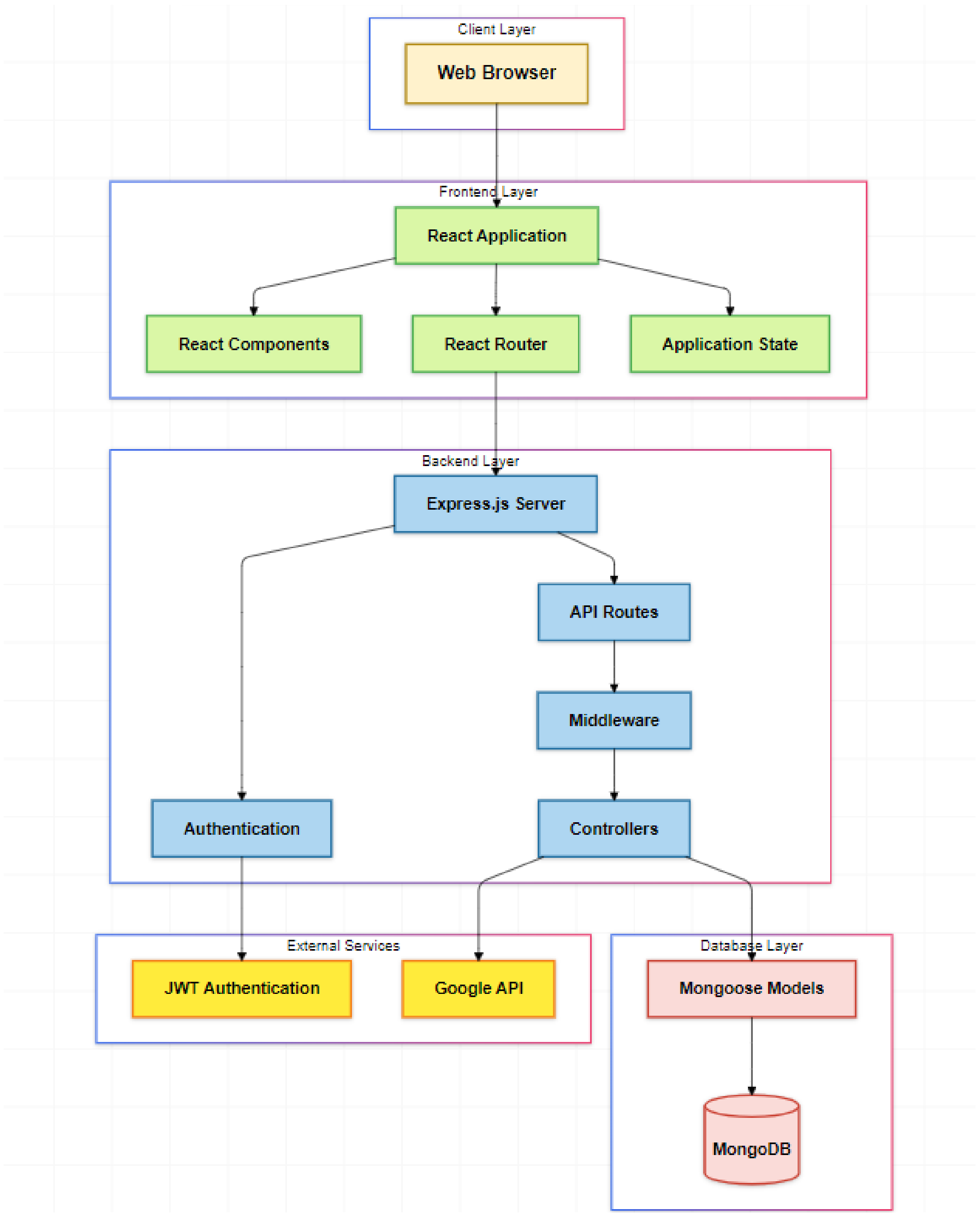
DriveHive is developed using the MERN stack (MongoDB, Express.js, React, Node.js) and follows modern web development standards. The platform is designed to ensure ease of maintenance, and a seamless, responsive user experience on all devices.

