



MovInSync Assignment

Problem Statement: Building a Ride-Sharing Platform
with Geofence Technology

Submitted by:

Ayush Singh

IIIT Sonapat

Email: ayushmail2u2@gmail.com

Index

Section	Page no.
Problem Statement	3
Abstract	4
Tech Stack	4
Project Structure	5
Scope of Project	5
Use case diagram	6
Class diagram	6
Sequential diagram	7
Activity diagram	8
Implementations	8
GitHub repository	29
Demo Video	29

List of Abbreviation

Abbreviation	Full-Form
API	Application Programming Interface
UI	User Interface
auth	Authentication
JWT	JSON Web Token
Latlong/Lat-long	Latitude and Longitude

Problem Statement

Building a Ride-Sharing Platform with User and Admin Functionalities:

- In this project, three distinct user roles have been established: Traveller, Admin, and the Traveller companion.

Traveller:

- Travellers should have the ability to share ride details (e.g., Trip ID, Driver Name, Driver Phone Number, cab number, etc.) through WhatsApp or SMS if the trip is in progress and expire the link after the Trip is complete.
- Users can review the audit trail of the rides they have shared.

Traveller companion:

- Track the ride of the Traveller.
- Should get a notification when the trip is complete.
- Should get the nearby notification when the cab hits the geofence of the Traveller drop.
- Share the feedback of the experience with Admin.

Admin:

- Admins have access to view all the rides shared by users.
- The Admin should be able to access the overall experience feedback of the users.

Constraints:

- Implement robust authentication measures to ensure the security of user accounts.
- Propose scalable strategies to accommodate an expanding user base.

Abstract

This project aims to develop a ride-sharing platform with distinct functionalities for three user roles: Traveller, Traveller Companion, and Admin.

The Traveller can share trip details (such as Trip ID, Driver Name, and cab information) via WhatsApp or SMS while the trip is active, with a secure, expiring link post-trip completion. They also have access to an audit trail of shared rides.

The Traveller Companion can monitor the Traveller's ride in real time, receive notifications upon trip completion and when the cab enters a predefined geofence near the drop location, and provide feedback to the Admin.

The Admin role allows for oversight of all shared rides and access to user experience feedback for continuous improvement. Key constraints focus on ensuring robust authentication for user security and implementing scalable strategies to accommodate a growing user base.

Tech Stack

Frontend:

- Language: JavaScript (Support OOPS), CSS, HTML
- Framework and Tools: ReactJS, Email JS, react libraries like Lucid-react, React-Toast, VS Code, GitHub

Backend:

- Node JS, Express, Nodemon, Mongoose
- Database: MongoDB
- API: Radar.io, Maplibre-gl

Hosting: Localhost

- Server: Port:8000
- API calling: localhost:5000

Project Structure

Rideshare-1

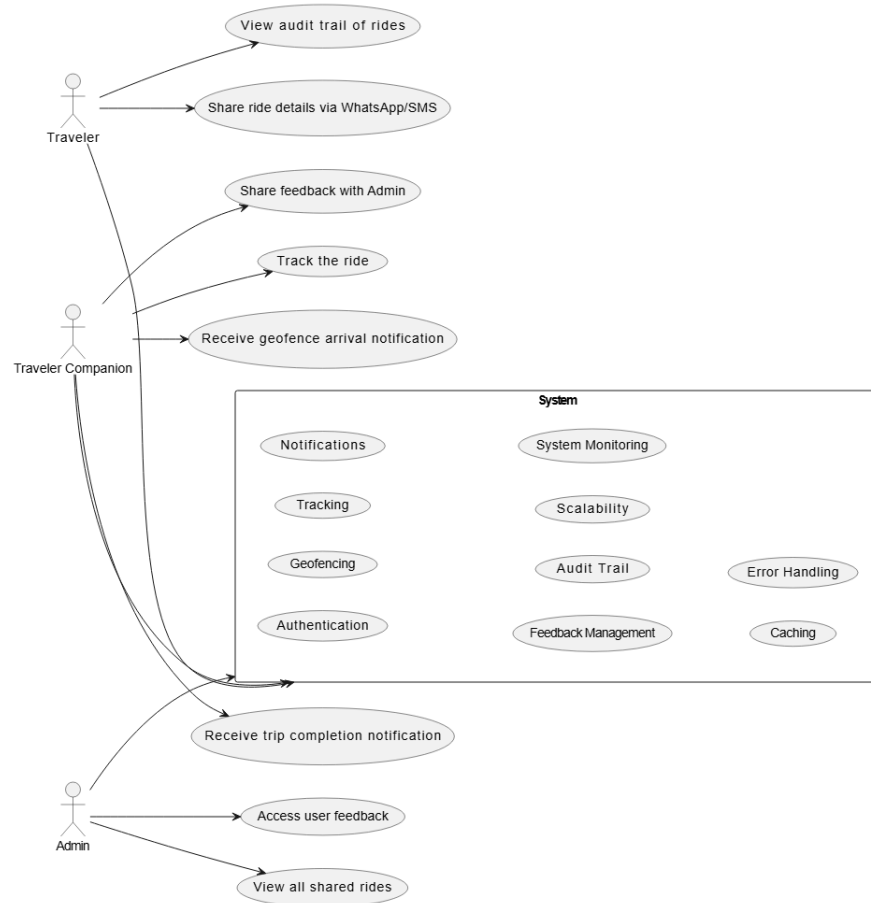
- Backend
 - Controllers
 - Middlewares
 - Models
 - Routes
 - Utils
 - Server.js
 - Package.json
 - Package-lock.json
- Frontend
 - Public
 - Src
 - Pages
 - App.js
 - Index.js

