Full Stack Web Development Project Description

Programming Languages for Projects:

- **For Frontend;**
  - HTML:
  - EJS (Template Engine)
  - CSS
  - JavaScript
  - React
- **For Backend;**
  - NodeJS
  - ExpressJS
  - MongoDB

Tools Used for Projects:

- **IDE;**
  - VS Code
- **Code Managing / Hosting / Database;**
  - GitHub
  - Render
  - MongoDB Atlas

1. Project

Project Name: Uber Clone

- ## Project Overview:

The "Uber Clone" project aims to build a ride-hailing platform similar to Uber. It focuses on creating a reliable and easy-to-use backend system to manage users, captains, and their interactions. While the project is still in progress, the key features like user and captain registration, login, and profile management are already implemented. The Various Software and Programming Languages used for this Project are;

Software tools: VS code, Postman, MongoDB Compass, GitHub, Render, Atlas etc.

Programming Languages: NodeJS, ExpressJS, React, MongoDB (MERN).

- ## Features:

1. **User Management:**

Registration: Users can register by providing their first name, last name (optional), email, and password. The registration process validates input to ensure proper formatting (e.g., valid email addresses and minimum password length).

Login: Authenticated users can log in using their email and password. Upon successful authentication, a JWT token is generated for session management.

Profile Retrieval: Authenticated users can view their profile information, including their full name and email.

Logout: Users can securely log out, invalidating their session token.

2. **Captain Management:**

Registration: Captains can register by providing personal details (first and last name, email, and password) and vehicle information (color, plate number, capacity, and type).

Login: Similar to users, captains authenticate with email and password, receiving a JWT token for session handling.

Profile Retrieval: Captains can access their profile details, including personal and vehicle information.

Vehicle Information Management: Captains must provide mandatory details about their vehicle, ensuring consistency in vehicle types (e.g., car, motorcycle, or auto).

- ## Technical Details:

Backend Framework: Express.js is used for its simplicity, performance, and widespread support within the Node.js ecosystem.

Database: Though details of the database integration are yet to be fully described, the project includes a connection setup, likely to a NoSQL or relational database.

Environment Configuration: Environment variables are managed using dotenv, ensuring sensitive credentials and configurations remain secure.

- ## Backend API Description

Routing Structure: Routes are divided into:

- **/users:** Handling user-related functionalities.
- **/captains:** Managing captain-specific operations.

A. User:

i. /users/register Endpoint:

Description: Registers a new user by creating a user account with the provided information.

HTTP Method: POST

Request Body: The request body should be in JSON format and include the following fields:

fullname (object):

firstname (string, required): User's first name (minimum 3 characters).

lastname (string, optional): User's last name (minimum 3 characters).

email (string, required): User's email address (must be a valid email).

password (string, required): User's password (minimum 6 characters).

Example Response:

fullname (object).

firstname (string): User's first name (minimum 3 characters).

lastname (string): User's last name (minimum 3 characters).

email (string): User's email address (must be a valid email).

password (string): User's password (minimum 6 characters).

token (String): JWT Token

ii. /users/login Endpoint

Description: Authenticates a user using their email and password, returning a JWT token upon successful login.

HTTP Method: POST

Request Body: The request body should be in JSON format and include the following fields:

email (string, required): User's email address (must be a valid email).

password (string, required): User's password (minimum 6 characters).

Example Response:

user (object):

fullname (object).

firstname (string): User's first name (minimum 3 characters).

lastname (string): User's last name (minimum 3 characters).

email (string): User's email address (must be a valid email).

password (string): User's password (minimum 6 characters).

token (String): JWT Token

iii. /users/logout Endpoint

Description: Logout the current user and blacklist the token provided in cookie or headers.

HTTP Method: GET

Authentication: Requires a valid JWT token in the Authorization header or cookie:

user (object):

fullname (object).

firstname (string): User's first name (minimum 3 characters).

lastname (string): User's last name (minimum 3 characters).

email (string): User's email address (must be a valid email).

password (string): User's password (minimum 6 characters).

token (String): JWT Token## /captains/register Endpoint

B. Captains

i. /captains/register Endpoint

Description: Registers a new captain by creating a captain account with the provided information.

HTTP Method: POST

Request Body: The request body should be in JSON format and include the following fields:

fullname (object):

firstname (string, required): Captain's first name (minimum 3 characters).

lastname (string, optional): Captain's last name (minimum 3 characters).

email (string, required): Captain's email address (must be a valid email).

password (string, required): Captain's password (minimum 6 characters).

vehicle (object):

color (string, required): Vehicle color (minimum 3 characters).

plate (string, required): Vehicle plate number (minimum 3 characters).

capacity (number, required): Vehicle passenger capacity (minimum 1).

vehicleType (string, required): Type of vehicle (must be 'car', 'motorcycle', or 'auto').