Architecture

High-Level Architecture

The system is designed using a three-tier architecture:

- **Frontend**: Built with React.js, it manages the client-side interface and user interactions.
- **Backend API**: Powered by Express.js and Node.js, it handles server-side logic, API endpoints, authentication, and business operations. Google APIs are integrated for maps and geographic data.
- **Database**: MongoDB is used to store user details, ride information, and related data.



Backend Design

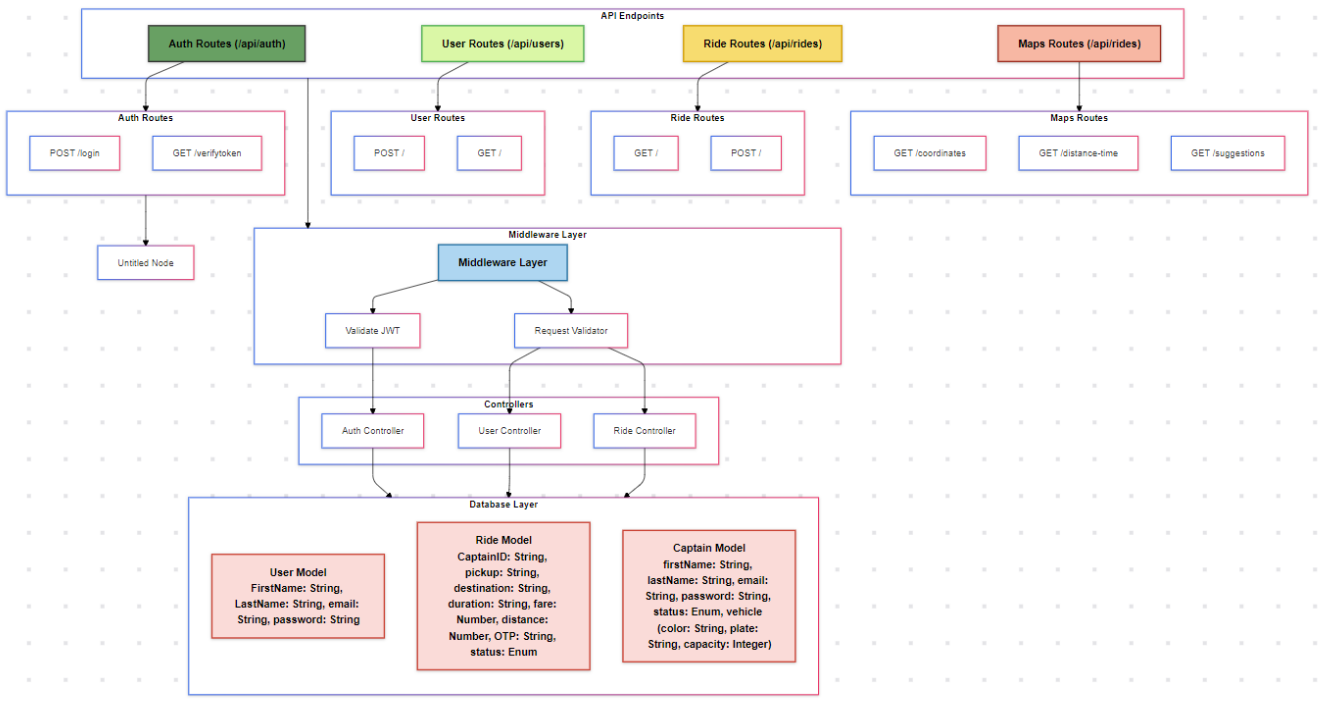
Technologies Used

- **Node.js**: JavaScript runtime for server-side development.
- **Express.js**: Framework for building APIs.
- **MongoDB**: NoSQL database for data storage.

- **Mongoose**: ODM (Object Data Modeling) library for MongoDB.
- **JWT**: JSON Web Tokens for secure authentication.
- **bcrypt**: Library for hashing passwords.

Project Structure

- **index.js**: Entry point of the server application.
- **middleware/**: Contains middleware functions, including authentication.
- **models/**: Mongoose schemas for User and Article models.
- **routes/**: Defines API endpoints for authentication, users, and rides.
- **services/**: Handles map-related tasks, including geographical coordinates, nearby locations, and distance and time calculations.



