User Authentication API

This project is a backend implementation of user authentication with features including user signup, JWT-based authentication, and protected routes. It uses Node.js, Express, and MongoDB.

Features

1. **User Signup**
    - Validates user input fields (name, email, phone number, password).
    - Checks for existing users by email or phone number.
    - Hashes passwords before saving to the database.
    - Generates a JWT token upon successful signup.
2. **Captain Signup**
    - Validates captain-specific fields like driving license details and vehicle information.
    - Ensures unique email, phone number, and license details.
    - Hashes passwords before saving to the database.
    - Generates a JWT token upon successful signup.
3. **JWT Authentication**
    - Protects sensitive routes by verifying JWT tokens.
    - Decodes tokens to fetch user information for authorization.
4. **Validation Middleware**
    - Validates user and captain inputs using `express-validator`.
    - Provides detailed error messages for invalid inputs.

Technologies Used

- **Node.js**: Runtime environment for building server-side applications.
- **Express.js**: Web framework for creating RESTful APIs.
- **MongoDB**: NoSQL database for storing user and captain data.
- **Mongoose**: ODM for MongoDB, used for schema modeling and data validation.
- **JSON Web Tokens (JWT)**: Securely authenticates users by issuing and verifying tokens.
- **express-validator**: Middleware for validating and sanitizing request data.

Installation

1. Clone the repository:

```
git clone <repository_url>
```

2. Navigate to the project directory:

   ```
   cd user-authentication-api
   ```

3. Install dependencies:

   ```
   npm install
   ```

4. Create a `.env` file in the root directory and configure the following environment variables:

   ```
   MONGO_URI=<your_mongodb_connection_string>
   JWT_SECRET=<your_jwt_secret>
   PORT=5000
   ```

5. Start the server:

   ```
   npm start
   ```

Endpoints

1. **User Signup**

POST `/api/users/signup`

Registers a new user. Validates inputs and generates a JWT token upon success.

Request Body:

```
{
  "name": "John Doe",
  "email": "johndoe@example.com",
  "phoneNumber": "1234567890",
  "password": "password123"
}
```

Response:

```
{
    "_id": "67892ba437902adc7e3d260c",
    "name": "John Doe",
    "email": "johndoe@example.com",
```

```
    "phoneNumber": "1234567890",
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3ODkyYmE0Mzc5MDJhZGM3ZTNkMjYwYyIs
ImlhdCI6MTczNzA0Mjg1MywiZXhwIjoxNzM5NjM0ODUzfQ.f_Molirs_KFle2HpJrWC2r9hpm4Z-
VpzaFxUmRfkJiE"
}
```

Errors:

- `400`: Validation errors or existing user.
- `500`: Server error.

2. **Captain Signup**

POST `/api/captains/signup`

Registers a new captain. Validates inputs, including driving license and vehicle details, and generates a JWT token upon success.

Request Body:

```
{
    "name": "Alice Navigator",
    "email": "alice.navigator@example.com",
    "phoneNumber": "+198765432109",
    "password": "securepassword123",
    "drivingLicense": {
        "number": "DL9876543210",
        "expiryDate": "2035-07-15T00:00:00.000Z"
    },
    "vehicle": {
        "make": "Ford",
        "model": "Mustang",
        "year": 2023,
        "color": "Rapid Red",
        "licensePlate": "FORD999"
    }
}
```

Response:

```
{
    "_id": "67893ce91511e6aef3d0386d",
    "name": "Alice Navigator",
    "email": "alice.navigator@example.com",
    "phoneNumber": "+198765432109",
    "drivingLicense": {
        "number": "DL9876543210",
        "expiryDate": "2035-07-15T00:00:00.000Z"
    },
    "vehicle": {
        "make": "Ford",
        "model": "Mustang",
        "year": 2023,
        "color": "Rapid Red",
        "licensePlate": "FORD999"
    },
    "isVerified": false,
    "isActive": false,
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3ODkzY2U5MTUxMWU2YWVmM2QwMzg2ZCIs
ImlhdCI6MTczNzA0NzI3MywiZXhwIjoxNzM5NjM5MjczfQ.KNrHj4vsJFWD9Ebw10YHD-
yRidMhwoF5gBiRrOolnUs"
}
```

Errors:

- `400`: Validation errors or existing captain.
- `500`: Server error.

Validation Rules

1. **Name**: Required, minimum 3 characters.
2. **Email**: Required, must be a valid email format.
3. **Phone Number**: Required, must be a valid phone number.
4. **Password**: Required, minimum 8 characters.
5. **Driving License Number**: Required, unique.
6. **Driving License Expiry Date**: Required, valid date.
7. **Vehicle Details**: Make, model, year, color, and license plate are all required.

How It Works

1. A user sends a signup request to the `/api/users/signup` endpoint.
2. The request is validated by `express-validator` middleware.
3. The server checks for existing users in the database.
4. If validation passes, a new user or captain is created, and a JWT token is issued.
5. The `protect` middleware ensures only authenticated users access protected routes.

Location Route and Controller

The location route and controller handle requests related to location suggestions.

Location Route

The location route is defined in `server/routes/location.route.js`. It uses the `getSuggestions` controller to fetch location suggestions from the Nominatim OpenStreetMap API.

Location Controller

The location controller is defined in `server/controllers/location.controller.js`. It exports the `getSuggestions` function, which handles GET requests to the `/api/locations/suggestions` endpoint.

License

This project is licensed under the MIT License. Feel free to use, modify, and distribute as needed.