Example Response:

captain (object):

fullname (object).

firstname (string): Captain's first name (minimum 3 characters).

lastname (string): Captain's last name (minimum 3 characters).

email (string): Captain's email address (must be a valid email).

password (string): Captain's password (minimum 6 characters).

vehicle (object):

color (string): Vehicle color.

plate (string): Vehicle plate number.

capacity (number): Vehicle passenger capacity.

vehicleType (string): Type of vehicle.

token (String): JWT Token

ii. /captains/login Endpoint

Description: Authenticates a captain using their email and password, returning a JWT token upon successful login.

HTTP Method: POST

Request Body: The request body should be in JSON format and include the following fields:

email (string, required): Captain's email address (must be a valid email).

password (string, required): Captain's password (minimum 6 characters).

Example Response:

captain (object):

fullname (object).

firstname (string): Captain's first name (minimum 3 characters).

lastname (string): Captain's last name (minimum 3 characters).

email (string): Captain's email address (must be a valid email).

password (string): Captain's password (minimum 6 characters).

vehicle (object):

color (string): Vehicle color.

plate (string): Vehicle plate number.

capacity (number): Vehicle passenger capacity.

vehicleType (string): Type of vehicle.

token (String): JWT Token

iii. /captains/logout Endpoint

Description: Logout the current captain and blacklist the token provided in cookie or headers.

HTTP Method: GET

Authentication: Requires a valid JWT token in the Authorization header or cookie.

Example Response: message (string): Logout successfully.

- ## Folder Strcuture and Files Details:

The Uber Directory has 2 Folders and 1 File in it, namely;

i. Backend (Folder)
ii. Fronted (Folder)
iii. .gitignore (File)

Backend: It contains the Backend logic including the routes, database interaction, etc. Its has various files and folder i.e.;

- **controllers (Folder):** has two files:
  - **captain.controller.js:** The captain.controller.js file is part of the backend logic in the "Uber Clone" project. It manages operations related to captains, such as registering, logging in, retrieving profiles, and logging out. Here's a breakdown of the file's purpose and functionality:

    Purpose:

    This controller handles HTTP requests related to captains, ensuring proper validation, interaction with the database, and appropriate responses to the client. It acts as a bridge between the HTTP endpoints and the underlying services/models.

  - **user.controller.js:** This file handles operations related to users (riders). It provides endpoints for user registration, login, profile retrieval, and logout.
  - **Purpose:** Authenticates a user and provides a session token.

- **db (Folder):** This has db.js file for connecting database connectivity with localHost at the time of development and with Atlas Database String at the time of production.
- **middlewares (Folder):** This has auth.middleware.js file that provides middleware functions to ensure the security of the "Uber Clone" application by verifying user or captain authentication status. These functions act as a gateway, ensuring that only authorized individuals can access specific routes or resources.
- **models (Folder):** It contains 3 files namely;
  - **blacklistToken.model.js:** defines a Mongoose schema for the BlacklistToken model, which is used to manage tokens that have been invalidated. These tokens are typically stored after a user or captain logs out or when a token needs to be revoked for security reasons.
  - **captain.model.js:** defines the Captain schema and model using Mongoose. This model is used to interact with the database, storing and managing data related to captains in the Uber Clone application. It incorporates features such as password hashing, token generation, and validation rules to ensure robust data integrity and security.
  - **user.model.js:** defines the User schema and model using Mongoose. This model is responsible for handling all user-related data and interactions with the MongoDB database. It includes essential fields, methods for authentication, and data validation rules to ensure security and reliability.

- **node_modules:** stores external dependencies of the project. Whenever we install some external package through “npm” or “yarn” in a project locally, it gets stored into the node_modules folder located at the root of the project directory by default.

- **routes:** This Folder has all the routes of the project. It has 2 files namely;

- **captain.routes.js:** contains the routes for Captain's Log-in (/register, /login, /profile, /logout etc).
- **user.routes.js:** contains the routes for User's Log-in (/register, /login, /profile, /logout etc).

- **services:** acts as a layer between the controllers and the database models. It encapsulates business logic and reusable functionality, keeping controllers clean and focused on handling HTTP requests and responses.
- **app.js (File):** is responsible for setting up and configuring the server application. It acts as the entry point where middleware, routes, and other key functionalities are initialized.
- **server.js (File):** is responsible for starting the HTTP server and binding it to a specified port. It is typically used to initialize the server by leveraging the configurations and middleware set up in the app.js file.
- **package-lock.json:** is a file that's automatically generated in Node.js projects when you run the npm install command. It locks down the exact versions of all packages and their dependencies in your project. This ensures that the same versions are installed each time, regardless of intermediate dependency updates.
- **package.json:** is a metadata file in a Node.js project that stores information about the project such as Dependencies, name and version, Description, keywords, files etc.

Frontend: It contains the Frontend (React's) Components for User Interactions. The React is imported or created using the Vite app by using the following cmds;

- npm create vite@latest my-app --template react