

1. INTRODUCTION

The rise of ride-hailing apps has been nothing short of transformative in the world of transportation. These digital platforms have fundamentally altered the way people move around cities, suburbs, and rural areas. From Uber and Lyft to local and international alternatives, ride-hailing apps have rapidly become an integral part of modern urban mobility. This essay delves into the purpose and scope of ride-hailing apps, exploring their impact on society, economy, and urban planning.

1.1. Purpose

Ride-hailing apps were conceptualized with the primary purpose of providing a convenient, safe, and efficient alternative to traditional taxis and public transportation. They aimed to address common pain points associated with these modes of transport:

- **Convenience:** Ride-hailing apps offer users the convenience of requesting a ride with a few taps on their smartphones. Gone are the days of flagging down taxis or waiting at bus stops.
- **Safety:** These apps introduced enhanced safety features such as driver and passenger ratings, real-time tracking, contributing to a safer travel experience.
- **Predictability:** Riders can estimate fares in advance, eliminating the uncertainty associated with traditional taxis. This transparency is a welcomed change.
- **Accessibility:** Ride-hailing services are often more accessible to people with disabilities, as they can request wheelchair-accessible vehicles when needed.
- **Employment Opportunities:** Beyond serving riders, ride-hailing platforms have created job opportunities for countless drivers, contributing to economic growth.

1.2. Scope

The scope of ride-hailing apps extends far beyond the initial goal of providing convenient rides. Here are some of the broader impacts and implications:

- **Economic Impact:** Ride-hailing apps have disrupted the traditional taxi industry and created a gig economy where individuals can earn income by driving their own vehicles. This shift has both positive and negative economic consequences, including job creation and concerns about labor rights.
- **Urban Mobility:** The proliferation of ride-hailing services has had significant implications for urban planning. Cities are reevaluating public transportation systems and road infrastructure to accommodate the changing landscape of transportation.
- **Environmental Considerations:** While ride-hailing apps have reduced car ownership in some urban areas, concerns remain about their contribution to congestion and emissions. Initiatives to promote shared rides and electric vehicles are addressing these concerns.
- **Innovation and Competition:** The success of ride-hailing apps has inspired innovation in transportation, including autonomous vehicles and new business models. The competitive landscape continues to evolve.
- **Global Expansion:** Ride-hailing apps have expanded their operations internationally, making transportation more accessible in regions where traditional services may be lacking or unreliable.

2. SYSTEM ANALYSIS AND REQUIREMENTS

2.1. Analysis

- The Rango Ride hailing app is a app that uses real time location Sharing and data transfer.
- The App requires Location services such GPS to access the drives and passenger location in real time.
- It requires to use Maps to display the location data to the driver and passenger.
- The App should be able to use the local Storage of a Phone to store the user credentials and data.
- The Ability to Search for places for passengers and converting the places name to geographic coordinates so that they can be used on the map to display current location and directions.

2.2. Requirements

Hardware requirements:

For Consumer Usage:

- Android Mobile with Android OS version 6+

For Development:

- RAM: 8GB or above.
- Processor: intel i5.
- OS: Any OS that support newer versions of Android Emulator.

Software Tools or Requirements

- Nodejs
- MongoDB
- Express.js
- React-Native
- Mapbox Maps API.

Nodejs is a cross-platform, open-source server environment that can run on Windows, Linux, Unix, macOS, and more. Node.js is a back-end JavaScript runtime environment, runs on the V8 JavaScript Engine, and executes JavaScript code outside a web browser.

Node.js lets developers use JavaScript to write command line tools and for server-side scripting. The functionality of running scripts server-side produces dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, rather than different languages for server-side and client-side scripts.

Node.js has an event-driven architecture capable of asynchronous I/O. These design choices aim to optimize throughput and scalability in web applications with many input/output operations, as well as for real-time Web applications (e.g., real-time communication programs and browser games).

MongoDB is a popular open-source NoSQL document-oriented database that stores data in a JSON-like format called BSON. It was first introduced in 2009 and has since become a widely-used database system, especially for web and mobile applications.

One of the key features of MongoDB is its flexibility and scalability. It allows for the storage of large amounts of data and can easily scale horizontally to handle increasing amounts of traffic and data. MongoDB also supports a variety of data types, including arrays, embedded documents, and binary data.

Another important feature of MongoDB is its ability to provide high availability and fault tolerance. This is achieved through replication, sharding, and automatic failover mechanisms.

MongoDB is also designed to work well with modern programming languages and frameworks such as Node.js, Ruby, Python, and Java. It provides a rich set of APIs and drivers that allow developers to easily integrate it into their applications.

Express.js (or simply Express) is a popular open-source web application framework for Node.js. It provides a set of features for building web applications and APIs, including routing, middleware, and HTTP utilities.

Express allows developers to build web applications more easily by providing a simple, flexible, and minimalist approach to web development. It is built on top of Node.js and is designed to work well with other libraries and frameworks. It also provides a robust set of features such as error handling, session management, and templating engines.

One of the key benefits of using Express is its simplicity and flexibility. It provides a lightweight framework that allows developers to easily create web applications and APIs without being bogged down by excessive configuration and boilerplate code.

Express also has a large and active community, which means that developers can easily find documentation, tutorials, and support when needed. Additionally, it provides a rich set of third-party middleware and plugins, which can be used to extend its functionality even further.

React Native (also known as RN) is a popular JavaScript-based mobile app framework that allows you to build natively-rendered mobile apps for iOS and Android. The framework lets you create an application for various platforms by using the same codebase.

React Native was first released by Facebook as an open-source project in 2015. In just a couple of years, it became one of the top solutions used for mobile development. React Native development is used to power some of the world's leading mobile apps, including Instagram, Facebook, and Skype. We discuss these and other examples of React Native-powered apps further in this post.

There are several reasons behind React Native's global success.

- Firstly, by using React Native, companies can create code just once and use it to power both their iOS and Android apps. This translates to huge time and resource savings.
- Secondly, React Native was built based on React – a JavaScript library, which was already hugely popular when the mobile framework was released. We discuss the differences between React and React Native in detail further in this section.
- Thirdly, the framework empowered frontend developers, who could previously only work with web-based technologies, to create robust, production-ready apps for mobile platforms.
- Interestingly, as with many revolutionary inventions, React Native was developed as a response to...a big technological mistake.

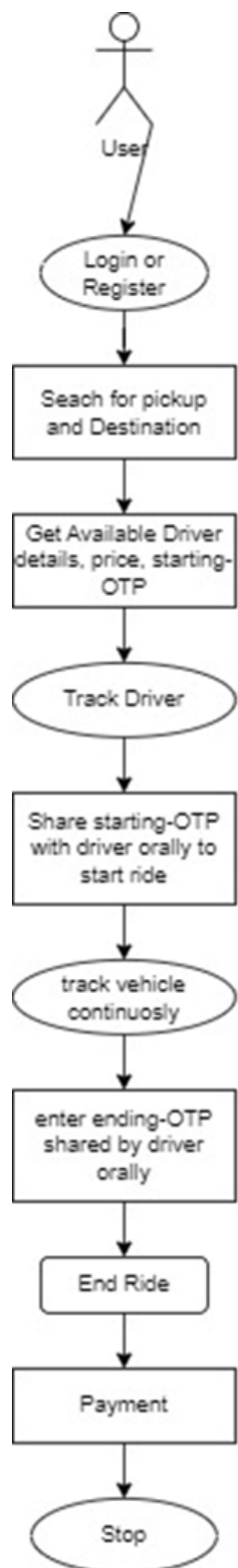
3. SYSTEM DESIGN

3.1. Use Case

A use case diagram is a type of behavioral diagram in the Unified Modeling Language (UML) that represents the interactions between a system and its external actors. It is a visual representation of the functional requirements of a system and illustrates the various use cases (i.e., scenarios or sequences of actions) that a system or a component of a system may have.

A use case diagram consists of several components, including actors, use cases, and relationships. Actors are external entities that interact with the system, such as users or other systems. Use cases represent the various functionalities or processes that the system provides to its actors. Relationships between actors and use cases show how the actors interact with the system and trigger the use cases.

Use case diagrams are useful for communicating the functional requirements of a system to stakeholders, including users, developers, and project managers. They can be used to identify the different features and functionalities of a system, as well as to identify any potential issues or gaps in the system's requirements. Use case diagrams can also be used to generate test cases and to validate the system's behavior.



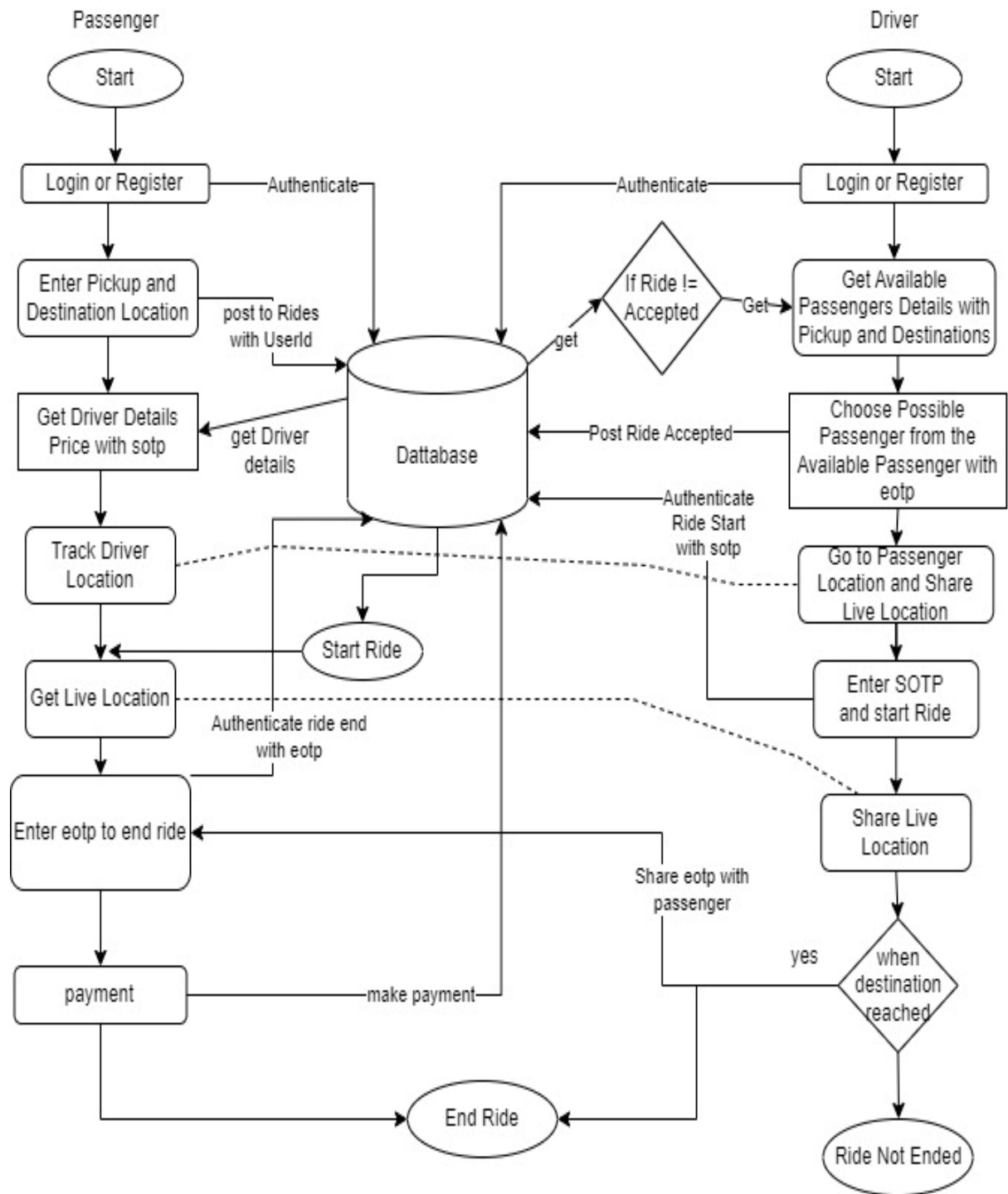
3.2 Data Flow Diagram

A data flow diagram (DFD) is a visual representation of the flow of data through a system or process. It is a type of diagram that shows how data is input, processed, and output in a system. DFDs are used to model the flow of data in a system, to identify the processes that are involved in handling the data, and to identify the data stores where the data is stored.