API Design

The backend exposes RESTful API endpoints categorized under:

- **Authentication (/api)**
  - POST/login: User login, verifies credentials, and issues a JWT token.
  - GET/verifytoken: Verifies the validity of the provided JWT token.
- **Users (/api/users)**
  - POST/: Registers a new user by creating a new account with provided details.
  - GET/: Retrieves the details of all registered users.

- **Rides (/api/rides)**
    - GET/: Retrieves a list of all available rides.
    - POST/: Creates a new ride offer with details such as destination, time, and price.
- **Maps(/api/rides)**
    - GET/coordinates: Retrieves nearby rides based on geographical coordinates.
    - GET/distance-time: Retrieves rides based on distance from the user’s location and time.
    - GET/suggestions: Provides suggested rides based on location, time, or other criteria.

Database Schema

USER MODEL (User.model.js)

- **FirstName** (String, required): User's first name.
- **LastName** (String, required): User’s last name.
- **email** (String, required, unique): User's email address.
- **password** (String, required): Hashed password.

Relations:

- A user can book multiple rides.

Notes:

- Passwords are hashed using bcrypt before being saved.

RIDES MODEL (Ride.model.js)

- **CaptainID** (String, required): Unique ID of the driver.
- **pickup** (String): Ride starting location.
- **destination** (String): Ride destination.
- **duration** (String): Estimated ride duration.
- **fare** (Number): Ride cost.
- **distance** (Number): Ride distance.
- **OTP** (String): One-time password for verification.
- **status** (Enum): Current ride status (e.g., “pending”, “accepted”, “completed”, “cancelled”).

Relations:

- Each ride is associated with a captain and user.

Notes:

- The **ride** field establishes a relationship between captain and users.

CAPTAINS MODEL (Ride.model.js)

Fields:

- **firstName** (String, required): Captain's first name.
- **lastName** (String, required): Captain's last name.
- **email** (String, required): Captain's email address, unique for login.
- **password** (String, required): Captain's account password.
- **status** (Enum): Captain's availability status (e.g., active, inactive).
- **vehicle** (Object):
    - **color** (String): Vehicle color.
    - **plate** (String): Vehicle license plate number.
    - **capacity** (Integer): Vehicle passenger capacity.
    - **vehicleType** (Enum): Type of the vehicle (e.g., sedan, SUV, van).

Relations:

- A captain can publish multiple rides.

Notes:

- Passwords are hashed using bcrypt before being saved.

Middlewares

Authentication Middleware (isAuthenticated.js)

- Validates JWT tokens in the Authorization header.
- Attaches authenticated user data to the request object.
- Secures routes that require user authentication.

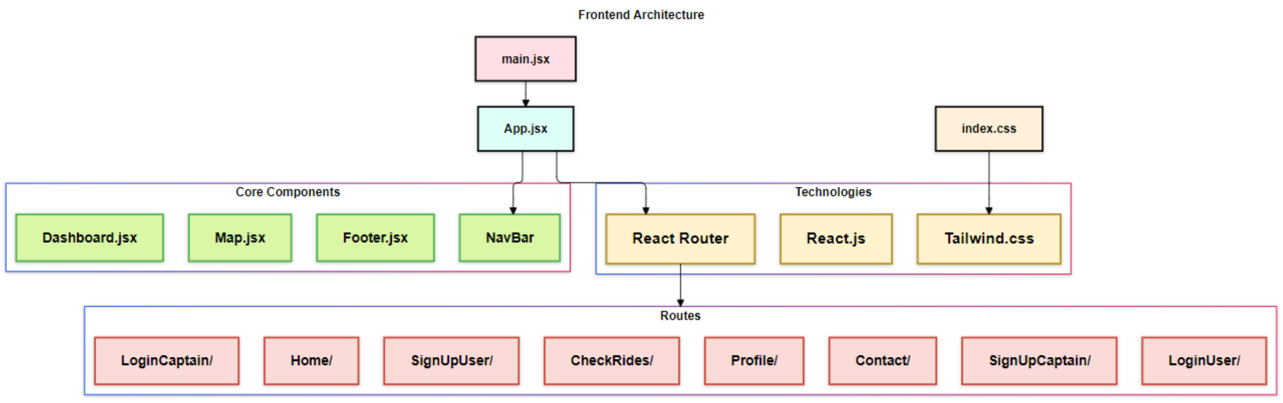
Frontend Design

Technologies Used

- **React.js:** JavaScript library for creating user interfaces.
- **React Router DOM:** Manages client-side routing.
- **Tailwind CSS:** Utility-first CSS framework for styling.
- **Vite:** Build tool for fast development.
- **React Spinner:** Used for loading indicators.

