Online Cab Booking System API

This is the backend system of an online cab booking system. It features user/captain signup, JWT authentication, and protected routes. The system is built with Node.js, Express, and MongoDB.

Table of Contents

Features

- User Authentication (signup, login, profile management)
- Captain Authentication (signup, login, profile management)
- JWT-based Authentication
- Location Services Integration
- Input Validation Middleware
- Protected Routes
- OpenStreetMap Integration

Technologies Used

- **Node.js** - Runtime environment
- **Express.js** - Web framework
- **MongoDB** - Database
- **Mongoose** - ODM for MongoDB
- **JWT** - Authentication tokens
- **express-validator** - Input validation
- **React Leaflet** - Map integration
- **OpenStreetMap API** - Location services

Installation

1. Clone the repository:

```
git clone <repository_url>
```

2. Navigate to project directory:

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

3. Install dependencies:

```
npm install
```

4. Configure environment variables:

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

5. Start the server:

```
npm start
```

API Documentation

User Routes

1. User Signup

- **POST** `/api/users/signup`
- **Description**: Register a new user
- **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..."
}
```

2. User Login

- **POST** `/api/users/login`
- **Description**: Authenticate existing user
- **Request Body**:

```
{
  "email": "johndoe@example.com",
  "password": "password123"
}
```

- **Response**:

```
{
  "_id": "678a59ce816da4cee492e4c7",
  "message": "User logged in successfully",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

3. User Profile

- **GET** `/api/users/account`
- **Description**: Fetch user profile
- **Headers**: `Authorization: Bearer <token>`
- **Response**:

```
{
  "_id": "678a930536748191acff0891",
  "name": "John Doe",
  "email": "johndoe@example.com",
  "phoneNumber": "1234567890"
}
```

Captain Routes

1. Captain Signup

- **POST** `/api/captains/signup`
- **Description**: Register a new captain
- **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"
  }
}
```

2. Captain Login

- **POST** `/api/captains/login`
- **Description**: Authenticate existing captain
- **Request Body**:

```
{
  "email": "alice.navigator@example.com",
  "password": "securepassword123"
}
```

Location Routes

1. Get Location Suggestions

- **GET** `/api/locations/suggestions`
- **Query Parameters**: `query` (location search term)
- **Description**: Fetch location suggestions from OpenStreetMap

2. Get Coordinates

- **GET** `/api/locations/get-coordinates`
- **Query Parameters**: `query` (location name)
- **Description**: Fetch coordinates for a location
- **Response**:

```
{
  "lat": "25.0057449",
```

```
  "lon": "88.1398483"
}
```


Data Validation

User Validation Rules

- Name: Required, minimum 3 characters
- Email: Required, valid email format
- Phone Number: Required, valid format
- Password: Required, minimum 8 characters

Captain Validation Rules

- All User validation rules plus:
- Driving License Number: Required, unique
- Driving License Expiry: Required, valid date
- Vehicle Details: Make, model, year, color, license plate required

User Authentication API

[Previous sections remain the same up to Location Routes]

Location Routes and Services

1. Get Location Suggestions

- **GET** `/api/locations/suggestions`
- **Description**: Fetch location suggestions from OpenStreetMap Nominatim API
- **Query Parameters**:
  - `query` (required): Location search term (e.g., "alipurduar")
- **Response**:


```
[
  {
    "place_id": 222689300,
    "licence": "Data © OpenStreetMap contributors, ODbL 1.0",
    "osm_type": "node",
    "osm_id": 568606431,
    "lat": "26.4851573",
    "lon": "89.5246926",
    "class": "place",
    "type": "city",
    "place_rank": 16,
    "importance": 0.4282188878241305,
    "addresstype": "city",
    "name": "Alipurduar",
    "display_name": "Alipurduar, Alipurduar - I, West Bengal, 736121, India",
```