DFDs consist of several components, including processes, data stores, data flows, and external entities. Processes represent the various tasks or operations that are performed on the data. Data stores represent the locations where the data is stored or retrieved from. Data flows represent the movement of data between processes and data stores. External entities represent the sources or destinations of data outside of the system being modeled.

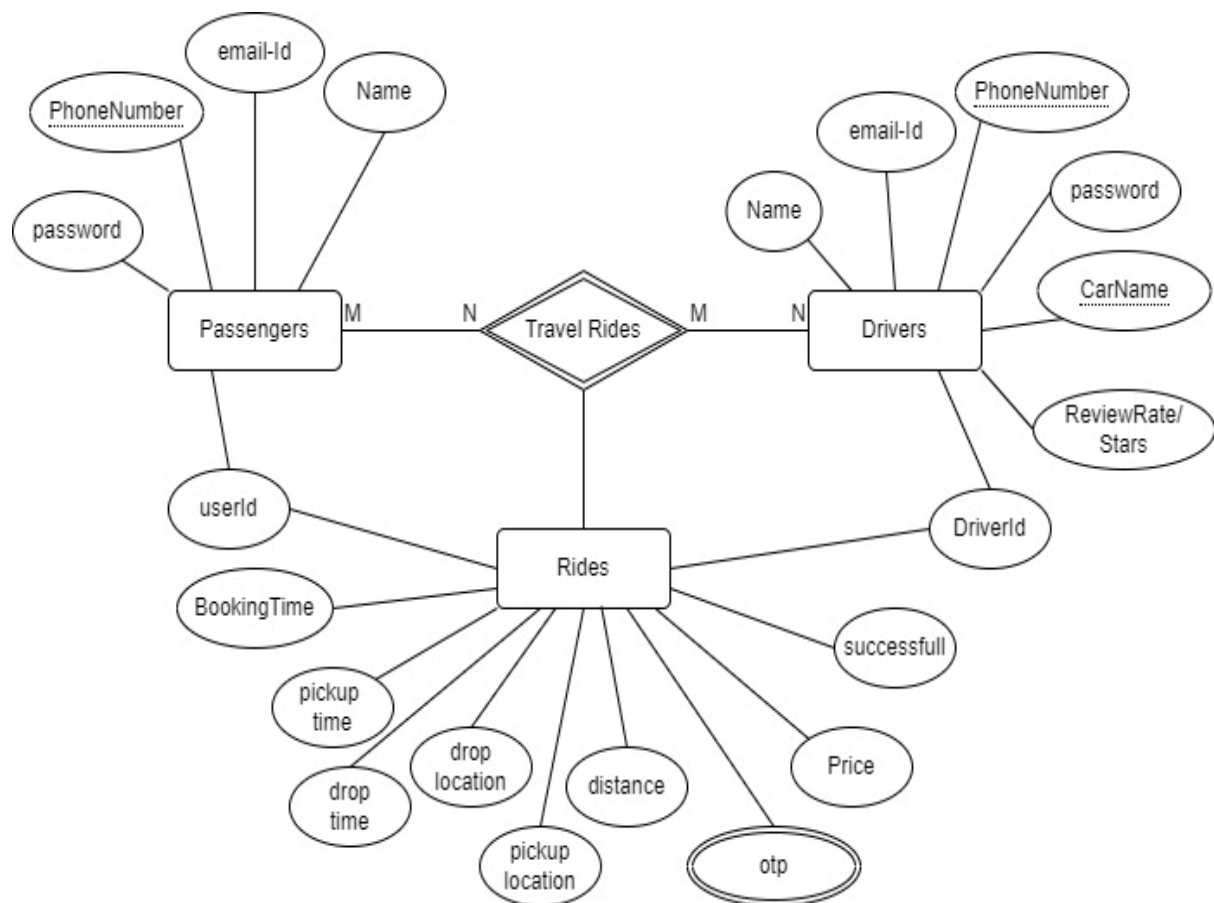
DFDs are useful for understanding and analyzing the flow of data in a system, and for identifying potential issues or areas for improvement. They are also useful for communicating the flow of data to stakeholders, including users, developers, and project managers. DFDs can be used to identify bottlenecks, redundancies, or inconsistencies in the data flow, and to design more efficient or effective processes for handling the data.

The application usage starts with User Logging into the application or registering if the user is new to the application. When the user logs in, the user is authenticated and redirected to the Chatting window or the Dashboard of the individual user. Through the chatting window the user can chat with other users in real-time. All the users data and messaging data are stored in the database instantly.



3.3 Entity Relationship Diagram

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research. Also known as ERDs or ER Models, they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes. They mirror grammatical structure, with entities as nouns and relationships as verbs.



3.4 Module Division

The Development process of the project is sub-divided into much smaller division or part so as to organize and Develop the Application in a transparent and understandable method. Each Module of the project development process were divided based on different feature or the functionality.

Developing a ride-hailing app with React Native and an Express backend with a Mongoose-MongoDB database involves dividing the application into various modules to ensure organized development. Here are ten essential modules for such an app:

The Module Division are as follows:

1. **Authentication Module:** Responsible for user registration and login. Manages user authentication and token-based security. Includes features like password reset and user profile management.
2. **User Management Module:** Allows users to create and manage their profiles. Handles user data, such as names, contact information, and payment methods.
3. **Ride Booking Module:** Enables users to request rides, specifying pick-up and drop-off locations. Utilizes real-time location services to calculate routes and provide fare estimates.
4. **Driver Management Module:** Allows drivers to sign up, submit required documents, and verify their identities. Manages driver availability and ride requests.
5. **Ride Matching and Allocation Module:** Implements algorithms for matching passengers with available drivers. Optimizes ride allocation based on proximity and demand.

6. **Payment and Billing Module:** Integrates with payment gateways to process ride payments. Manages transactions, invoices, and digital receipts.
7. **Rating and Feedback Module:** Allows passengers and drivers to rate and review each other after each ride. Manages feedback and resolves disputes if necessary.
8. **Navigation and Maps Module:** Integrates mapping services (e.g., Mapbox or Google Maps) for real-time navigation. Provides drivers with turn-by-turn directions to pick-up and drop-off locations.

These eight modules provide a structured framework for developing a ride-hailing app.

4. IMPLEMENTATION

4.1. Mobile App Source Code:

App.js /

```
import React from 'react';
import {View, SafeAreaView} from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import InitDriver from './driverScreens/initDriver';
import InitPassenger from './passengerScreens/initPassenger';
import Role from './role';
import { ParamProvider } from './ParamContext';

const Stack = createNativeStackNavigator();

function App(){
  return(
    <NavigationContainer>
      <ParamProvider>
        <Stack.Navigator initialRouteName="Role">
          <Stack.Screen component={Role} name="Role" options={{ headerShown:
false }}/>
          <Stack.Screen component={InitPassenger} name="InitPassenger" options={{
headerShown: false }}/>
          <Stack.Screen component={InitDriver} name="InitDriver" options={{
headerShown: false }} />
        </Stack.Navigator>
      </ParamProvider>
    </NavigationContainer>
  );
}
```

```

        </Stack.Navigator>

        </ParamProvider>

    </NavigationContainer>

    );
}

export default App;

```

role.jsx /

```

import React from 'react';

import {View, SafeAreaView, Text, Button, StyleSheet, TouchableOpacity} from 'react-native';

import InitDriver from './driverScreens/initDriver';
import InitPassenger from './passengerScreens/initPassenger';

function Role({navigation}){
    return(
        <View style={styles.container}>
            <Text style={styles.title}>CHOOSE YOUR ROLE</Text>

            <TouchableOpacity
                onPress={() => {navigation.navigate('InitDriver')}}
                style={styles.loginBtn}>

                <Text style={styles.loginText}>I AM A DRIVER </Text>

            </TouchableOpacity>

            <TouchableOpacity
                onPress={() => {navigation.navigate('InitPassenger')}}

```

```

        style={styles.loginBtn}>
        <Text style={styles.loginText}>I AM A PASSENGER </Text>
    </TouchableOpacity>
</View>
);
}

```

```

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#BBB',
    alignItems: 'center',
    justifyContent: 'center',
  },
  loginBtn: {
    width: "80%",
    backgroundColor: '#fb5b5a',
    borderRadius: 25,
    height: 200,
    alignItems: "center",
    justifyContent: "center",
    marginTop: 40,
    marginBottom: 10
  },
  title: {
    fontWeight: "bold",

```



```

        fontSize: 40,
        color: "#00abf0",
        marginBottom: 40,
    },
})

export default Role;

initPassanger.jsx/

import React from 'react';
import {View, SafeAreaView} from 'react-native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import Passengerlogin from './Passengerlogin';
import Passengersignup from './Passengersignup';
import Passenger from './Passenger';

const Stack = createNativeStackNavigator();

function InitPassenger(){
    return(
        <Stack.Navigator initialRouteName="Passengerlogin">
            <Stack.Screen component={Passengerlogin}
name="Passengerlogin" options={{ headerShown: false }}/>
            <Stack.Screen component={Passengersignup}
name="Passengersignup" options={{ headerShown: false }}/>
            <Stack.Screen component={Passenger} name="Passenger" options={{
headerShown: false }}/>
        </Stack.Navigator>
    )
}

```

```
);  
}
```

```
export default InitPassenger;
```

initDriver.jsx

```
import React from 'react';  
import {View, SafeAreaView} from 'react-native';  
import { createNativeStackNavigator } from '@react-navigation/native-stack';  
import Passengerlogin from './Passengerlogin';  
import Passengersignup from './Passengersignup';  
import Passenger from './Passenger';
```

```
const Stack = createNativeStackNavigator();
```

```
function InitPassenger(){  
  return(  
    <Stack.Navigator initialRouteName="Passengerlogin">  
      <Stack.Screen component={Passengerlogin}  
name="Passengerlogin" options={{ headerShown: false }}/>  
      <Stack.Screen component={Passengersignup}  
name="Passengersignup" options={{ headerShown: false }}/>  
      <Stack.Screen component={Passenger} name="Passenger" options={{  
headerShown: false }}/>  
    </Stack.Navigator>  
  )  
}
```

```
);  
}
```

```
export default InitPassenger;
```

```
Passneger.login.jsx/
```

```
import React, { useState, useContext } from 'react';
```

```
import {
```

```
  StyleSheet,
```

```
  Alert,
```

```
  Text,
```

```
  View,
```

```
  TextInput,
```

```
  TouchableOpacity,
```

```
} from 'react-native';
```

```
import axios from 'axios';
```

```
import { passengerloginRoute } from '../APIroutes';
```

```
//component
```

```
const Passengerlogin = function ({ navigation }) {
```

```
  const [User, setUser] = useState({
```

```
    email: "",
```

```
    password: "",
```

```
  });
```

```

const handleValidation = () => {
  const password = User.password;
  const email = User.email;
  if (password === "") {
    Alert.alert("Email and Password is required.");
    return false;
  } else if (email === "") {
    Alert.alert("Email and Password is required.");
    return false;
  }
  return true;
};

```

```

function navigating() {
  navigation.navigate('Passenger',
    {
      screen: 'Chooser',
      params: {
        CurrentUser: User.email
      }
    }
  );
}

```

```

async function gettheUser() {
  try {
    if (handleValidation()) {

```

```

    console.log(User)

    const useeer = await axios.post(passengerloginRoute, User);

    console.log(useeer.data, 'was the response status \n');

    if (useeer.data.status === false) {

        Alert.alert(` couldn't login`, 'the email or password is wrong');

    }

    if (useeer.data.status === true) {

        navigating();

    }

}

} catch (e) {

    console.log(e)

}

}

```

```

return (
    <View style={styles.container}>

        <Text style={styles.title} >RIDER LOG-IN</Text>

        <View style={styles.inputView}>

            <TextInput

                style={styles.inputText}

                placeholder="Email"

                placeholderTextColor="#003f5c"

                onChangeText={text => {

                    User.email = text