Scope of Project

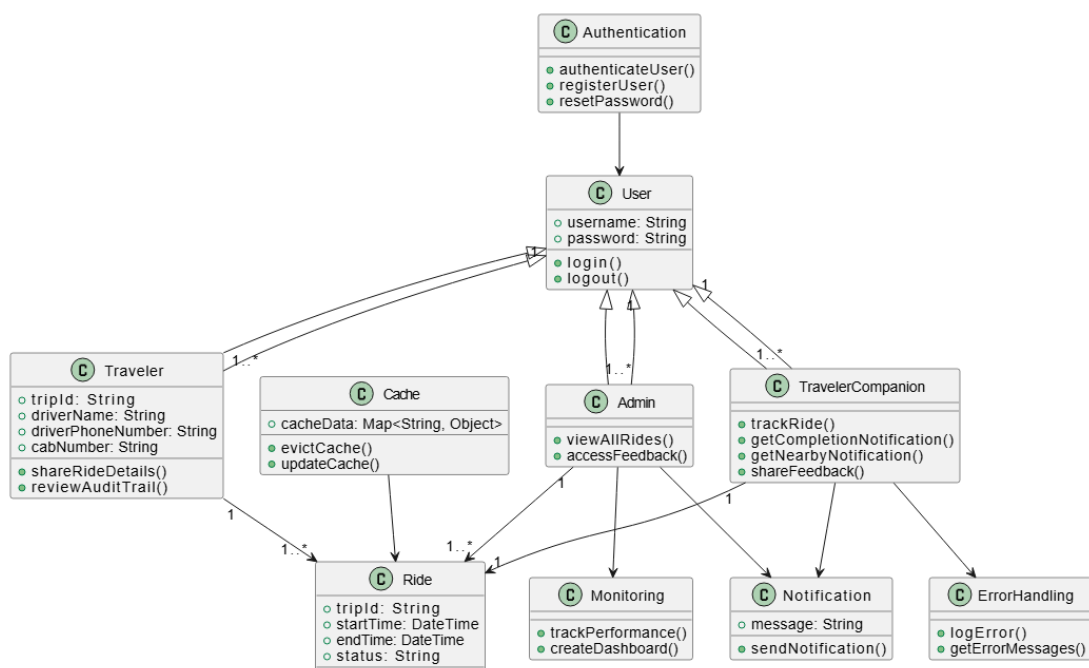
- Traveller Features: Share trip details securely via WhatsApp/SMS with an expiring link and access an audit trail of shared rides.
- Traveller Companion: Track the Traveller's ride, receive notifications for trip completion and geofence entry, and provide feedback to the Admin.
- Security and Scalability: Implement strong authentication, data privacy measures, and design for scalability on cloud infrastructure.
- Geofencing and Notifications: Enable real-time ride tracking and notifications based on location-based triggers.
- Out of Scope: Payment processing, integration with external ride-hailing services, and advanced route optimization.

UML Diagrams

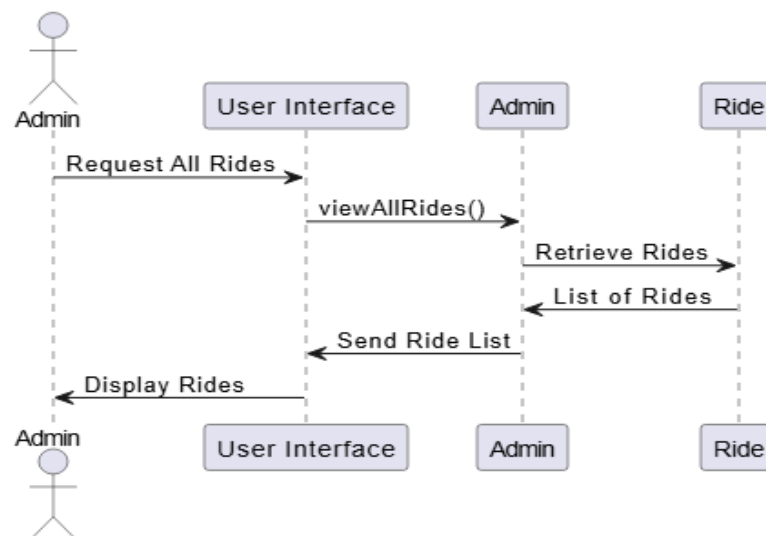
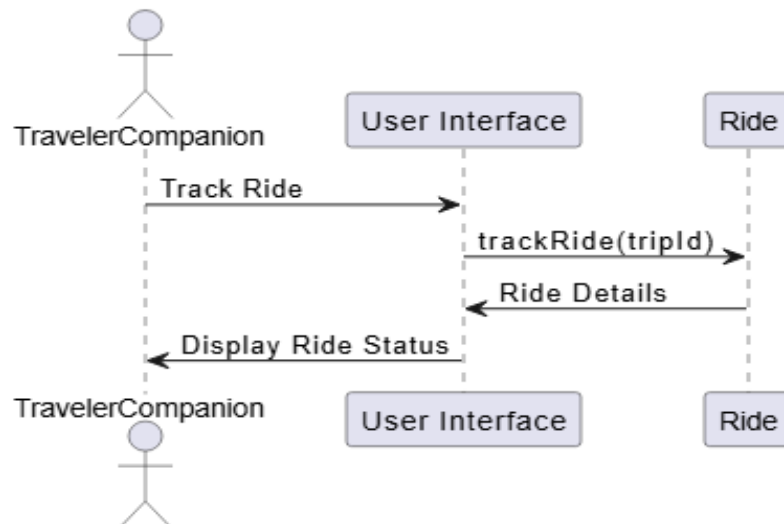
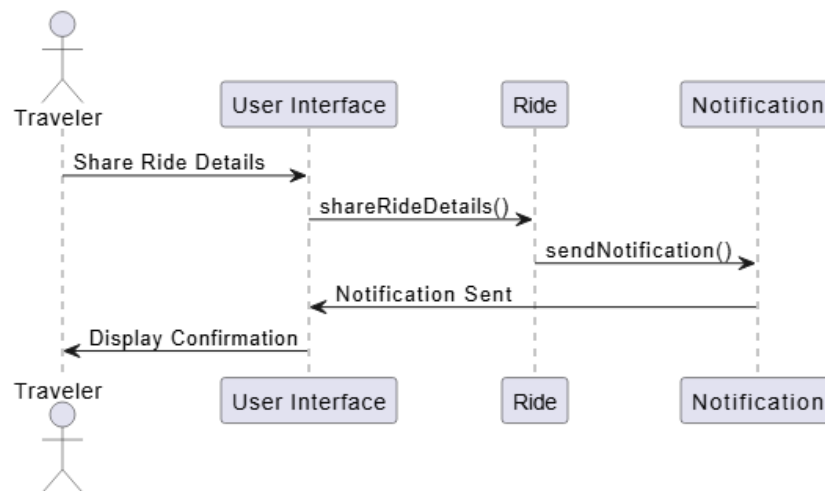
Use case:



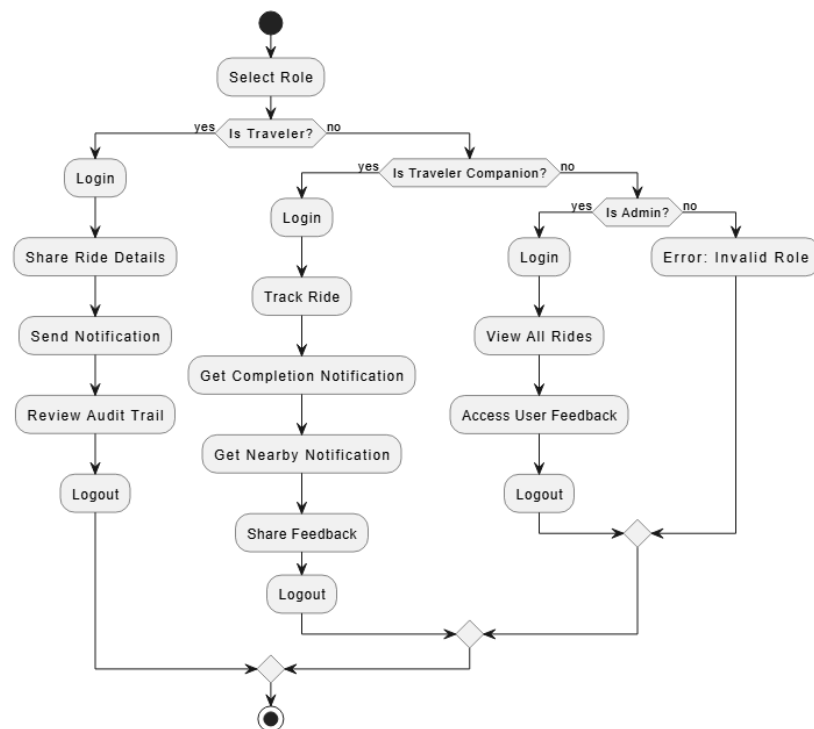
Class Diagram:



Sequential Diagram:



Activity Diagram:



Implementation

- Start the backend server and frontend using command: npm start
- Login Page

UI:

The UI mockup shows a login page with a white card on a light gray background. The card has a title 'Welcome' and a subtitle 'Enter your credentials to access your account'. Below the subtitle are two input fields: 'Email' with a person icon and 'Password' with a lock icon. A blue 'Sign in' button is positioned below the password field. At the bottom of the card, there is a link: 'Don't have an account? [Sign up](#)'.

Code Snippet:

```
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import { User, Lock } from 'lucide-react';

const Login = ({ setUser }) => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);
  const navigate = useNavigate();

  const handleLogin = async (e) => {
    e.preventDefault();
    setError('');
    setLoading(true);

    try {
      const { data } = await axios.post('http://localhost:5000/api/auth/login', {
        email,
        password
      });
      localStorage.setItem('token', data.token);
      setUser(data);
      navigate('/');
    } catch (error) {
      setError(error.response?.data?.message || 'Failed to login. Please try again.');
```

You, 20 hours ago • Uncommitted changes

```
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="login-container">
      <div className="login-card">
        <div className="card-header">
          <h2>Welcome</h2>
          <p>Enter your credentials to access your account</p>
        </div>

        <div className="card-content">
          <form onSubmit={handleLogin}>
            {error && (
              <div className="error-alert">
                {error}
              </div>
            )}

            <div className="input-group">
              <div className="input-wrapper">
                <User className="input-icon" />
                <input
                  type="email"
                  placeholder="Email"
                  value={email}
                  onChange={(e) => setEmail(e.target.value)}
                  required
                />
              </div>
            </div>
          </form>
        </div>
      </div>
    </div>
  );
};
```

Note: for details code kindly visit GitHub repository provided at end.

Explanations:

- Login page is created using React with JavaScript as shown in above code snippet.
- Styling is done with CSS
- Various react libraries have been used like Lucid-react, for icons
- API calling have been done for storing credentials in Mongo DB, and tokens in local storage.

Code Snippet of Schema for users and ride:

```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const UserSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  role: { type: String, enum: ['Traveler', 'TravelerCompanion', 'Admin'], required: true },
});

// Encrypt password before saving
Tabnine | Edit | Test | Explain | Document | Ask
UserSchema.pre('save', async function (next) {
  if (!this.isModified('password')) return next();
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});

// Compare entered password with hashed password
Tabnine | Edit | Test | Explain | Document | Ask
UserSchema.methods.matchPassword = async function (enteredPassword) {
  return await bcrypt.compare(enteredPassword, this.password);
};

module.exports = mongoose.model('User', UserSchema);

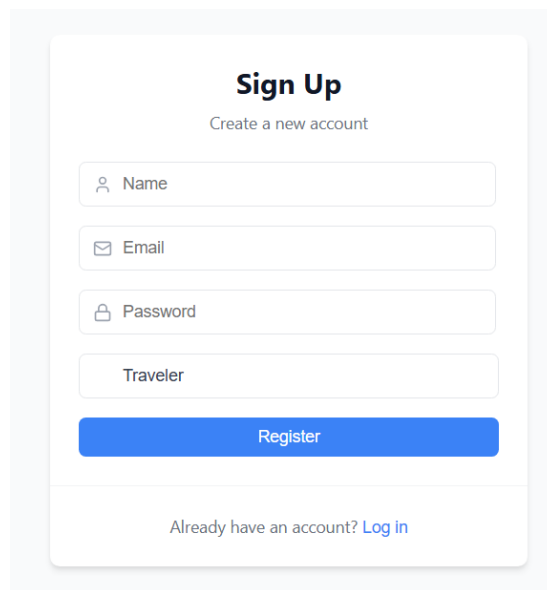
const mongoose = require('mongoose');

const RideSchema = new mongoose.Schema({
  driver: { type: String, required: true },
  traveler: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  driverPhone: { type: String, required: true },
  cabNumber: { type: String, required: true },
  tripId: { type: String, required: true, unique: true },
  status: { type: String, enum: ['in-progress', 'completed'], default: 'in-progress' },
  startLocation: {
    latitude: { type: Number, required: true },
    longitude: { type: Number, required: true }
  },
  endLocation: {
    latitude: { type: Number, required: true },
    longitude: { type: Number, required: true }
  },
  currentLocation: {
    latitude: { type: Number },
    longitude: { type: Number }
  },
  createdAt: { type: Date, default: Date.now },
});

module.exports = mongoose.model('Ride', RideSchema);
```