```
    "address": {
      "city": "Alipurduar",
      "county": "Alipurduar - I",
      "state": "West Bengal",
      "postcode": "736121",
      "country": "India",
      "country_code": "in"
    },
    "boundingbox": [
      "26.3251573",
      "26.6451573",
      "89.3646926",
      "89.6846926"
    ]
  }
]
```


2. Get Coordinates

- **GET** `/api/locations/get-coordinates`
- **Description**: Fetch specific coordinates for a location
- **Query Parameters**:
  - `query` (required): Location name (e.g., "malda")
- **Response**:


```
{
  "lat": "25.0057449",
  "lon": "88.1398483"
}
```


- **Error Response** (400):


```
{
  "error": "Location not found"
}
```


Map Integration with React Leaflet

1. Installation and Setup

```
# Install required packages
npm install react-leaflet leaflet @types/leaflet
```

Add Leaflet CSS to your index.html:

```
<link
  rel="stylesheet"
  href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css"
/>
```

2. Basic Map Component

```
import React from 'react';
import { MapContainer, TileLayer, Marker, Popup } from 'react-leaflet';
import 'leaflet/dist/leaflet.css';
import { Icon } from 'leaflet';

// Custom marker icon setup
const customIcon = new Icon({
  iconUrl: '/marker-icon.png',
  iconSize: [25, 41],
  iconAnchor: [12, 41]
});

const Map = ({ position, zoom = 13 }) => {
  return (
    <MapContainer
      center={position}
      zoom={zoom}
      style={{ height: '400px', width: '100%' }}
    >
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>'
      />
      <Marker position={position} icon={customIcon}>
        <Popup>
          Latitude: {position.lat}<br />
          Longitude: {position.lng}
        </Popup>
      </Marker>
    </MapContainer>
  );
};

export default Map;
```

Location Controller Implementation

```
// controllers/location.controller.js

const axios = require('axios');
```

```
// Get location suggestions
exports.getSuggestions = async (req, res) => {
  try {
    const { query } = req.query;

    if (!query) {
      return res.status(400).json({ error: 'Search query is required' });
    }

    const response = await axios.get(
      `https://nominatim.openstreetmap.org/search`,
      {
        params: {
          q: query,
          format: 'json',
          addressdetails: 1,
          limit: 5
        },
        headers: {
          'User-Agent': 'YourAppName/1.0'
        }
      }
    );

    res.json(response.data);
  } catch (error) {
    console.error('Location suggestion error:', error);
    res.status(500).json({ error: 'Failed to fetch location suggestions' });
  }
};

// Get coordinates for a location
exports.getCoordinates = async (req, res) => {
  try {
    const { query } = req.query;

    if (!query) {
      return res.status(400).json({ error: 'Location query is required' });
    }

    const response = await axios.get(
      `https://nominatim.openstreetmap.org/search`,
      {
        params: {
          q: query,
          format: 'json',
          limit: 1
        },
        headers: {
          'User-Agent': 'YourAppName/1.0'
        }
      }
    );
```

```
    if (response.data.length === 0) {
      return res.status(400).json({ error: 'Location not found' });
    }

    const { lat, lon } = response.data[0];
    res.json({ lat, lon });
  } catch (error) {
    console.error('Coordinates fetch error:', error);
    res.status(500).json({ error: 'Failed to fetch coordinates' });
  }
};
```

Location Route Implementation

```
// routes/location.routes.js

const express = require('express');
const router = express.Router();
const {
  getSuggestions,
  getCoordinates
} = require('../controllers/location.controller');
const { protect } = require('../middleware/auth.middleware');

router.get('/suggestions', protect, getSuggestions);
router.get('/get-coordinates', protect, getCoordinates);

module.exports = router;
```

Error Handling Middleware

```
// middleware/error.middleware.js

const errorHandler = (err, req, res, next) => {
  console.error(err.stack);

  if (err.name === 'ValidationError') {
    return res.status(400).json({
      error: 'Validation Error',
      details: err.message
    });
  }

  if (err.name === 'AxiosError') {
    return res.status(500).json({
      error: 'External API Error',
      details: 'Failed to fetch location data'
    });
```