```

```

        setUser({ ...User })
      }} />
    </View>
    <View style={styles.inputView}>
      <TextInput
        style={styles.inputText}
        secureTextEntry
        placeholder="Password"
        placeholderTextColor="#003f5c"
        onChangeText={text => {
          User.password = text
          setUser({ ...User })
        }} />
    </View>

    <TouchableOpacity
      onPress={() => { gettheUser() }}
      style={styles.loginBtn}>
      <Text style={styles.loginText}>LOGIN </Text>
    </TouchableOpacity>
    <TouchableOpacity
      onPress={() => { navigation.navigate('Passengersignup') }}>
      <Text style={styles.forgotAndSignUpText}>No Account then
Signup</Text>
    </TouchableOpacity>
  </View>
);

```

```
}  
  
const styles = StyleSheet.create({  
  container: {  
    flex: 1,  
    backgroundColor: '#BBB',  
    alignItems: 'center',  
    justifyContent: 'center',  
  },  
  title: {  
    fontWeight: "bold",  
    fontSize: 50,  
    color: "#00abf0",  
    marginBottom: 40,  
  },  
  inputView: {  
    width: "80%",  
    backgroundColor: "#888",  
    borderRadius: 25,  
    height: 50,  
    marginBottom: 20,  
    justifyContent: "center",  
    padding: 20  
  },  
  inputText: {  
    height: 50,  
    color: "white"
```

```

    },
    forgotAndSignUpText: {
      color: "white",
      fontSize: 15
    },
    loginBtn: {
      width: "80%",
      backgroundColor: "#00abf0",
      borderRadius: 25,
      height: 50,
      alignItems: "center",
      justifyContent: "center",
      marginTop: 40,
      marginBottom: 10
    },
  },
});

export default Passengerlogin;

```

Passengersignup.jsx/

```

import React, { useState, useEffect ,useContext} from 'react';
import {
  StyleSheet,
  Text,
  View,
  TextInput,
  TouchableOpacity,
  Alert,

```



```

} from 'react-native';

import axios from 'axios';

import { passengersignupRoute } from '../APIroutes';

const Passengersignup = function ({ navigation }) {

  const [User, setUser] = useState({
    name: "",
    email: "",
    password: "",
  });

  const handleValidation= () => {
    const { username, email, password} = User;
    if (username === "") {
      Alert.alert("Email and Password is required.");
      return false;
    } else if (password === "") {
      Alert.alert("Email and Password is required.");
      return false;
    } else if(email === ""){
      Alert.alert("Email and Password is required.");
      return false;
    }
    return true;
  };

```

```

function navigating(){
    navigation.navigate('Passenger',
        {
            screen: 'Chooser',
            params: {
                CurrentUser: User.email
            }
        }
    );
}

```

```

async function settheUser() {
    console.log(User);
    if (handleValidation()) {
        const dataa = await axios.post(passengersignupRoute, User );

        console.log(dataa)
        if (dataa.data.status === false) {
            Alert.alert(`the emailid or password is already used`);
        }
        if( dataa.data.status === true) {
            navigating();
        }
    }
}

```

```

return (
  <View style={styles.container}>
    <Text style={styles.title}>RIDER SIGN-UP</Text>
    <View style={styles.inputView}>
      <TextInput
        style={styles.inputText}
        placeholder="name"
        placeholderTextColor="#003f5c0"
        onChangeText={text => {
          User.name = text
          setUser({ ...User })
        }} />
    </View>
    <View style={styles.inputView}>
      <TextInput
        style={styles.inputText}
        placeholder="Email"
        placeholderTextColor="#003f5c"
        onChangeText={text => {
          User.email = text
          setUser({ ...User })
        }} />
    </View>
    <View style={styles.inputView}>
      <TextInput

```

```

        style={styles.inputText}
        secureTextEntry
        placeholder="Password"
        placeholderTextColor="#003f5c"
        onChangeText={text => {
            User.password = text
            setUser({ ...User })
        }} />
    </View>

    <TouchableOpacity
        onPress={(e) => { settheUser() }}
        style={styles.loginBtn}>
        <Text style={styles.loginText}>SIGN-UP</Text>
    </TouchableOpacity>

    <TouchableOpacity
        onPress={() => { navigation.goBack('Passengerlogin') }}>
        <Text style={styles.forgotAndSignUpText}>Login Instead</Text>
    </TouchableOpacity>

</View>

);
}

const styles = StyleSheet.create({
    container: {
        flex: 1,
        backgroundColor: '#BBB',

```

```
        alignItems: 'center',
        justifyContent: 'center',
    },
    title: {
        fontWeight: "bold",
        fontSize: 50,
        color: "#00abf0",
        marginBottom: 40,
    },
    inputView: {
        width: "80%",
        backgroundColor: "#888",
        borderRadius: 25,
        height: 50,
        marginBottom: 20,
        justifyContent: "center",
        padding: 20
    },
    inputText: {
        height: 50,
        color: "white"
    },
    forgotAndSignUpText: {
        color: "white",
        fontSize: 15
    },
    },
```

```

loginBtn: {
  width: "80%",
  backgroundColor: "#00abfb",
  borderRadius: 25,
  height: 50,
  alignItems: "center",
  justifyContent: "center",
  marginTop: 40,
  marginBottom: 10
},
});
export default Passengersignup;

```

passenger.jsx/

```

import React, { useState } from 'react';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import Chooser from './Chooser';
import PassengerDetails from './PassengerDetails';
import PassengerMap from './PassengerMap';
import PassengerReview from './PassengerReview';
import Payments from './Payments';
import { NavigationContainer } from '@react-navigation/native';
import PassengerHome from './passengerHome';
import LoadingScreen from './loadingScreen';

```

```

const PassengerStack = createNativeStackNavigator();

function Passenger() {
  return (
    <PassengerStack.Navigator initialRouteName='Chooser'>

      <PassengerStack.Screen component={Chooser} name="Chooser"
options={{ headerShown: false }} />

      <PassengerStack.Screen component={PassengerHome}
name="PassengerHome" option={{ headerShown: false }} />

      <PassengerStack.Screen component={PassengerReview}
name="PassengerReview" options={{ headerShown: false }} />

      <PassengerStack.Screen component={Payments} name="Payments"
options={{ headerShown: false }} />

      <PassengerStack.Screen component={LoadingScreen}
name='LoadingScreen' options={{ headerShown: false }} />

    </PassengerStack.Navigator>

  );
}

export default Passenger;

```

Choose.jsx /

```

import React, { useState, useContext,useEffect } from 'react';

import { View, Text, TextInput, Alert, StyleSheet, TouchableOpacity, FlatList } from
'react-native';

import { MAPBOX_API_KEY } from '../config';

import axios from 'axios';

import { ChooserRoute } from '../APIroutes';

import { CurrentUserContext } from '../ParamContext';


const searchPlaces = async (query) => {

  try {

    const response = await axios.get(

      `https://api.mapbox.com/geocoding/v5/mapbox.places/${query}.json?acc
ess_token=${MAPBOX_API_KEY}`

    );

    if (response.data.features && response.data.features.length > 0) {

      return response.data.features;

    } else {

      return [];

    }

  } catch (error) {

    console.error('Error searching places:', error);

    return [];

  }

};

```



```

const Chooser = function ({ route,navigation }) {

  const [pickup, setpickup] = useState([]);
  const [Destination, setDestination] = useState([]);

  const [searchQueryPick, setSearchQueryPick] = useState("");
  const [searchQueryDest, setsearchQueryDest] = useState("");
  const [searchResultsPick, setSearchResultsPick] = useState([]);
  const [searchResultsDest, setSearchResultsDest] = useState([]);

  const handleSearchPickup = async () => {
    const results = await searchPlaces(searchQueryPick);

    const thepoint = results.map((ding) => {
      const coord = ding['center']
      const placename = ding['place_name']
      return { coord, placename };
    })
    setSearchResultsPick(thepoint);
    console.log('the pickup list \n', thepoint)
  };

  const handleSearchDest = async () => {
    const results = await searchPlaces(searchQueryDest);
    const thepoint = results.map((ding) => {
      const coord = ding['center']
      const placename = ding['place_name']

```

```

        return { coord, placename };
    })
    setSearchResultsDest(thepoint);
    console.log('the destination list \n', thepoint)

};

function rideidgen(){
    let d = Date.now();
    return d;
}

function navigating(rideid){
    console.log('navi ridedid',rideid," ding ",route.params.CurrentUser);
    const ridew = rideid;
    const cc = route.params.CurrentUser;
    console.log('navi ridedid',rideid," ding ",cc);
    navigation.navigate('LoadingScreen',{

        CurrentUser: route.params.CurrentUser,
        RangoRideId:ridew

    });
}

const RideInitiator = async () => {

```

```

    console.log(pickup," ",Destination);
    const rideid = rideidgen();
    console.log('the ride generted now',rideid)
    const ridinator = await axios.post(ChooserRoute,{

        pickup,
        Destination,
        rideremail:'abhinesh@gmail.com',
        rideid,

    });
    console.log(ridinator.data);
    if(ridinator.data.status=== false){
        Alert.alert(` couldn't process the data try again`,`choose pickup and
destination`)
    }
    if(ridinator.data.status === true){
        navigating(rideid);
    }
}

return (
    <View style={styles.container}>

        <Text style={styles.title}>Choose PickUp and Destination</Text>
        <View style={styles.place}>
            <View style={styles.inputView}>

```

```

<TextInput
  placeholder="Search for Pickup Location"
  value={searchQueryPick}
  onChangeText={(text) => setSearchQueryPick(text)}
  onBlur={handleSearchPickup}
  style={styles.inputText}
/>
</View>
<View>{
  searchResultsPick.map((searchResultsPick, index) => {
    return (
      <TouchableOpacity
        key={index}
        onPress={() =>
{setpickup(searchResultsPick.coord);console.log(searchResultsPick.placename) }}>

          <Text
style={styles.loginText}>{searchResultsPick.placename}</Text>

        </TouchableOpacity>)
    )
  }
</View>
</View>
<View style={styles.place}>
  <View style={styles.inputView}>
    <TextInput
      placeholder="Search for Destination"

```

```

        value={searchQueryDest}

        onChangeText={(text) => setSearchQueryDest(text)}

        onBlur={handleSearchDest}

        style={styles.inputText}

    />
</View>
<View>
{
    searchResultsDest.map((searchResultsDest, index) => {
        return (
            <TouchableOpacity
                key={index}
                onPress={() =>
                    {setDestination(searchResultsDest.coord);console.log(searchResultsDest.placename)
                }}>

                <Text
                    style={styles.loginText}>{searchResultsDest.placename}</Text>

                </TouchableOpacity>)

        })
    }
</View>
</View>

<TouchableOpacity
    onPress={() => { RideInitiator()}}
    style={styles.loginBtn}>

```

```

        <Text style={styles.loginText}>Let's Ride</Text>
      </TouchableOpacity>
    </View>
  );
}

```

```

const styles = StyleSheet.create({
  container: {
    alignItems: 'center',
    backgroundColor: '#BBB',
    justifyContent: 'center',

  },
  place: {
    width: '100%',
    alignItems: 'center',
    height: '30%',
    justifyContent: 'center',
  },
  ,
  inputText: {
    height: 50,
    color: "white"
  },
  title: {
    fontWeight: "bold",