Sign Up Page

UI:



Sign Up
Create a new account

Register

Already have an account? [Log in](#)

Code Snippet:

```
import React, { useState } from "react";
import axios from "axios";
import { useNavigate } from "react-router-dom";
import { User, Mail, Lock } from "lucide-react";

const Register = () => {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [role, setRole] = useState("Traveler");
  const [message, setMessage] = useState("");
  const navigate = useNavigate();

  const handleRegister = async (e) => {
    e.preventDefault();
    try {
      const { data } = await axios.post(
        "http://localhost:5000/api/auth/register",
        {
          name,
          email,
          password,
          role,
        }
      );
      setMessage("Registration successful! Redirecting to login...");
      setTimeout(() => {
        navigate("/login");
      }, 2000);
    } catch (error) {
      setMessage("Error registering. Please try again.");
      console.error("Registration error:", error);
    }
  };
};
```

```

return (
  <div className="login-container">
    <div className="login-card">
      <div className="card-header">
        <h2>Sign Up</h2>
        <p>Create a new account</p>
      </div>

      <div className="card-content">
        <form onSubmit={handleRegister}>
          {message && <div className="error-alert">{message}</div>}

          <div className="input-group">
            <div className="input-wrapper">
              <User className="input-icon" />
              <input
                type="text"
                placeholder="Name"
                value={name}
                onChange={(e) => setName(e.target.value)}
                required
              />
            </div>
          </div>

          <div className="input-group">
            <div className="input-wrapper">
              <Mail className="input-icon" />
              <input
                type="email"
                placeholder="Email"
                value={email}
                onChange={(e) => setEmail(e.target.value)}
                required
              />
            </div>
          </div>
        </form>
      </div>
    </div>
  </div>
)

```

Explanation:

- If not registered, then sign up by clicking on Sign up and fill the details on the sign up page.
- After submitting it will automatically redirect to login page.
- Credentials to be stored in Mongo DB database.
- Login again to navigate to dashboard.
- Authentication is done through JWT token.

4. Authentication:

Code snippet:

authController

```
const User = require('../models/User');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');

const generateToken = (id) => {
  return jwt.sign({ id }, process.env.JWT_SECRET, { expiresIn: '30d' });
};

// Register a new user
Tabnine | Edit | Test | Explain | Document | Ask
exports.registerUser = async (req, res) => {
  const { name, email, password, role } = req.body;
  try {
    const userExists = await User.findOne({ email });
    if (userExists) return res.status(400).json({ message: 'User already exists' });

    const user = await User.create({ name, email, password, role });
    res.status(201).json({
      _id: user._id,
      name: user.name,
      email: user.email,
      role: user.role,
      token: generateToken(user._id),
    });
  } catch (error) {
    res.status(500).json({ message: 'Server error', error });
  }
};
```

```
// User Login
Tabnine | Edit | Test | Explain | Document | Ask
exports.loginUser = async (req, res) => {
  const { email, password } = req.body;
  try {
    const user = await User.findOne({ email });
    if (user && (await user.matchPassword(password))) {
      res.json({
        _id: user._id,
        name: user.name,
        email: user.email,
        role: user.role,
        token: generateToken(user._id),
      });
    } else {
      res.status(401).json({ message: 'Invalid credentials' });
    }
  } catch (error) {
    res.status(500).json({ message: 'Server error', error });
  }
};
```

authMiddleware

```
const jwt = require('jsonwebtoken');
const User = require('../models/User');

Tabnine | Edit | Test | Explain | Document | Ask
exports.protect = async (req, res, next) => {
  let token;
  if (req.headers.authorization && req.headers.authorization.startsWith('Bearer')) {
    try {
      token = req.headers.authorization.split(' ')[1];
      const decoded = jwt.verify(token, process.env.JWT_SECRET);
      req.user = await User.findById(decoded.id).select('-password');
      next();
    } catch (error) {
      res.status(401).json({ message: 'Not authorized, token failed' });
    }
  }
  if (!token) res.status(401).json({ message: 'Not authorized, no token' });
};
```

authRoutes

```
const express = require('express');
const { registerUser, loginUser } = require('../controllers/authController');
const router = express.Router();

router.post('/register', registerUser);
router.post('/login', loginUser);

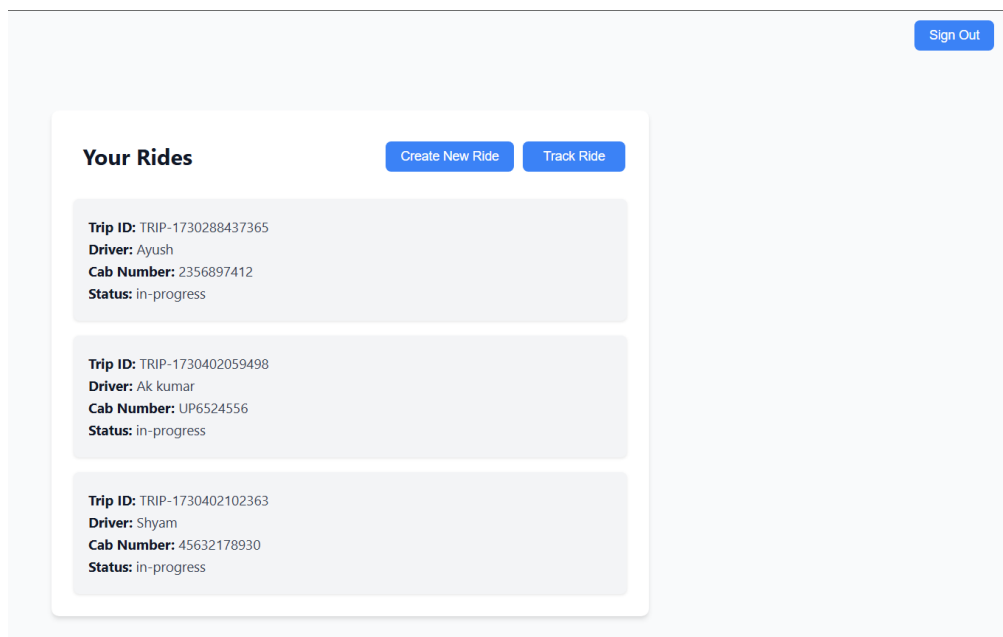
module.exports = router;
```

Explanation:

1. JWT tokens is being used for authentication and stored in local storage.
2. authController, authMiddleware and authRoutes have been defined as above.
3. During registration it is stored and then it is being verified during login.