Project Structure

- **main.jsx:** Entry point of the React application.
- **App.jsx:** Main component that defines routes and their corresponding paths.
- **components/shared:** Reusable UI components used across the app.
- **components/:** Components specific to individual routes.
- **routes/:** Manages client-side routing.
- **Utils/constants.jsx:** Utility functions storing backend API links.
- **index.css:** Global CSS with Tailwind directives.
- **public/:** Directory for static assets.



Routing

Implemented using React Router:

- /: Home page.
- /signUser: User registration (signing up).
- /loginUser: User login.
- /signCaptain: Captain (driver) registration.
- /loginCaptain: Captain (driver) login.
- /profile: Fetch or update user or captain profile details.
- /about: Information about the platform.
- /contact: Contact details or inquiries.
- /findRides: Search for available rides based on criteria like location and time.

State Management

- **Local State**: Managed using useState and useEffect hooks.
- **Authentication State**:
    - Stored in localStorage.
    - Accessed via custom hooks or utility functions.
- **Data Fetching**:
    - Utilizes the fetch API.
    - Handles loading and error states.
- **Redux:**
    - Used Redux for state management, with all slices combined in store.js.
- **Slices:**
    - authSlice: Stores user data.
    - loadingSlice: Tracks the app's loading status.
    - locationSlice: Stores location data.

Key Components

NavBar:

- Displays navigation links.
- Shows different options based on authentication state.
- Includes a logout mechanism.

Map:

- Displays the map with locations.
- Dynamically updates latitude and longitude based on the user's searched locations.

Dashboard:

- Main UI of the app, where all the core features and functionalities are displayed.

Footer:

- Contains general information, links, or copyright details. Typically includes contact information, privacy policy, and terms of service.

Security Considerations

Authentication

- **JWT Tokens**:
    - Securely generated and signed with a secret key.
    - Stored in the client's localStorage.
- **Password Security**:
    - Passwords hashed using bcrypt before storing in the database.
    - Plain passwords are never stored or logged.

CORS Configuration

- **Access-Control Policies**:
    - Configured to allow requests from trusted origins.
    - Did authentication by passing the bearer token in the header.