```

```
    fontSize: 27,  
    color: "#fb5b5a",  
    marginBottom: 80,  
    marginTop: 40,  
  },  
  loginBtn: {  
    width: "80%",  
    backgroundColor: "#00abf0",  
    borderRadius: 25,  
    height: 50,  
    alignItems: "center",  
    justifyContent: "center",  
    marginTop: 40,  
    marginBottom: 10  
  },  
  inputView: {  
    width: "100%",  
    backgroundColor: "#888",  
    borderRadius: 25,  
    height: 50,  
    marginBottom: 20,  
    justifyContent: "center",  
    padding: 20,  
    fontSize: 26,  
  },  
});
```

```
export default Chooser;
```

loadingScreen.jsx/

```
import React, { useState, useEffect } from 'react';
import { View, Text, StyleSheet } from 'react-native';
import axios from 'axios';
import { getRiderWait } from '../APIroutes';
import { useSharedParams } from '../ParamContext';
```

```
const LoadingScreen = ({ route, navigation }) => {
```

```
  const { setSharedParams } = useSharedParams();
```

```
  useEffect(() => {
```

```
    const getRide = async () => {
```

```
      try {
```

```
        console.log(route.params.RangoRideId)
```

```
        const a = route.params.RangoRideId;
```

```
        const response = await axios.get(getRiderWait,{
          params:{ rideid:a,}
```

```
        });
```

```
        const result = response.data;
```

```
        console.log(result)
```

```
        if (result.status === true) {
```



```

clearInterval(interval);

setSharedParams({
    CurrentUser: route.params.CurrentUser,
    RangoRideId: route.params.RangoRideId,
});

navigation.navigate('PassengerHome', {
    screen: 'PassengerDetails',
});
}

}catch(error){
    console.error('Error fetching data',error)
}

};

const interval = setInterval(getRide,5000);

return () => clearInterval(interval);
},[]);

```

```

return (
    <View style={styles.container}>
        <Text>Waiting for a Driver</Text>
    </View>

```

```

    );
};

const styles = StyleSheet.create({
  container: {
    alignItems: 'center',
    justifyContent: 'center',
    backgroundColor: '#BBB',
  },
});

```

```
export default LoadingScreen;
```

PassengerHome.jsx/

```

import React from 'react';
import {createBottomTabNavigator} from '@react-navigation/bottom-tabs';
import { NavigationContainer } from '@react-navigation/native';
import PassengerMap from './PassengerMap';
import PassengerDetails from './PassengerDetails';
import Eotp from './Eotp';
const passengerTab = createBottomTabNavigator();

function PassengerHome(){

  return(

```

```

    <passengerTab.Navigator>

      <passengerTab.Screen component={PassengerDetails}
name="PassengerDetails" options={{ headerShown: false }} />

      <passengerTab.Screen component={PassengerMap}
name="PassengerMap" options={{ headerShown: false }}/>

      <passengerTab.Screen component={Eotp} name="Eotp" options={{
headerShown: false }}/>

    </passengerTab.Navigator>

  );
}

export default PassengerHome;

```

PassengerDetails.jsx/

```

import React, { useState, useEffect } from 'react';

import { View, Text, StyleSheet, TouchableOpacity, ActivityIndicator } from 'react-
native';

import axios from 'axios';

import { RiderDetailsRoute } from '../APIroutes';

import { MAPBOX_API_KEY } from '../config';

import { useSharedParams } from '../ParamContext';

const PassengerDetails = function ({ route }) {

  const { sharedParams } = useSharedParams();

  const Currentuser = sharedParams.CurrentUser;

```

```

const Rangorideid = sharedParams.RangoRideid;

const [pcoords, setpcoords] = useState("");
const [dcoords, setdcoords] = useState("");
const [distance, setdistance] = useState("");
const [otp, setotp] = useState("");
const [driveremail, setdriveremail] = useState("");
const [gotten, changegotten] = useState(false);

useEffect(() => {
    getData();
}, []);

function geocoding(thecoord1, thecoord2) {
    console.log(thecoord1, ' ', thecoord2)

    const names =
    axios.get(`https://api.mapbox.com/geocoding/v5/mapbox.places/${thecoord1},${t
hecoord2}.json?access_token=${MAPBOX_API_KEY}`)

    const placename = names.data.features;

    const ding = placename.map((name) => {
        if (name['center'][0] == thecoord1 && name['center'][1] == thecoord2) {
            return name['place_name']
        }
        else {
            return 'bling';
        }
    });
}

```

```

let actualname;
for (const value of ding) {
  if (value !== 'bling') {
    actualname = value;
    break;
  }
}

console.log('the name', actualname)
return actualname;
}

async function getData() {
  try {
    console.log(Rangorideid);
    const a = Rangorideid;
    console.log(a)
    const response = await axios.get(RiderDetailsRoute, {
      params: {
        rideid: a,
      }
    });

    const result = response.data;
    console.log(result.status);
    if (result.status === true) {
      const dest = geocoding(...result.datas.destinationLocation)
      console.log(dest)
    }
  }
}

```

```

        setdcoords(dest);

        const pick = geocoding(...result.datas.pickupLocation)

        console.log(pick)

        setpcoords(pick);

        setotp(result.datas.sotp);

        setdriveremail(result.datas.driverEmail);

        changegotten(true);

        console.log(dcoords, pcoords, otp, driveremail, gotten)

    }

    } catch (err) {

        console.log('asfdasdfsdf', err);

    }

}

return (

    <View style={styles.container}>

        {gotten ? (

            <View>

                <View style={styles.card}>

                    <Text style={styles.texter}>From : destination location{ }</Text>

                    <Text style={styles.texter} >To : destination location goes here{

}</Text>

                    <Text style={styles.texter}>Distance : distance goes here{ }</Text>

                </View>

                <View style={styles.pcard}>

                    <Text style={styles.texter}>Start OTP : {otp}</Text>

                </View>


```

```

    <View style={styles.card}>
      <Text style={styles.texter}>Driver Email : {driveremail}</Text>

    </View>

    <View>
      <TouchableOpacity
        style={styles.loginBtn}>
        <Text style={styles.loginText}>CANCEL THE RIDE</Text>
      </TouchableOpacity>
    </View>
  </View>
) : (
  <View style={styles.container}>
    <Text>
      loading the details....
    </Text>
  </View>
  )}
</View>

)
}

```

```

const styles = StyleSheet.create({
  container: {

```

```
        backgroundColor: '#BBB',
        alignItems: 'center',
        justifyContent: 'flex-start',
    },
    card: {
        backgroundColor: '#fff',
        marginTop: 25,
        marginBottom: 25,
        height: '29%',
        width: '90%',
        borderRadius: 10,
        padding: 15,
    },
    pcard: {
        backgroundColor: '#fff',
        height: '15%',
        borderRadius: 10,
        padding: 15,
        width: '90%',
    },
    texter: {
        fontSize: 20,
        padding: 5,
    },
    loginBtn: {
        width: "100%",
```



```

        backgroundColor: "#00abf0",
        borderRadius: 25,
        height: 70,
        alignItems: "center",
        justifyContent: "center",
        padding: 15,
    },
});

```

```
export default PassengerDetails;
```

PassengerMap.jsx/

```

import React from 'react';
import { StyleSheet, View } from 'react-native';
import Mapbox from '@rnmapbox/maps';
import { MAPBOX_API_KEY } from '../config';
import { useSharedParams } from '../ParamContext';

```

```
Mapbox.setAccessToken(MAPBOX_API_KEY);
```

```

const PassengerMap = () => {
  const { sharedParams } = useSharedParams();
  const Currentuser = sharedParams.CurrentUser;
  const Rangorideid = sharedParams.RangoRideId;

```

```

return (
  <View style={styles.page}>
    <View style={styles.container}>
      <Mapbox.MapView style={styles.map} />
    </View>
  </View>
);
}

```

```

export default PassengerMap;

```

```

const styles = StyleSheet.create({
  page: {
    flex: 1,
    alignItems: 'center',
  },
  container: {
    height: '90%',
    width: '90%',
  },
  map: {
    flex: 1
  }
});

```

Eotp.jsx/

```
import React,{useState} from 'react';

import { View, Text, TextInput, TouchableOpacity, StyleSheet ,Alert} from 'react-
native';

import axios from 'axios';

import { EndRideOTP } from '../APIroutes';

import { useSharedParams } from '../ParamContext';

const Eotp = function ({navigation}) {

  const { sharedParams } = useSharedParams();

  const Currentuser = sharedParams.CurrentUser;

  const Rangorideid = sharedParams.RangoRideId;

  const [otp, setotp] = useState(Number);

  async function EndRide() {

    try {

      console.log('ding ding')

      const response = await axios.post(EndRideOTP,{

        rangoid:Rangorideid,

        stotp:otp,

      })

      if(response.data.status=== true){

        Alert.alert("the ride has Ended");

        navigation.navigate('Payments');
```

```

    }

    if(response.data.status === false){
        Alert.alert('Wrong otp',"retry with correct otp")
    }

} catch (er) {
    console.error(er)
}
}

return (
    <View style={styles.container}>
        <Text style={styles.title}>Enter Your End OTP</Text>
        <View style={styles.inputView}>
            <TextInput
                style={styles.inputText}
                placeholder="Enter the EOTP"
                placeholderTextColor="#003f5c"
                onChangeText={text => {
                    setotp(text)
                }}
            />
        </View>
        <TouchableOpacity
            onPress={()=>{

```

```

        EndRide();
    }}
    style={styles.loginBtn}>
    <Text style={styles.loginText}>END THE RIDE</Text>
  </TouchableOpacity>
</View>);
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#BBB',
    alignItems: 'center',
  },
  title: {
    fontWeight: "bold",
    fontSize: 25,
    color: "#fb5b5a",
    marginBottom: 40,
    marginTop: 40,
  },
  inputView: {
    width: "80%",
    backgroundColor: "#888",
    borderRadius: 25,
    height: 50,
    marginBottom: 20,
  },
});

```

```
      justifyContent: "center",
      padding: 20
    },
    inputText: {
      height: 60,
      color: "white"
    },
    loginBtn: {
      width: "80%",
      backgroundColor: "#fb5b5a",
      borderRadius: 25,
      height: 70,
      alignItems: "center",
      justifyContent: "center",
      marginTop: 40,
      marginBottom: 10
    },
  },
});

export default Eotp;
```

initDriver.jsx/

```
import React from 'react';
import {View, SafeAreaView} from 'react-native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import Driver from './Driver';
import Driverlogin from './Driverlogin';
import Driversignup from './Driversignup';

const Stack = createNativeStackNavigator();

function InitDriver(){
  return(
    <Stack.Navigator initialRouteName='Driverlogin'>
      <Stack.Screen component={Driverlogin} name="Driverlogin" options={{
headerShown: false }}/>
      <Stack.Screen component={Driversignup} name="Driversignup" options={{
headerShown: false }}/>
      <Stack.Screen component={Driver} name="Driver" options={{ headerShown:
false }}/>
    </Stack.Navigator>
  );
}

export default InitDriver;
```

Driverlogin.jsx./

```
import React, { useState, useEffect } from 'react';
import {
  StyleSheet,
  Alert,
  Text,
  View,
  TextInput,
  TouchableOpacity,
} from 'react-native';
import axios from 'axios';
import { driverloginRoute } from '../APIRoutes';

const Driverlogin = function ({ navigation }) {
  const [Driver, setDriver] = useState({ email: '', password: '' });

  const handleValidation = () => {
    const { email, password } = Driver;
    if (password === '') {
      Alert.alert("Email and Password is required.");
      return false;
    } else if (email === '') {
      Alert.alert("Email and Password is required.");
      return false;
    }
  }
}
```



```

        return true;
    };

function navigating(){
    console.log(Driver.email)
    navigation.navigate('Driver', {
        screen: 'AvailRide',
        params: {
            CurrentUser: Driver.email,
        }
    });
}

async function gettheDriver(){
    try{
        if(handleValidation()){
            console.log(Driver.email)
            const useeer = await axios.post(driverloginRoute,Driver)
            console.log(useeer.data.status,useeer.data.msg , 'was the response
status \n' );
            if(useeer.data.status=== false){
                Alert.alert(` couldn't login `);
            }
            if(useeer.data.status === true){

                navigating();
            }
        }
    }
}

```

```

    }
  }catch(e){
    console.log(e);
  }
}

return (
  <View style={styles.container}>
    <Text style={styles.title}>DRIVER LOG-IN</Text>
    <View style={styles.inputView}>
      <TextInput
        style={styles.inputText}
        placeholder="Email"
        placeholderTextColor="#003f5c"
        onChangeText={text => {
          Driver.email = text
          setDriver({ ...Driver })
        }} />
    </View>
    <View style={styles.inputView}>
      <TextInput
        style={styles.inputText}
        secureTextEntry
        placeholder="Password"
        placeholderTextColor="#003f5c"
        onChangeText={text => {