5. Dashboard

UI



Code snippets:

```
import React, { useEffect, useState } from "react";
import axios from "axios";
import { Link, useNavigate } from "react-router-dom";

You, yesterday • first_commit

const Dashboard = () => {
  const [rides, setRides] = useState([]);
  const navigate = useNavigate();

  useEffect(() => {
    const fetchRides = async () => {
      try {
        const token = localStorage.getItem("token");
        const config = {
          headers: {
            Authorization: `Bearer ${token}`,
          },
        };
        const { data } = await axios.get(
          "http://localhost:5000/api/rides",
          config
        );
        setRides(data);
      } catch (error) {
        console.error("Error fetching rides:", error);
      }
    };
    fetchRides();
  }, []);

  const handleSignOut = () => {
    localStorage.removeItem("token"); // Remove token from LocalStorage
    navigate("/login"); // Redirect to Login page
  };
};
```

```

return (
  <div className="main-container">
    <button className="signout-button" onClick={handleSignOut}>
      Sign Out
    </button>

    <div className="dashboard-container">
      <div className="dashboard-card">
        <div className="card-header">
          <h2>Your Rides</h2>
          <Link to="/create-ride">
            <button className="create-button">Create New Ride</button>
          </Link>
          <Link to="/track">
            <button className="track-button">Track Ride</button>
          </Link>
        </div>
      </div>
    </div>
  </div>
)

```

Explanations:

1. Dashboard has sign out button for sign out and it navigate to login page.
2. It is done by clearing tokens from local storage.
3. Dashboard Shows previously created rides by fetching from database.
4. Dashboard has options for creating new rides and tracking rides.
5. Click on these buttons for desired actions.

6. Creating Rides:

UI

Create a New Ride

Start Location

End Location

Code snippets:

```
You, 7 hours ago | 1 author (You)
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import Radar from "radar-sdk-js";
import axios from "axios";
import emailjs from '@emailjs/browser';

Radar.initialize("prj_live_pk_62bbb2ba7cc089ea9d9f98dfbf0dd3a3830bfedb");
emailjs.init("r1fW0MTiQI23ByQic");

const CreateRide = () => {
  const navigate = useNavigate();
  const [driver, setDriver] = useState("");
  const [driverPhone, setDriverPhone] = useState("");
  const [cabNumber, setCabNumber] = useState("");
  const [startLocationName, setStartLocationName] = useState("");
  const [endLocationName, setEndLocationName] = useState("");
  const [message, setMessage] = useState("");

  const fetchCoordinates = async (locationName) => {
    try {
      const loc = await Radar.autocomplete({
        query: locationName,
        limit: 1,
      });
      return {
        latitude: loc.addresses[0].latitude,
        longitude: loc.addresses[0].longitude,
      };
    } catch (error) {
      console.error("Error fetching coordinates:", error);
      return { latitude: "", longitude: "" };
    }
  };
};
```

```
const CreateRide = () => {
  // ...
};

const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    const startCoords = await fetchCoordinates(startLocationName);
    const endCoords = await fetchCoordinates(endLocationName);

    const token = localStorage.getItem("token");
    const config = {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    };
    const rideData = {
      driver,
      driverPhone,
      cabNumber,
      startLocation: startCoords,
      endLocation: endCoords,
    };
  };
};
```

```

const { data } = await axios.post(
  "http://localhost:5000/api/rides/create",
  rideData,
  config
);
setMessage("Ride created successfully!");
emailjs
  .send(
    "service_4djdxz9", // Replace with your EmailJS service ID
    "template_wwf3c28", // Replace with your EmailJS template ID
    rideData, // Replace with your
    "r1fW0MTiQI23ByQic" // Replace with your EmailJS user ID
  )
  .then(
    (result) => {
      console.log("Email sent:", result.text);
      setMessage("Ride shared successfully via email!");
    },
    (error) => {
      console.error("Email error:", error.text);
      setMessage("Error sharing ride.");
    }
  );
navigate("/");
} catch (error) {
  setMessage("Error creating ride.");
  console.error("Ride creation error:", error);
}
}

```

Explanations:

1. Here you can create new rides and as well as share it to other over e-mails.
2. Details to be stored in MongoDB
3. Error handling is done in each and every important part.
4. Sharing through e-mail is achieved by using Email JS.
5. Click on share ride to create and share ride.
6. Click on create ride for only ride creation or use them according to needs.
7. Clicking on any button will redirect to dashboard after its working.
8. Screen shot of email sent is also attached.

New message from Ayush



Ayush Singh <12112029it@gmail.com>

to me ▾

Hello ,

You got a new message from Ayush:

driver: Ayush

driverPhone: 1234567890

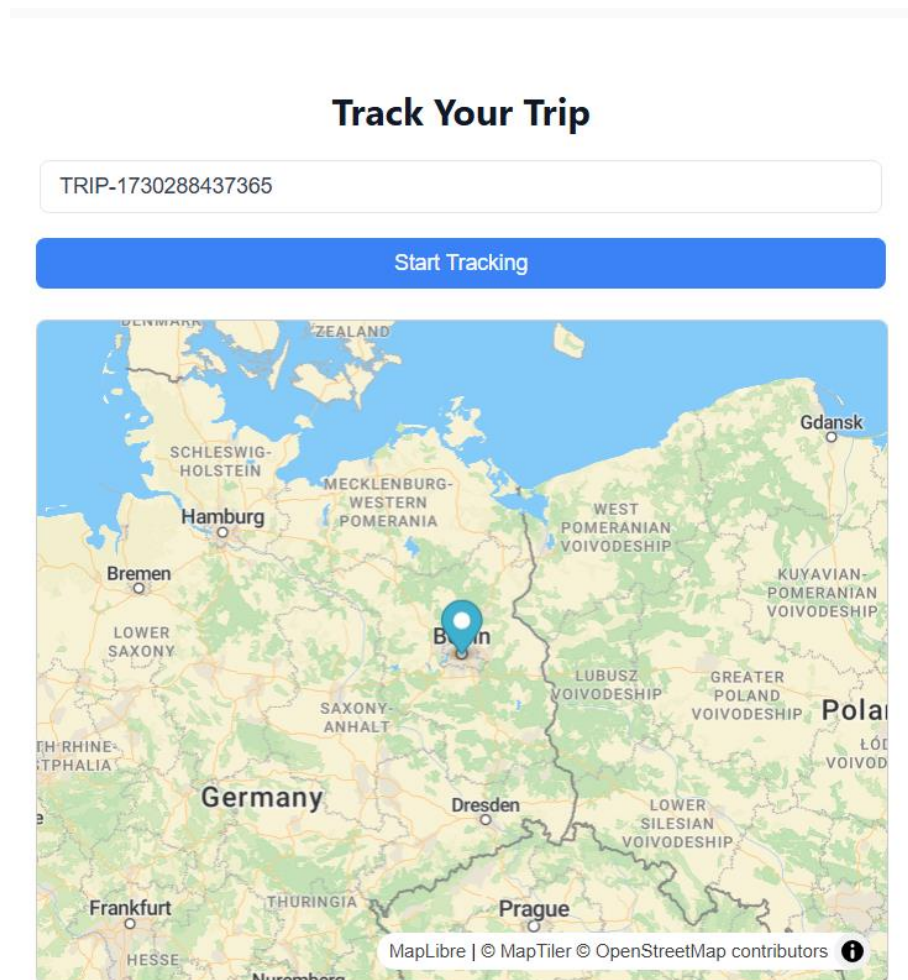
Vehicle Number:0001

tripId:TRIP-1730288437365

Note: It is not hard coded, for reference watch demo video for live sending.

7. *Ride Tracking:

UI:



Code Snippets:

```
import React, { useState, useEffect } from "react";
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import MapLibre from "maplibre-gl";
import Radar from "radar-sdk-js";
import axios from "axios";
import { circle } from "@turf/turf";
import "maplibre-gl/dist/maplibre-gl.css";

// Initialize Radar with your API key
Radar.initialize("prj_live_pk_62bbb2ba7cc089ea9d9f98dfbf0dd3a3830bfedb");
```

Tabnine | Edit | Test | Fix | Explain | Document | Ask

```
function TripTracker() {
  const [tripId, setTripId] = useState("");
  const [tracking, setTracking] = useState(false);
  const [location, setLocation] = useState(null);
  const [travellerLocation, setTravellerLocation] = useState(null);
  const [map, setMap] = useState(null);
  const [marker, setMarker] = useState(null);
  const [travellerMarker, setTravellerMarker] = useState(null);
  const [once, setOnce] = useState(false);
  const [hasShownToast, setHasShownToast] = useState(false);
```

```
useEffect(() => {
  if (location && !map) {
    const mapInstance = new MapLibre.Map({
      container: "map",
      style: `https://api.maptiler.com/maps/streets-v2/style.json?key=z0kg5iNOA7kBJR5nnGF7`,
      center: [location.Longitude, location.Latitude],
      zoom: 5,
    });

    const newMarker = new MapLibre.Marker()
      .setLngLat([location.Longitude, location.Latitude])
      .addTo(mapInstance);
    const newTravellerMarker = new MapLibre.Marker({
      color: "yellow",
    })
      .setLngLat([travellerLocation.Longitude, travellerLocation.Latitude])
      .addTo(mapInstance);

    setMarker(newMarker);
    setTravellerMarker(newTravellerMarker);
    setMap(mapInstance);
  }
}, [location]);
```

Explanation:

1. Above code snippet is for implementation of ride tracking system.
2. Above useEffect section is for Map instance generation using **MapLibre**.
3. Marker has also been created to show location pinned.

```
useEffect(() => {
  if (tracking) {
    const interval = setInterval(() => {
      Radar.trackOnce()
        .then((result) => {
          const { location } = result;
          if (location) {
            setLocation(location);
            if (marker) {
              marker.setLngLat([location.longitude, location.latitude]);
            }
            if (map && !once) {
              setOnce(true);
              map.flyTo({
                center: [location.longitude, location.latitude],
                zoom: 5,
              });
            }
          }
        })
        .catch((err) => console.error("Error tracking location:", err));
      }, 15000);

    return () => clearInterval(interval);
  }
}, [tracking, map, marker]);
```

Explanations:

1. Above code snippet is used for tracking current location using **Radar.io** API and stored in **location** state variable.
2. As soon as we enter **trip ID** and click start tracking button it starts tracking **current location**.
3. Interval of **15 second** has been set after which it again fetches current location.
4. Error handling is done in each important part.

Note: Prefer demo video if you want to know more.

```
return (
  <div className="trip-tracker-container">
    <div className="trip-tracker-card">
      <h2>Track Your Trip</h2>
      <input
        type="text"
        value={tripId}
        onChange={(e) => setTripId(e.target.value)}
        placeholder="Enter Trip ID"
        className="trip-input"
      />
      <button
        onClick={handleStartTracking}
        disabled={!tripId || tracking}
        className="start-button"
      >
        Start Tracking
      </button>
      <div id="map" className="map-container">
        {location ? (
          <p className="location-info">
            Current Location: {location.latitude.toFixed(5)},{" "}
            {location.longitude.toFixed(5)}
          </p>
        ) : (
          <p className="location-info">Location not available</p>
        )}
      </div>
    </div>
  </div>
)
```

Explanations:

1. Returns UI components to user.

```
const express = require('express');
const { createRide, getRides, updateRideStatus, updateLocation, getRideByTripId } = require('../controllers/rideController');
const router = express.Router();

// Authentication middleware can be added to protect these routes
const { protect } = require('../middlewares/authMiddleware');

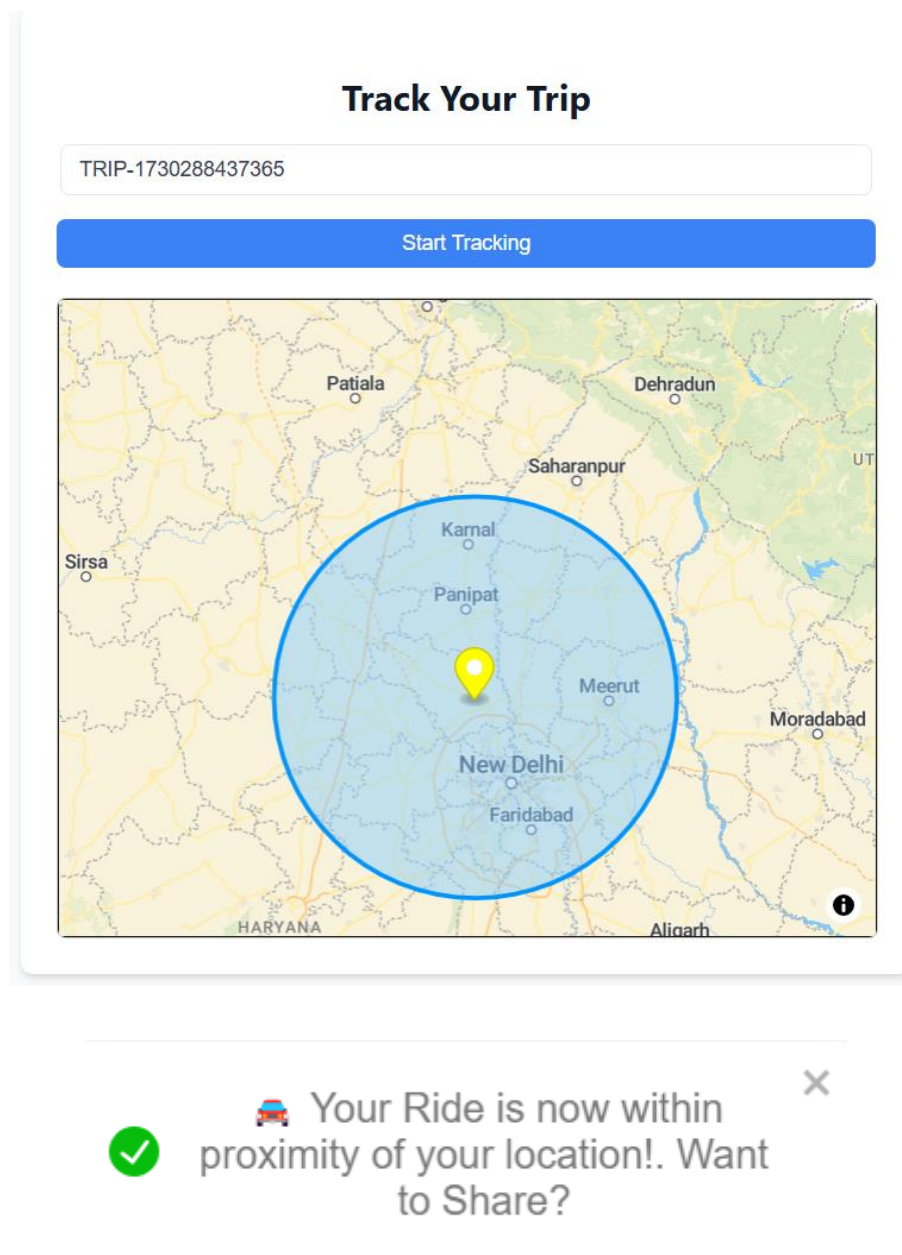
router.post('/create', protect, createRide); // Create a new ride
router.get('/', protect, getRides); // Get rides based on role
router.put('/:rideId', protect, updateRideStatus); // Update ride status
router.put('/:rideId/location', protect, updateLocation);
router.post('/', protect, getRideByTripId); // Get trip based on trip id

module.exports = router;
```

Explanation: Routes related to all ride related things.

8. **Geofencing, Polygon/Circle generation and Alert notification:

UI:

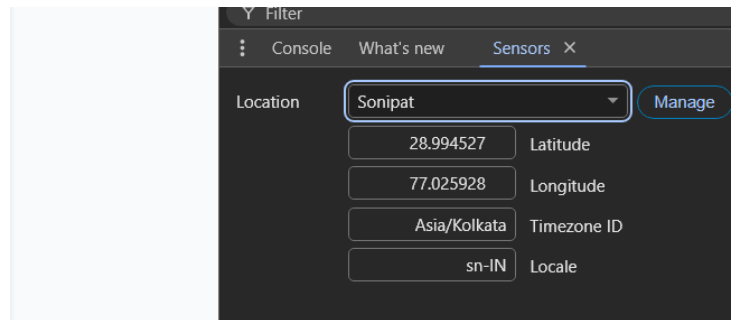
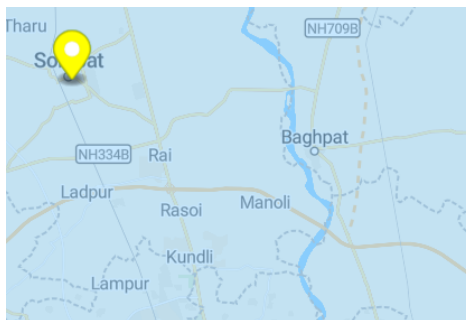
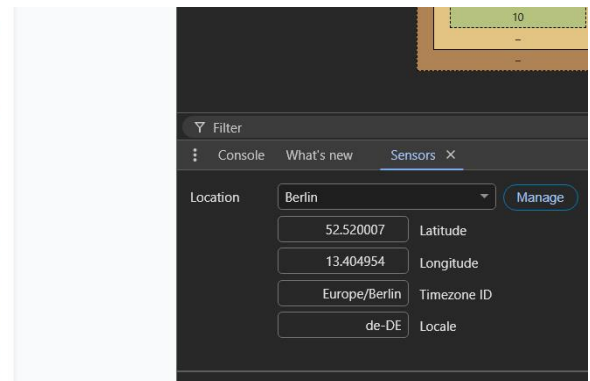
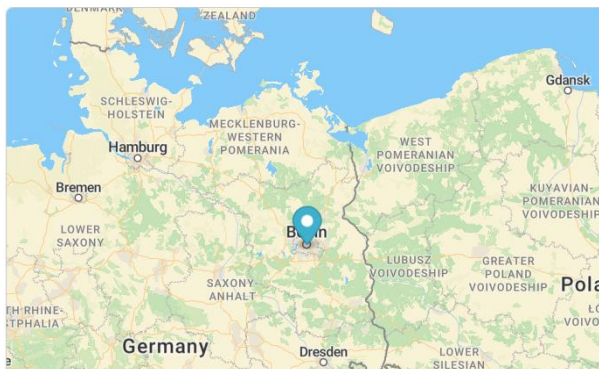


Explanations:

1. Geofencing is done in the circle form as shown above and it has yellow marker while current location has blue marker.
2. Geofenced area is shown by circle boundary.
3. 2nd image is showing notification as soon as current location of ride is in geofence region.