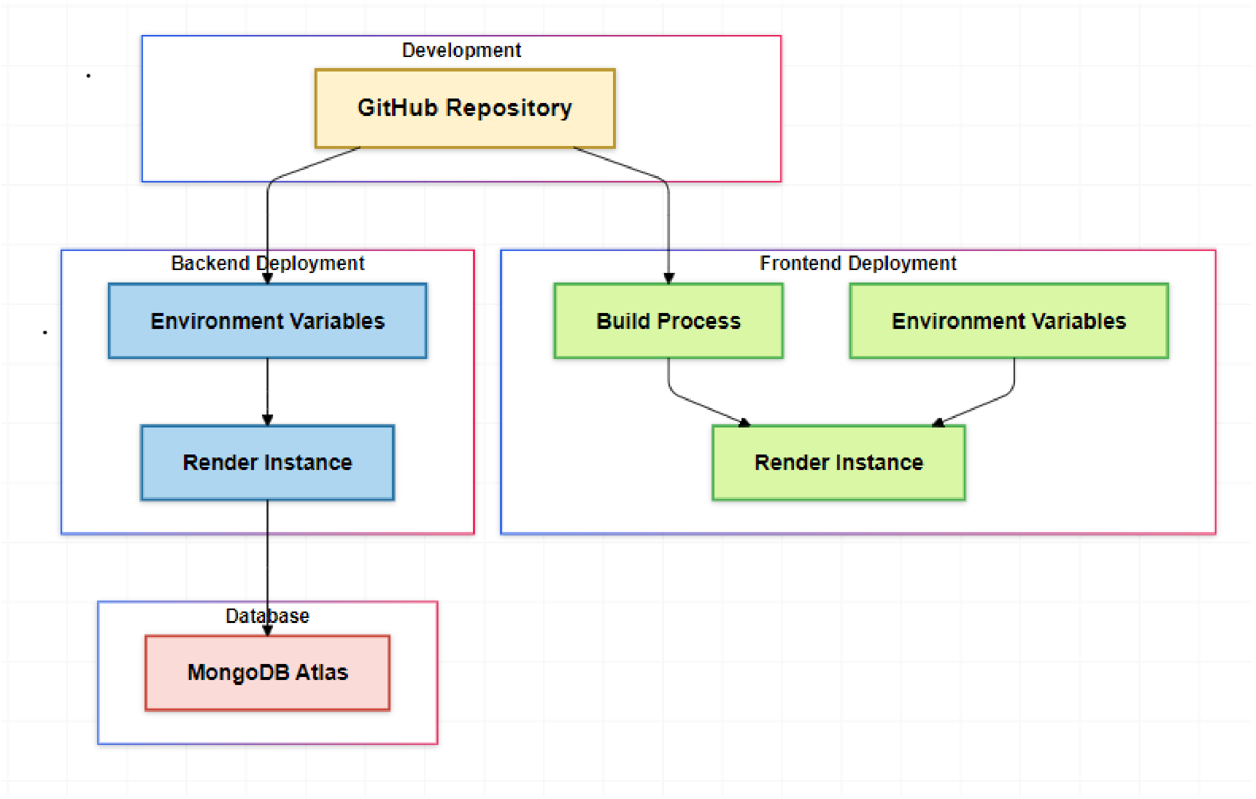
Deployment Plans

Environment Setup

- **Backend Environment Variables**:
    - PORT: Port number for the server.
    - DB_CONNECTION_STRING: MongoDB connection URI.
    - JWT_SECRET: Secret key for signing JWTs.
    - GOOGLE_API_KEY: for fetching maps and locations.
- **Frontend Environment Variables**:
    - USER_REGISTER_API: URL for user registration.
    - USER_LOGIN_API: URL for user login.
    - CAPTAIN_REGISTER_API: URL for captain (driver) registration.
    - CAPTAIN_LOGIN_API: URL for captain (driver) login.
    - LOCATION_API: URL for fetching location-based data.
    - GOOGLE_API_KEY: API key for Google services (maps, location, etc.).
    - GET_COORDINATES: URL for fetching geographical coordinates based on location.

Deployment Steps

1. **Backend Deployment**:
   - Hosted on render.
   - Ensure environment variables are securely set.
2. **Frontend Deployment**:
   - Build the React application using npm run build.
   - Host static files on services like Vercel.
3. **Database Hosting**:
   - Use managed MongoDB services like MongoDB Atlas.
   - Configure IP whitelisting and security measures.
4. **Continuous Deployment**:
   - Use GitHub Actions or other CI/CD tools.



Future Enhancements

Technical Improvements

- **Switch to TypeScript**:
    - Introduce TypeScript for type safety and better maintainability.
- **WebSockets**:
    - Use Socket.IO for real-time features like live comments or notifications.

Feature Enhancements

- **Chat Option:**
    - Enables real-time communication between users and captains.
- **Rating System:**
    - Allows users to rate captains based on their ride experience.
- **Notifications:**
    - Provides real-time notifications for user-captain interactions, such as ride updates or messages.
- **Analytics Dashboard:**
    - Offers users insights and statistics on their past rides, including ride frequency, cost, and ratings.

Conclusion

The DriveHive project is a full-stack web application developed using modern technologies to ensure scalability, security, and an excellent user experience. React.js is used for the frontend, Express.js and Node.js for the backend, and MongoDB for the database, creating a seamless platform for knowledge sharing.
The development process emphasized clear architecture with well-organized frontend and backend directories, modular components, and well-defined routes.

Working on DriveHive provided valuable insights into web development and reinforced the importance of user-centered design, making it a rewarding experience.