```

```

        Driver.password = text

        setDriver({ ...Driver })

    }} />

</View>

<TouchableOpacity
    onPress={() => {gettheDriver()}}
    style={styles.loginBtn}>
    <Text style={styles.loginText}>LOGIN </Text>
</TouchableOpacity>

<TouchableOpacity
    onPress={() => { navigation.navigate('Driversignup') }}>
    <Text style={styles.forgotAndSignUpText}>No Account then
Signup</Text>
</TouchableOpacity>
</View>

);
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#BBB',
    alignItems: 'center',
    justifyContent: 'center',
  },
  title: {
    fontWeight: "bold",

```

```
    fontSize: 50,
    color: "#00abf0",
    marginBottom: 40,
  },
  inputView: {
    width: "80%",
    backgroundColor: "#888",
    borderRadius: 25,
    height: 50,
    marginBottom: 20,
    justifyContent: "center",
    padding: 20
  },
  inputText: {
    height: 50,
    color: "white"
  },
  forgotAndSignUpText: {
    color: "white",
    fontSize: 15
  },
  loginBtn: {
    width: "80%",
    backgroundColor: "#00abf0",
    borderRadius: 25,
    height: 50,
```

```

        alignItems: "center",
        justifyContent: "center",
        marginTop: 40,
        marginBottom: 10
    },
});
export default Driverlogin;

```

Driversignup.jsx/

```

import React, { useState, useEffect } from 'react';
import {
    StyleSheet,
    Alert,
    Text,
    View,
    TextInput,
    TouchableOpacity,
} from 'react-native';
import axios from 'axios';
import { driverloginRoute } from '../APIroutes';

const Driverlogin = function ({ navigation }) {
    const [Driver, setDriver] = useState({ email: "", password: ""});

    const handleValidation= () => {

```

```

const { email, password} = Driver;

if (password === "") {

    Alert.alert("Email and Password is required.");

    return false;

} else if(email === ""){

    Alert.alert("Email and Password is required.");

    return false;

}

return true;

};

```

```

function navigating(){

    console.log(Driver.email)

    navigation.navigate('Driver', {

        screen: 'AvailRide',

        params: {

            CurrentUser: Driver.email,

        }

    });

}

```

```

async function gettheDriver(){

    try{

        if(handleValidation()){

            console.log(Driver.email)

            const useer = await axios.post(driverloginRoute,Driver)

```

```
        console.log(useeer.data.status,useeer.data.msg , 'was the response  
status \n' );
```

```
        if(useeer.data.status=== false){
```

```
            Alert.alert(` couldn't login `);
```

```
        }
```

```
        if(useeer.data.status === true){
```

```
            navigating();
```

```
        }
```

```
    }
```

```
    }catch(e){
```

```
        console.log(e);
```

```
    }
```

```
}
```

```
return (
```

```
    <View style={styles.container}>
```

```
        <Text style={styles.title}>DRIVER LOG-IN</Text>
```

```
        <View style={styles.inputView}>
```

```
            <TextInput
```

```
                style={styles.inputText}
```

```
                placeholder="Email"
```

```
                placeholderTextColor="#003f5c"
```

```
                onChangeText={text => {
```

```
                    Driver.email = text
```

```
                    setDriver({ ...Driver })
```

```
                }} />
```

```

</View>

<View style={styles.inputView}>
  <TextInput
    style={styles.inputText}
    secureTextEntry
    placeholder="Password"
    placeholderTextColor="#003f5c"
    onChangeText={text => {
      Driver.password = text
      setDriver({ ...Driver })
    }} />
</View>

<TouchableOpacity
  onPress={() => {gettheDriver()}}
  style={styles.loginBtn}>
  <Text style={styles.loginText}>LOGIN </Text>
</TouchableOpacity>

<TouchableOpacity
  onPress={() => { navigation.navigate('Driversignup') }}>
  <Text style={styles.forgotAndSignUpText}>No Account then
Signup</Text>
</TouchableOpacity>
</View>

);
}

const styles = StyleSheet.create({

```



```
container: {  
  flex: 1,  
  backgroundColor: '#BBB',  
  alignItems: 'center',  
  justifyContent: 'center',  
},  
title: {  
  fontWeight: "bold",  
  fontSize: 50,  
  color: "#00abf0",  
  marginBottom: 40,  
},  
inputView: {  
  width: "80%",  
  backgroundColor: "#888",  
  borderRadius: 25,  
  height: 50,  
  marginBottom: 20,  
  justifyContent: "center",  
  padding: 20  
},  
inputText: {  
  height: 50,  
  color: "white"  
},  
forgotAndSignUpText: {
```

```

        color: "white",
        fontSize: 15
    },
    loginBtn: {
        width: "80%",
        backgroundColor: "#00abf0",
        borderRadius: 25,
        height: 50,
        alignItems: "center",
        justifyContent: "center",
        marginTop: 40,
        marginBottom: 10
    },
});
export default Driverlogin;

```

Driver.jsx/

```

import React, { useState, useEffect } from 'react';
import {
    StyleSheet,
    Alert,
    Text,
    View,
    TextInput,
    TouchableOpacity,

```

```

} from 'react-native';

import axios from 'axios';

import { driverloginRoute } from '../APIroutes';

const Driverlogin = function ({ navigation }) {

  const [Driver, setDriver] = useState({ email: "", password: ""});

  const handleValidation= () => {

    const { email, password} = Driver;

    if (password === "") {

      Alert.alert("Email and Password is required.");

      return false;

    } else if(email === ""){

      Alert.alert("Email and Password is required.");

      return false;

    }

    return true;

  };

  function navigating(){

    console.log(Driver.email)

    navigation.navigate('Driver', {

      screen: 'AvailRide',

      params: {

        CurrentUser: Driver.email,

      }

    })
  }

```

```

    });
}

async function gettheDriver(){
    try{
        if(handleValidation()){
            console.log(Driver.email)

            const useeer = await axios.post(driverloginRoute,Driver)

            console.log(useeer.data.status,useeer.data.msg , 'was the response
status \n' );

            if(useeer.data.status=== false){
                Alert.alert(` couldn't login `);
            }

            if(useeer.data.status === true){

                navigating();
            }
        }
    }catch(e){
        console.log(e);
    }
}

return (
    <View style={styles.container}>
        <Text style={styles.title}>DRIVER LOG-IN</Text>
        <View style={styles.inputView}>

```

```

<TextInput
  style={styles.inputText}
  placeholder="Email"
  placeholderTextColor="#003f5c"
  onChangeText={text => {
    Driver.email = text
    setDriver({ ...Driver })
  }} />
</View>
<View style={styles.inputView}>
  <TextInput
    style={styles.inputText}
    secureTextEntry
    placeholder="Password"
    placeholderTextColor="#003f5c"
    onChangeText={text => {
      Driver.password = text
      setDriver({ ...Driver })
    }} />
</View>

<TouchableOpacity
  onPress={() => {gettheDriver()}}
  style={styles.loginBtn}>
  <Text style={styles.loginText}>LOGIN </Text>
</TouchableOpacity>

```

```

    <TouchableOpacity
      onPress={() => { navigation.navigate('Driversignup') }}>
      <Text style={styles.forgotAndSignUpText}>No Account then
Signup</Text>
    </TouchableOpacity>
  </View>
);
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#BBB',
    alignItems: 'center',
    justifyContent: 'center',
  },
  title: {
    fontWeight: "bold",
    fontSize: 50,
    color: "#00abf0",
    marginBottom: 40,
  },
  inputView: {
    width: "80%",
    backgroundColor: "#888",
    borderRadius: 25,
    height: 50,
    marginBottom: 20,
  },

```

```

        justifyContent: "center",
        padding: 20
    },
    inputText: {
        height: 50,
        color: "white"
    },
    forgotAndSignUpText: {
        color: "white",
        fontSize: 15
    },
    loginBtn: {
        width: "80%",
        backgroundColor: "#00abf0",
        borderRadius: 25,
        height: 50,
        alignItems: "center",
        justifyContent: "center",
        marginTop: 40,
        marginBottom: 10
    },
});

export default Driverlogin;

```

AvailRide.jsx/

```
import React, { useState, useEffect } from 'react';
import { View, Button, Text, ScrollView, StyleSheet } from 'react-native';
import { AvailRideRoute } from '../APIroutes';
import { ChosenDriverRoute } from '../APIroutes';
import axios from 'axios';
import { MAPBOX_API_KEY } from '../config';
import { useSharedParams } from '../ParamContext';

const AvailRide = function ({ route, navigation }) {

  const { setSharedParams } = useSharedParams();

  const [gotRider, setGotRider] = useState([]);
  const [gotten, changegotten] = useState(false);
  useEffect(() => {
    getRide();
  }, []);

  async function geocoding(thecoord1, thecoord2) {
    console.log(thecoord1, ' ', thecoord2)

    const names = await
    axios.get(`https://api.mapbox.com/geocoding/v5/mapbox.places/${thecoord1},${t
hecoord2}.json?access_token=${MAPBOX_API_KEY}`)

    const placename = names.data.features;

    const ding = placename.map((name) => {
```



```

    if (name['center'][0] == thecoord1 && name['center'][1] == thecoord2) {
        return name['place_name']
    }
    else {
        return 'bling';
    }
});

let actualname;
for (const value of ding) {
    if (value !== 'bling') {
        actualname = value;
        break;
    }
}

console.log('the name', actualname)
return actualname;
}

```

```

const getRide = async () => {
    try {

        const response = await axios.get(AvailRideRoute);
        console.log(response.data, ' asdfsdf ')
        if (response.data.status === true) {
            const theARides = (response.data.Available).map((zing) => {
                console.log(zing['rangold']);
            });

```

```

        const rideId = zing['rangold'];
        const dest = zing['destinationLocation'];
        const Dest = geocoding(...(zing['destinationLocation']))
        const pick = zing['pickupLocation'];
        const Pick = geocoding(...(zing['pickupLocation']));
        return { rideId, Dest, Pick };
    })
    setGotRider(theARides);
    changegotten(true);
    console.log(gotRider)
  }
} catch (error) {
  console.error(error);
}
}

```

```

function navigating(rid) {
  console.log('ging',rid)
  setSharedParams({
    CurrentUser: route.params.CurrentUser,
    RangoRideId: rid,
  });

  navigation.navigate('DriverHome',
    {
      screen: 'DriverDetails',

```

```

    });
}

const setRide = async (thisride) => {
  try {
    console.log(thisride, "\n", thisride.rideId);
    const response = await axios.post(ChoosenDriverRoute, {
      rangoid: thisride.rideId,
      driveremail: route.params.CurrentUser,
    });
    if (response.data.status === true) {
      navigating(thisride.rideId)
    }
  } catch (error) {
    console.error(error);
  }
}

return (

  <ScrollView style={styles.container}>
    {
      gotRider.map((AvailableRides, index) => {
        return (
          <View style={styles.card} key={index} onPress={() => {
'DriverHome' }}>

```

```

        <Text>Pickup = {AvailableRides.pick} </Text>

        <Text>Destination = {AvailableRides.dest}</Text>

        <Button title={"i am ready for this ride"} onPress={() => {
setRide(AvailableRides) }} />

        </View>

    )

})

}

</ScrollView>

);
}

```

```

const styles = StyleSheet.create({
  container: {

    backgroundColor: '#BBB',

  },
  card: {
    backgroundColor: '#fff',
    marginTop: 25,
    marginBottom: 25,
    height: 100,

```

```

        width: '100%',
        borderRadius: 10,
        padding: 15,
    },
  });
export default AvailRide;

```

DriverHome.jsx/

```

import React from 'react';
import {createBottomTabNavigator} from '@react-navigation/bottom-tabs';
import { NavigationContainer } from '@react-navigation/native';
import DriverMap from './DriverMap';
import DriverDetails from './DriverDetails';
import Sotp from './Sotp';
const passengerTab = createBottomTabNavigator();

function PassengerHome(){

  return(

    <passengerTab.Navigator>

      <passengerTab.Screen component={DriverDetails}
name="DriverDetails" options={{ headerShown: false }} />

      <passengerTab.Screen component={DriverMap}
name="DriverMap" options={{ headerShown: false }} />