Note: It's not hard coded as clearly shown in demo video and how to change current location of pc while sitting in one place to check it's working.

4. It's done by changing Latitude and longitude using sensors (since it is not possible to move using pc to change current location).
5. For example, first the current location was Berlin and then it is changed to sonipat as shown below.



Code Snippets:

```
const newMarker = new MapLibre.Marker()  
  .setLngLat([location.longitude, location.latitude])  
  .addTo(mapInstance);  
const newTravellerMarker = new MapLibre.Marker({  
  color: "yellow",  
})  
  .setLngLat([travellerLocation.longitude, travellerLocation.latitude])  
  .addTo(mapInstance);  
  
setMarker(newMarker);  
setTravellerMarker(newTravellerMarker);  
setMap(mapInstance);  
}
```



```

useEffect(() => {
  if (tracking && tripId && map && travellerLocation) {
    map.on("load", () => {
      const radiusCenter = [
        travellerLocation.longitude,
        travellerLocation.latitude,
      ];
      const radius = 100;
      const options = { steps: 64, units: "kilometers" };
      const circleDrawn = circle(radiusCenter, radius, options);

      if (!map.getSource("location-radius")) {
        map.addSource("location-radius", {
          type: "geojson",
          data: circleDrawn,
        });
        map.addLayer({
          id: "location-radius",
          type: "fill",
          source: "location-radius",
          paint: { "fill-color": "#8CCFFF", "fill-opacity": 0.5 },
        });
        map.addLayer({
          id: "location-radius-outline",
          type: "line",
          source: "location-radius",
          paint: { "line-color": "#0094ff", "line-width": 3 },
        });
      } else {
        map.getSource("location-radius").setData(circleDrawn);
      }
    });
  }
}, [tracking, tripId, map, travellerLocation]);

```

Explanations:

1. Above 2 code snippet is used for generating Geofencing circle and yellow marker.
2. Here **travellerLocation** state variable contains end location co-ordinates where Geofencing will be created.
3. Geofencing is done automatically as soon as traveller create any trip.
4. Circle is made to track proximity evenly in all direction of **Geofencing**.

```

useEffect(() => {
  if (tracking && tripId) {
    try {
      const interval = setInterval(async () => {
        const { data } = await axios.post(
          `http://localhost:5000/api/rides`,
          { tripId },
          {
            headers: {
              Authorization: `Bearer ${localStorage.getItem("token")}`,
            },
          },
        );
        if (data) {
          setTravellerLocation(data.endLocation);
          if (travellerMarker) {
            travellerMarker.setLngLat([
              travellerLocation.Longitude,
              travellerLocation.Latitude,
            ]);
          }
        }
      }, 15000);

      return () => clearInterval(interval);
    } catch (e) {
      console.error(e);
    }
  }
}, [tracking, map, travellerMarker]);

```

Explanations:

1. Above code snippet used for calculating end location by fetching it from ride details present in database.
2. It is then stored in **travellerLocation** state variable.
3. Then it is used for generating geo-fence as discussed previously.

Most important part: How to calculate whether the traveller current location is in geofence or not?

Explanations:

1. The Haversian formula is used to calculate the shortest distance between two points on a sphere (Earth, in most applications) given their latitudes and longitudes. It is widely used in navigation and geolocation services.
2. As shown below firstly, the coordinates are converted to radian.
3. Then latitude & longitude of current and geofence location is used to calculate distance between them by the using formula shown below.

$$d = 2R \cdot \arcsin \left(\sqrt{\sin^2 \left(\frac{\Delta \text{lat}}{2} \right) + \cos(\text{lat}_1) \cdot \cos(\text{lat}_2) \cdot \sin^2 \left(\frac{\Delta \text{lng}}{2} \right)} \right)$$

Code snippet:

```
const haversineDistance = (coords1, coords2) => {  
  const toRadians = (value) => (value * Math.PI) / 180;  
  const R = 6371e3; // Earth's radius in meters  
  
  const lat1 = toRadians(coords1.latitude);  
  const lon1 = toRadians(coords1.longitude);  
  const lat2 = toRadians(coords2.latitude);  
  const lon2 = toRadians(coords2.longitude);  
  
  const deltaLat = lat2 - lat1;  
  const deltaLon = lon2 - lon1;  
  
  const a = Math.sin(deltaLat / 2) * Math.sin(deltaLat / 2) +  
    Math.cos(lat1) * Math.cos(lat2) *  
    Math.sin(deltaLon / 2) * Math.sin(deltaLon / 2);  
  
  const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));  
  return R * c; // Distance in meters  
};
```

```
const isWithinRadius = (currentLocation, destination, radius = 500) => {
  const distance = haversineDistance(currentLocation, destination);
  return distance <= radius; // Return true if within the radius
};
```

Explanation:

1. The **isWithinRadius** function is created to check the obtained **proximity** distance is within geofence or not.

```
const calculateProximityToEndLocation = async (location) => {
  if (travellerLocation) {
    if (isWithinRadius(location, travellerLocation)) {
      console.log("User is within proximity of the end location.");
      // Trigger push notification or any other action here
      if(!hasShownToast){
        toast.success("📍 Your Ride is now within proximity of your destination!. Want to Share?", {
          autoClose: 4000, // duration in milliseconds
          hideProgressBar: true,
          closeOnClick: true,
          pauseOnHover: true,
          draggable: false,
        });
        setHasShownToast(true);
      }
    } else {
      console.log("User is outside the end location geofence.");
      setHasShownToast(false);
    }
  } else {
    console.log("End location not found.");
  }
};
```

```
useEffect(() => {
  if (location) {
    calculateProximityToEndLocation(location); // Check proximity whenever location updates
  }
}, [location]);
```

Explanations:

1. Above 2 code snippets show how to trigger notification when the traveller is in geofence area.
2. After it triggers, it then shows the **Notification** with custom message that has certain properties which can be changed depending on the requirements.
3. Through this traveller companion will get know whether the ride is in its proximity range or not.
4. As soon as the end location changes; geofence area also change.

GitHub repository: [Click Here](#)

Demo Video: [Click Here](#)