```
  }

  res.status(500).json({
    error: 'Internal Server Error',
    details: process.env.NODE_ENV === 'development' ? err.message : undefined
  });
};

module.exports = errorHandler;
```

[Previous sections remain the same]

Real-time Communication

This project implements real-time communication between users and captains using Socket.IO, enabling instant updates and location tracking.

Socket.IO Integration

Installation

```
# Server-side
npm install socket.io
# Client-side
npm install socket.io-client
```

Basic Server Setup

```
// server.js
const express = require('express');
const http = require('http');
const socketIO = require('socket.io');

const app = express();
const server = http.createServer(app);
const io = socketIO(server);

io.on('connection', (socket) => {
  console.log('New client connected');

  // Handle ride request
  socket.on('ride_request', (data) => {
    // Emit to available captains
    io.emit('new_ride_request', data);
  });

  // Handle location updates
  socket.on('location_update', (data) => {
```

```
    io.emit('captain_location', data);
  });

  socket.on('disconnect', () => {
    console.log('Client disconnected');
  });
});

server.listen(5000, () => {
  console.log('Server running with Socket.IO on port 5000');
});
```

Basic Client Setup

```
// client.js
import io from 'socket.io-client';

const socket = io('http://localhost:5000');

// Connect to Socket.IO server
socket.on('connect', () => {
  console.log('Connected to server');
});
```

Features

1. **Real-time Updates**
   - Instant ride request notifications
   - Live location tracking
   - Status updates for both users and captains
2. **Event Types**
   - `ride_request`: When user requests a ride
   - `new_ride_request`: Notifies captains of new requests
   - `location_update`: Real-time captain location updates
   - `ride_status`: Updates on ride progress
3. **Use Cases**
   - Ride matching between users and captains
   - Live tracking during rides
   - Instant notifications for both parties
   - Real-time chat functionality (planned)

Implementation Overview

1. **Server-side Events**

- Connection management
- Event broadcasting
- Room management for private communications

2. **Client-side Integration**

- Event listeners for updates
- Real-time map updates
- Status synchronization

3. **Data Flow**

- User requests ride → Server → Available captains
- Captain accepts → Server → User notified
- Location updates → All relevant parties

Planned Features

1. **Chat System**

- Direct messaging between user and captain
- Support for multimedia messages

2. **Enhanced Tracking**

- Improved location accuracy
- ETA calculations
- Route visualization

3. **Status Updates**

- Ride progress notifications
- Payment status
- Rating system

[Rest of the documentation remains the same]

Project Structure

```
.
├── controllers/
│   ├── user.controller.js
│   ├── captain.controller.js
│   └── location.controller.js
├── database/
│   └── db.js
├── middleware/
│   ├── auth.middleware.js
├── models/
│   ├── user.model.js
│   └── captain.model.js
├── routes/
```

```
│   ├── user.routes.js
│   ├── captain.routes.js
│   └── location.routes.js
├── services/
│   ├── hashPassword.js
│   ├── JWToken.js
│   └── comparePassword.js
├── .env
├── package.json
└── server.js
```

How It Works

1. Request Handling:
    - Incoming requests are routed to appropriate controllers
    - Middleware validates inputs and authenticates tokens
    - Controllers process requests and interact with database
2. Authentication Flow:
    - User/Captain submits credentials
    - Server validates input
    - On success, JWT token is generated
    - Token is required for protected routes
3. Location Services:
    - Integrates with OpenStreetMap API
    - Provides location suggestions and coordinates
    - Supports map visualization with React Leaflet
4. Real-time Communication:
    - Socket.io is used to establish real-time communication between users and drivers
    - Enables instant updates and notifications for ride requests, status, and location sharing

License