```

```
        <passengerTab.Screen component={Sotp} name="Sotp" options={{
headerShown: false }}/>
```

```
    </passengerTab.Navigator>
```

```
  );
```

```
}
```

```
export default PassengerHome;
```

```
DriverDetails.jsx/
```

```
import React, { useState, useEffect } from 'react';
```

```
import {View,Button,Text,StyleSheet,TouchableOpacity} from 'react-native';
```

```
import { DriverDetailsRoute } from '../APIRoutes';
```

```
import { MAPBOX_API_KEY } from '../config';
```

```
import axios from 'axios';
```

```
import { useSharedParams } from '../ParamContext';
```

```
const DriverDetails = function({route,navigation}){
```

```
    const { sharedParams } = useSharedParams();
```

```
    const Currentuser = sharedParams.CurrentUser;
```

```
    const Rangorideid = sharedParams.RangoRideId;
```

```
    const [pcoords, setpcoords] = useState("");
```

```
    const [dcoords, setdcoords] = useState("");
```

```

const [distance, setdistance] = useState("");
const [otp, setotp] = useState("");
const [rideremail, setrideremail] = useState("");
const [gotten, changegotten] = useState(false);

useEffect(() => {
  getData();
}, []);

function geocoding(thecoord1, thecoord2) {
  console.log(thecoord1, ' ', thecoord2)

  const names =
    axios.get(`https://api.mapbox.com/geocoding/v5/mapbox.places/${thecoord1},${t
hecoord2}.json?access_token=${MAPBOX_API_KEY}`)

  const placename = names.data.features;
  const ding = placename.map((name) => {
    if (name['center'][0] == thecoord1 && name['center'][1] == thecoord2) {
      return name['place_name']
    }
    else {
      return 'bling';
    }
  });

  let actualname;
  for (const value of ding) {
    if (value != 'bling') {
      actualname = value;
    }
  }
}

```

```

        break;
    }
}
console.log('the name', actualname)
return actualname;
}

```

```

async function getData() {
  try {
    console.log('ding ding',Rangorideid);
    const a = Rangorideid;
    console.log(a)
    const response = await axios.get(DriverDetailsRoute, {
      params: {
        rideid: a,
      }
    });
    const result = response.data;
    console.log(result.status);
    if (result.status === true) {
      const dest = geocoding(...result.datas.destinationLocation)
      console.log(dest)
      setdcoords(dest);
      const pick = geocoding(...result.datas.pickupLocation)
      console.log(pick)
      setpcoords(pick);
    }
  }
}

```



```

        setotp(result.datas.sotp);
        setdriveremail(result.datas.driverEmail);
        changegotten(true);
        console.log(dcoords, pcoords, otp, driveremail, gotten)
    }
} catch (err) {
    console.log('asfdasdfsdf', err);
}
}

return(
    <View style={styles.container}>
        {gotten? (
            <View>
                <View style={styles.card}>
                    <Text style={styles.texter}>From : Pickup loacation goes here{</Text>
                    <Text style={styles.texter} >To : Destination location goes
here{</Text>
                    <Text style={styles.texter}>Distance : Distance gose here{</Text>
                </View>
                <View style={styles.pcard}>
                    <Text style={styles.texter}>Price : price goes here</Text>
                    <Text style={styles.texter}>Start OTP : goes here</Text>
                </View>
                <View style={styles.card}>
                    <Text style={styles.texter}>Passenger Name : Passenger name goes
here</Text>

```

```
        <Text style={styles.texter}>Passenger phone NUmber : phone number  
goes here</Text>
```

```
    </View>
```

```
    <View>
```

```
        <TouchableOpacity
```

```
        style={styles.loginBtn}>
```

```
        <Text style={styles.loginText}>CANCEL THE RIDE</Text>
```

```
    </TouchableOpacity>
```

```
    </View>
```

```
  </View>
```

```
):( <View style={styles.container}>
```

```
    <Text>
```

```
        loading the details....
```

```
    </Text>
```

```
  </View>
```

```
  )}
```

```
</View>
```

```
);
```

```
}
```

```
const styles = StyleSheet.create({
```

```
  container: {
```

```
    flex: 1,
```

```
    backgroundColor: '#BBB',
```

```
    alignItems: 'center',
```

```
    justifyContent: 'flex-start',
```

```
    color: "#000",
```

```
},
card:{
  backgroundColor:'#fff',
  marginTop:25,
  marginBottom:25,
  height:'29%',
  width:'90%',
  borderRadius:10,
  padding:15,
},
pcard:{
  backgroundColor:'#fff',
  height:'15%',
  borderRadius:10,
  padding:15,
  width:'90%',
},
texter:{
  fontSize:20,
  padding:5,
  color:"#000",
},
loginBtn: {
  width: "100%",
  backgroundColor: "#fb5b5a",
  borderRadius: 25,
```

```
    height: 70,  
    alignItems: "center",  
    justifyContent: "center",  
    padding: 15,  
  },  
});
```

```
export default DriverDetails;
```

```
DriverMap.jsx/
```

```
import React, { useState, useEffect } from 'react';  
import { StyleSheet, View, Text, Button } from 'react-native';  
import MapboxGL, { Logger } from '@rnmapbox/maps';  
import Geolocation from '@react-native-community/geolocation';  
import axios from 'axios';  
import { MAPBOX_API_KEY } from '../config';  
import { useSharedParams } from '../ParamContext';
```

```
Logger.setLogCallback((log) => {
```

```
  const { message } = log;  
  if (  
    message.match('Request failed due to a permanent error: Canceled') ||  
    message.match('Request failed due to a permanent error: Socket closed')  
  ) {
```

```

    return true;
  }
  return false;
});
MapboxGL.setAccessToken(MAPBOX_API_KEY);
{/*MapboxGL.setConnected(true); */ }
MapboxGL.setTelemetryEnabled(false);
MapboxGL.setWellKnownTileServer('Mapbox');
Geolocation.setRNConfiguration({
  skipPermissionRequests: false,
  authorizationLevel: 'auto',
});

const DriverMap = () => {

  const { sharedParams } = useSharedParams();
  const Currentuser = sharedParams.CurrentUser;
  const Rangorideid = sharedParams.RangoRideId;

  const [coords, setcoords] = useState([]);
  const [destinationCoords, setdestinationCoords] = useState([]);
  {/* const [distance, setdistance] = useState(null);
  const [time, settime] = useState(null); */}
  const [routeDirection, setrouteDirection] = useState(null);

  setdestinationCoords([76.6394, 12.2958]);

```

```

async function getPermissionLocation() {
  try {
    const geo = await Geolocation.getCurrentPosition(
      location => setcoords([location.coords.longitude, location.coords.latitude]),
      err => console.log(err),
      { enableHighAccuracy: true },
    );
  } catch (err) {
    console.error('error getting location', error);
  }
}

```

```

useEffect(() => {
  getPermissionLocation();
  console.log(coords);
},coords);

```

```

function makeRouterFeature(long,lat) {
  let routerFeature = {
    type: 'FeatureCollection',
    features: [
      {
        type: 'Feature',
        properties: {},
        geometry: {

```

```

    type: 'LineString',
    coordinates: ` ${[[long,lat]]} `,
  },
},
],
};
return routerFeature;
}

```

```

async function createRouterLine(coords1,coords2) {
  const startCoords = ` ${[[coords1,coords2]]} `;
  const endCoords = ` ${[[destinationCoords[0],destinationCoords[1]]]} `;
  const geometries = 'geojson';
  const url =
`https://api.mapbox.com/directions/v5/mapbox/driving/${startCoords};${endCoords}?alternatives=true&geometries=${geometries}&access_token=${MAPBOX_API_KEY}`;

```

```

try {
  let response = await axios.get(url);
  let json = await response.data;
  console.log(json);
  const data = json.routes.map((data) => {
    console.log(data);
  });
  let coordinates = json['routes'][0]['geometry']['coordinates'];
  if (coordinates.length) {

```

```

    const routerFeature = makeRouterFeature(coordinates);

    setrouteDirection(routerFeature)
  }
} catch (err) {
  console.log(err);
}
}

return (
  <View style={styles.container}>
    <MapboxGL.MapView>
      style={styles.map}
      zoomEnabled={true}
      styleURL = 'mapbox://styles/mapbox/navigation-day-v1'
      rotateEnabled={true}
      onDidFinishLoadingMap={async () => {
        await createRouterLine([...coords]);
      }}
    <MapboxGL.Camera
      zoomLevel={7}
      centerCoordinate={[12.9716, 77.5946]}
      pitch={60}
      animationMode={'flyTo'}
      animationDuration={2000}
    />
    {routeDirection && (

```



```

    <MapboxGL.ShapeSource
      id='line1'
      shape={routeDirection}
    >

    <MapboxGL.LineLayer
      id='routerLine01'
      style={{
        lineColor: '#fa9314',
        lineWidth: 4
      }}
    />

  </MapboxGL.ShapeSource>
)}

  <MapboxGL.UserLocation
    animated={true}
    androidRenderMode='gps'
    showsUserHeadingIndicator={true}
  />

</MapboxGL.MapView>
</View>

);
};

const styles = StyleSheet.create({
  container: {

```

```
    flex: 1,  
  },  
  map:{  
    flex:1,  
  }  
});
```

```
export default DriverMap;
```

```
sotp.jsx/
```

```
import React, { useState } from 'react';  
import { View, Text, TextInput, TouchableOpacity, StyleSheet,Alert } from 'react-  
native';  
import axios from 'axios';  
import { StartRideOTP } from '../APIroutes';  
import { useSharedParams } from '../ParamContext';  
  
const Sotp = function ({route,navigation}) {  
  
  const { sharedParams } = useSharedParams();  
  const Currentuser = sharedParams.CurrentUser;  
  const Rangorideid = sharedParams.RangoRideld;  
  
  const [otp, setotp] = useState(Number);
```

```

async function StartRide() {
  try {
    console.log('ding ding')

    const response = await axios.post(StartRideOTP,{
      rangoid:Rangorideid,
      stotp:otp,
    })

    if(response.data.status=== true){
      navigation.navigate('DriverMap');
    }

    if(response.data.status === false){
      Alert.alert('Wrong otp',"retry with correct otp")
    }

  } catch (er) {
    console.error(er)
  }
}

return (
  <View style={styles.container}>
    <Text style={styles.title}>Enter Your End OTP</Text>
    <View style={styles.inputView}>
      <TextInput
        style={styles.inputText}
        placeholder="Enter the EOTP"

```

```

        placeholderTextColor="#003f5c"
        onChangeText={text => {
            setotp(text)
        }}
    />
</View>
<TouchableOpacity
    onPress={() => {
        StartRide()
    }}
    style={styles.loginBtn}>
    <Text style={styles.loginText}>START THE RIDE</Text>
</TouchableOpacity>
</View>);
}

const styles = StyleSheet.create({
    container: {
        flex: 1,
        backgroundColor: '#BBB',
        alignItems: 'center',
    },
    title: {
        fontWeight: "bold",
        fontSize: 25,
        color: "#fb5b5a",
        marginBottom: 40,
    },

```

```

        marginTop: 40,
    },
    inputView: {
        width: "80%",
        backgroundColor: "#888",
        borderRadius: 25,
        height: 50,
        marginBottom: 20,
        justifyContent: "center",
        padding: 20
    },
    inputText: {
        height: 60,
        color: "white"
    },
    loginBtn: {
        width: "80%",
        backgroundColor: "#fb5b5a",
        borderRadius: 25,
        height: 70,
        alignItems: "center",
        justifyContent: "center",
        marginTop: 40,
        marginBottom: 10
    },
}); export default Sotp;

```

4.2 Back-end Source Code

Index.js

```
const express = require("express");
const mongoose = require('mongoose');
const {DBConnect} = require('./DBConnect');
const DB_URL = process.env.MONGO_URL || `mongodb://127.0.0.1:27017/`;

const app = express();
app.use(express.json());

//DB

DBConnect(DB_URL);

// the routes
const riderauthRoutes = require('./routes/riderRoutes');
const driverauthRoutes = require('./routes/driverRoutes');
const rangoRiderRoutes = require('./routes/rangoRiderRoutes');
const rangoDriverRoutes = require('./routes/rangoDriverRoutes')

//-----

app.use('/riderauth',riderauthRoutes);
```

```
app.use('/driverauth',driverauthRoutes);
app.use('/rangoRider',rangoRiderRoutes);
app.use('/rangoDriver',rangoDriverRoutes);

app.listen(5000, () => {
  console.log(`Server is running on port 5000.`);
});
```

DBconnect.js

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

module.exports.DBConnect = async (DB_URL)=>{
  try{
    const dbOptions ={
      dbName:'Rango',
    }
    await mongoose.connect(DB_URL,dbOptions);
    console.log("the database was connected");
  }catch(err){
    console.log(err);
  }
}
```

driverRoutes.js

```
const {  
  driverlogin,  
  driversignup,  
} = require("../controllers/authController");  
  
const router = require("express").Router();  
  
router.post("/driverlogin",driverlogin);  
router.post("/driversignup",driversignup);  
  
module.exports = router;
```

rideRoutes.js

```
const {  
  passengerlogin,  
  passengersignup,  
} = require("../controllers/authController");  
  
const router = require("express").Router();
```



```
router.post("/passengerlogin",passengerlogin);  
router.post("/passengersignup",passengersignup);
```

```
module.exports = router;
```

```
rangoDriverRoutes.js
```

```
const {  
  AvailableRides,  
  DriverChoose,  
  DriverDetails,  
  DriverLiveLocation,  
  StartRideOTP  
} = require("../controllers/rideControllers");  
  
const router = require('express').Router();  
  
router.get('/AvailableRides',AvailableRides);  
router.post('/DriverChoose',DriverChoose);  
router.get('/DriverDetails',DriverDetails);  
router.put('/DriverLiveLocation',DriverLiveLocation);  
router.post('/StartRideOTP',StartRideOTP);  
  
module.exports = router;
```

rangoRiderRoutes.js

```
const {  
  Chooser,  
  RiderDetails,  
  TrackDriver,  
  EndRideOTP,  
  RiderWait,  
} = require("../controllers/rideControllers");  
  
const router = require('express').Router();  
  
router.post('/Chooser', Chooser);  
router.get('/getRiderWait', RiderWait)  
router.get('/RiderDetails', RiderDetails);  
router.get('/TrackDriver', TrackDriver);  
router.post('/EndRideOTP', EndRideOTP);  
  
module.exports = router;
```

driverModel.js

```
const mongoose = require("mongoose");

const driverSchema = new mongoose.Schema({
  driverName: {
    type: String,
    required: true,
    min: 3,
    max: 20,
    unique: true,
  },
  driverEmail: {
    type: String,
    required: true,
    unique: true,
    max: 50,
  },
  driverPassword: {
    type: String,
    required: true,
    min: 8,
    unique: true,
  },
});
```

```
module.exports = mongoose.model("Driver", driverSchema);
```

userModel.js

```
const mongoose = require("mongoose");
```

```
const riderSchema = new mongoose.Schema({
```

```
  riderName: {
```

```
    type: String,
```

```
    required: true,
```

```
    min: 3,
```

```
    max: 20,
```

```
    unique: true,
```

```
  },
```

```
  riderEmail: {
```

```
    type: String,
```

```
    required: true,
```

```
    unique: true,
```

```
    max: 50,
```

```
  },
```

```
  riderPassword: {
```

```
    type: String,
```

```
    required: true,
```

```
    min: 8,
```

```
    unique: true,
```

```
  },
```

```
});
```

```
module.exports = mongoose.model("Rider", riderSchema);
```

ridetourModel.js

```
const mongoose = require("mongoose");
```

```
const rideSchema = new mongoose.Schema({
```

```
  riderEmail: {
```

```
    type: String,
```

```
  },
```

```
  driverEmail: {
```

```
    type: String,
```

```
  },
```

```
  rangold: {
```

```
    type: String,
```

```
    required: true,
```

```
    unique:true,
```

```
  },
```

```
  rideStart: {
```

```
    type: Boolean,
```

```
  },
```

```
  rideEnd: {
```

```
    type: Boolean,
```

```

    },
    sotp: {
      type: Number,
      max: 9999,
      min: 0,
    },
    eotp: {
      type: Number,
      min: 0,
      max: 9999,
    },
    pickupLocation: {
      type: Array,
      required: true,
    },
    destinationLocation: {
      type: Array,
      required: true,
    },
  },
});

module.exports = mongoose.model("RideTour", rideSchema);

```

authController.js

```
const Rider = require("../models/userModel");
const Driver = require("../models/driverModel");
const bcrypt = require('bcrypt');

module.exports.driverlogin = async (req, res, next) => {
  try {
    const driverEmail = req.body.email;
    const driverPassword = req.body.password;
    console.log(driverEmail)
    console.log(driverPassword)
    const user = await Driver.findOne({ driverEmail });
    if (!user)
      return res.json({ msg: "Incorrect Username or Password", status: false });
    const isPasswordValid = await Driver.findOne({driverPassword})
    if (!isPasswordValid)
      return res.json({ msg: "Incorrect Username or Password", status: false });
    delete Driver.driverPassword;
    return res.json({ status: true, user });
  } catch (ex) {
    next(ex);
  }
};
```

```

module.exports.driversignup = async (req, res, next) => {
  try {

    const driverName = req.body.name;
    const driverEmail = req.body.email;
    const driverPassword = req.body.password;
    console.log(req.body)
    const Driveer = await Driver.create({
      driverName,
      driverEmail,
      driverPassword,
    });
    console.log(Driveer);
    delete Driveer.password;
    return res.json({ status: true, Driveer });
  } catch (err) {
    return res.json({status:false})
    next(err)
  }
};

```

```

module.exports.passengerlogin = async (req, res, next) => {
  try {
    const riderEmail = req.body.email;
    const riderPassword = req.body.password;
    console.log('emailid',riderEmail)

```



```

    console.log('password',riderPassword)

    const user = await Rider.findOne({ riderEmail });

    if (!user)

        return res.json({ msg: "Incorrect Username or Password", status: false });

    const isPasswordValid = await Rider.findOne({riderPassword})

    if (!isPasswordValid)

        return res.json({ msg: "Incorrect Username or Password", status: false });

    delete Rider.riderPassword;

    return res.json({ status: true, user });

} catch (ex) {

    next(ex);

}

}

```

```

module.exports.passengersignup = async (req, res, next) => {

    try {

        const riderName = req.body.name;

        const riderEmail = req.body.email;

        const riderPassword = req.body.password;

        console.log('asdfsfd',req.body)

        console.log(riderName)


        const user = await Rider.create({

            riderName,

            riderEmail,

            riderPassword,

```

```

    });

    delete Rider.riderPassword;

    return res.json({ status: true, user });

  } catch (err) {

    console.log(err)

    return res.json({status:false})

    next(err)

  }

};

```

rideController.js

```

const RideTour = require("../models/ridetourModel");
const otpGenetator = require("otp-generator");

module.exports.Chooser = async (req, res, next) => {
  try {
    console.log(req.body);
    console.log('ding ding');
    const pickup = req.body.pickup;
    const destination = req.body.Destination;
    const rideremail = req.body.rideremail;
    const riderid = req.body.rideid;
    console.log(pickup, " ", destination, " ", rideremail, " ", riderid)
  }
};

```

```

const startotp = otpGenetator.generate(4, {
  digits: true,
  upperCaseAlphabets: false,
  lowerCaseAlphabets: false,
  specialChars: false,
});

const endotp = otpGenetator.generate(4, {
  digits: true,
  upperCaseAlphabets: false,
  lowerCaseAlphabets: false,
  specialChars: false,
});

console.log(startotp, " ", endotp)

```

```

const tour = await RideTour.create({
  riderEmail: rideremail,
  driverEmail: "nil",
  rangold: riderid,
  rideStart: false,
  rideEnd: false,
  sotp: startotp,
  eotp: endotp,
  pickupLocation: pickup,
  destinationLocation: destination,
});

```

```

    res.json({ status: true, tour })

  } catch (err) {

    res.json({ status: false });

    console.log(err)

    next(err);

  }
}

module.exports.RiderWait = async (req, res, next) => {
  try {
    console.log(req.query)

    const data = req.query.rideid;

    console.log("the loading Screen", req.query.rideid)

    const selected = await RideTour.where("rangold").equals(data).findOne({
driverEmail: { $ne: 'nil' } })

    console.log('zing', selected)

    if (!selected) {
      res.json({ status: false });
    }

    if (selected) {
      const d = await RideTour.findOne({ driverEmail: { $eq: 'nil' } })

      if (d) {
        res.json({ status: true, selected })
      }

      if (!d) {

```

```

        res.json({ status: false })
      }
    }
  } catch (err) {
    console.log(err)
    next(err);
  }
}

```

```

module.exports.RiderDetails = async (req, res, next) => {
  try {
    const randoid = req.query.rideid;
    console.log('ddd', randoid)
    const datas = await RideTour.findOne({ rangold: { $eq: randoid } })

    if (datas) {
      res.json({ status: true, datas });
    }
    if (!datas) {
      res.json({ status: false, datas });
    }

  } catch (err) {
    console.log('RiderDetailsRouter: ', err)
  }
}

```

```

module.exports.TrackDriver = async (req, res, next) => {
  try {
    return res.json({
      ding: 'driversignup',
      num: 23,
    })
    console.log('ding ding');
  } catch (err) {
    console.log('TrackDriverRouter: ', err)
  }
}

```

```

module.exports.EndRideOTP = async (req, res, next) => {
  try {
    console.log(req.body)
    const randoid = req.body.rangoid;
    const otp = req.body.stotp;
    const started = await
    RideTour.findOne({rangold:{ $eq:randoid}}).where('eotp').equals(otp);
    console.log(started)
    if(started){
      await RideTour.updateOne({ rangold: { $eq: randoid } }, { $set:
      {rideEnd:true } });
      res.json({status:true,started})
    }
    if(!started){

```

```

        res.json({status:false,started})
    }
} catch (err) {
    console.log('StartRideOTPRouter: ', err)
}
}

module.exports.AvailableRides = async (req, res, next) => {
    try {
        const Available = await RideTour.find({ rideStart: { $ne: true } });
        if (Available) {

            res.json({ status: true, Available })
            console.log('kling')
        }
        if (!Available) {
            res.json({ status: false });
            console.log('dunk')
        }

        console.log('ding ding');
    } catch (err) {
        res.json({ status: false })
        next(err)
    }
}
}

```

```

module.exports.DriverChoose = async (req, res, next) => {
  try {
    console.log(req.body)

    const randold = req.body.rangoid;
    const drivermaily = req.body.driveremail
    console.log(randold, drivermaily)

    const Rideset = await RideTour.updateOne({ rangold: { $eq: randold } }, { $set: {
driverEmail: drivermaily } });

    console.log(Rideset)

    if(Rideset.modifiedCount == 1 || Rideset.matchedCount == 1){

      res.json({status:true});
    }

    if(Rideset.modifiedCount == 0){
      res.json({status:true});
    }
  } catch (err) {
    console.log('DriverChoosenRouter: ', err)
    next(err)
  }
}

```

```

module.exports.DriverDetails = async (req, res, next) => {
  try {

```



```

const randoid = req.query.rideid;

console.log('ddd', randoid)

const datas = await RideTour.findOne({ rangold: { $eq: randoid } })

if (datas) {
  res.json({ status: true, datas });
}

if (!datas) {
  res.json({ status: false, datas });
}

} catch (err) {
  console.log('RiderDetailsRouter: ', err)
}
}

module.exports.DriverLiveLocation = async (req, res, next) => {
  try {

    console.log('ding ding');

  } catch (err) {
    console.log('DriverLiveLocationRouter: ', err)
  }
}

module.exports.StartRideOTP = async (req, res, next) => {

```

```

try {
  console.log(req.body)
  const randoid = req.body.rangoid;
  const otp = req.body.stotp;
  const started = await
RideTour.findOne({rangold:{$eq:randoid}}).where('sotp').equals(otp);
  console.log(started)
  if(started){
    await RideTour.updateOne({ rangold: { $eq: randoid } }, { $set:
{rideStart:true } });
    res.json({status:true,started})
  }
  if(!started){
    res.json({status:false,started})
  }
} catch (err) {
  console.log('StartRideOTPRouter: ', err)
}
}

```

4.3. PROJECT IMPLEMENTATION

Starting the Backend Server:

```
PS C:\Users\Abhinesh\Desktop\repos\rango\rango-server> npm start

> rango-server@1.0.0 start
> nodemon index.js

[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Server is running on port 5000.
the database was connected
█
```

Starting the MongoDB database :

1. Run the ‘mongo’ command in the terminal to instantiated the mongoDB server which starts on "mongodb://127.0.0.1:27017”

```
PS C:\Users\Abhinesh\Desktop\repos\rango\rango-server> mongosh
Current Mongosh Log ID: 64ffd4b2757aeb38d0b85631
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.10.4
Using MongoDB:      6.0.8
Using Mongosh:       1.10.4

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

-----
  The server generated these startup warnings when booting
  2023-09-10T07:21:31.557+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----

test> use Rango
switched to db Rango
Rango> |
```

Starting the React App with Android Emulator

```
PS C:\Users\Abhinesh\Desktop\repos\rango\rango> npm run android
> Rango@0.0.1 android
> react-native run-android

info Starting JS server...
* daemon not running; starting now at tcp:5037
[ ] daemon started successfully
info launching emulator...
error Failed to launch emulator . Reason: It took too long to start and connect with Android emulator: Pixel_3a_API_34_extension_level_7_x86_64. You can try starting the emulator manually from the terminal with: C:\Users\Abhinesh\AppData\Local\Android\Sdk\emulator\emulator @Pixel_3a_API_34_extension_level_7_x86_64.
warn Please launch an emulator manually or connect a device. Otherwise app may fail to launch.
info Installing the app...

> Task :app:installDebug
Installing APK 'app-debug.apk' on 'Pixel_3a_API_34_extension_level_7_x86_64(AVD) - 14' for :app:debug
Installed on 1 device.

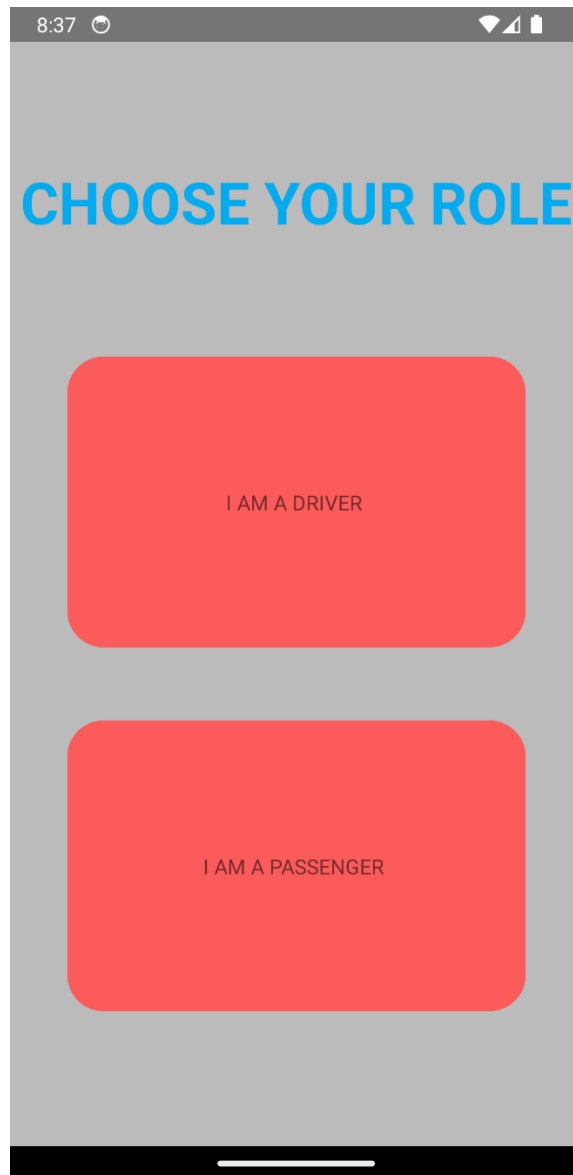
Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.
html#sec:command_line_warnings

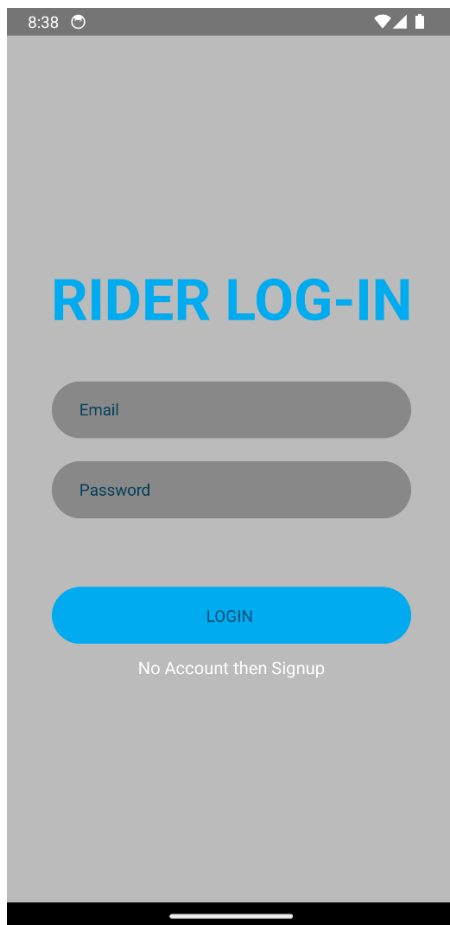
BUILD SUCCESSFUL in 49s
163 actionable tasks: 2 executed, 161 up-to-date
info Connecting to the development server...
info Starting the app...
Starting: Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] cmp=com.rango.MainActivity }
```

4.4. PROJECT SNAPSHOTS

Select Role Page:



User login and Driver login:



8:38

RIDER LOG-IN

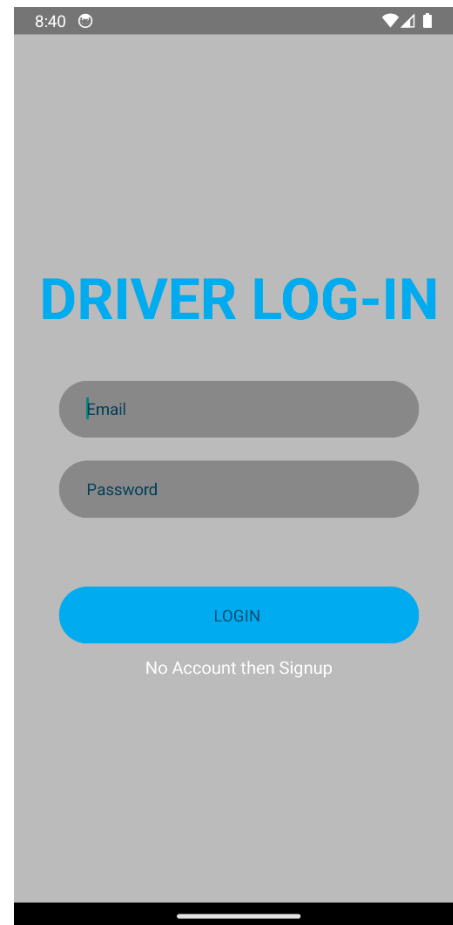
Email

Password

LOGIN

No Account then Signup

This is a mobile app screen for rider login. It features a grey background with a white status bar at the top showing the time 8:38. The title 'RIDER LOG-IN' is in large blue capital letters. Below the title are two rounded grey input fields for 'Email' and 'Password'. A blue rounded button labeled 'LOGIN' is centered below the fields. At the bottom, the text 'No Account then Signup' is displayed in a small grey font. A black home indicator bar is at the very bottom.



8:40

DRIVER LOG-IN

Email

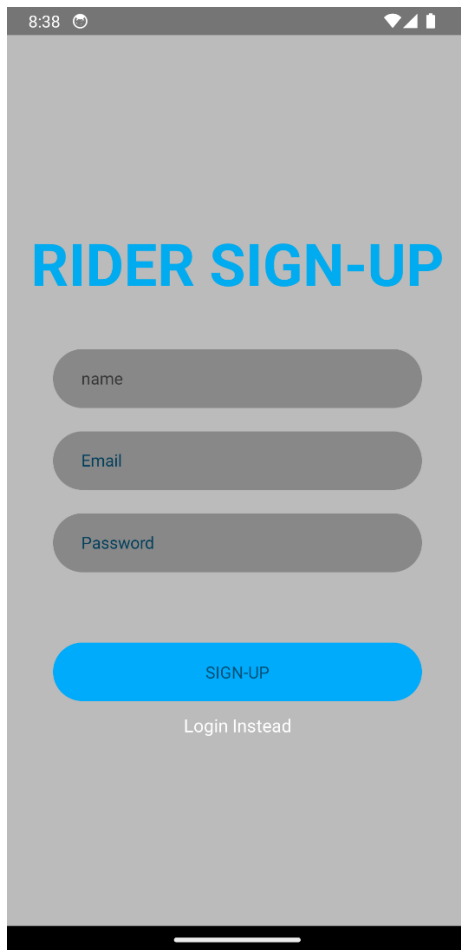
Password

LOGIN

No Account then Signup

This is a mobile app screen for driver login. It features a grey background with a white status bar at the top showing the time 8:40. The title 'DRIVER LOG-IN' is in large blue capital letters. Below the title are two rounded grey input fields for 'Email' and 'Password'. A blue rounded button labeled 'LOGIN' is centered below the fields. At the bottom, the text 'No Account then Signup' is displayed in a small grey font. A black home indicator bar is at the very bottom.

User signup and Driver signup



8:38

RIDER SIGN-UP

name

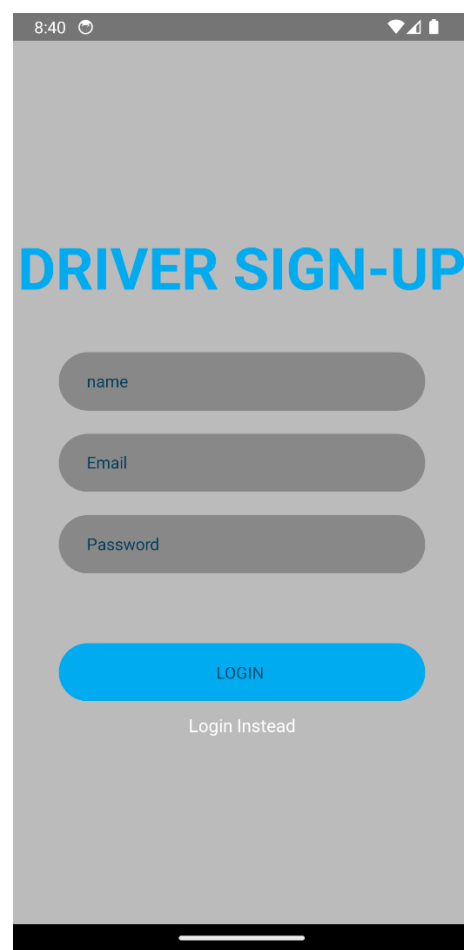
Email

Password

SIGN-UP

Login Instead

This is a mobile app screen for rider sign-up. It features a grey background with a blue title 'RIDER SIGN-UP'. Below the title are three rounded grey input fields labeled 'name', 'Email', and 'Password'. A blue rounded button labeled 'SIGN-UP' is positioned below the input fields. At the bottom, there is a link 'Login Instead' in a smaller font. The status bar at the top shows the time as 8:38 and standard mobile icons.



8:40

DRIVER SIGN-UP

name

Email

Password

LOGIN

Login Instead

This is a mobile app screen for driver sign-up. It features a grey background with a blue title 'DRIVER SIGN-UP'. Below the title are three rounded grey input fields labeled 'name', 'Email', and 'Password'. A blue rounded button labeled 'LOGIN' is positioned below the input fields. At the bottom, there is a link 'Login Instead' in a smaller font. The status bar at the top shows the time as 8:40 and standard mobile icons.

Places Search for Passenger:

8:39

Choose PickUp and Destination

malleshwaram

Malleshwaram, Bengaluru, Bengaluru Urban, Karnataka, India
Malleshwaram Club, 7th Cross, Bengaluru, Karnataka 560003, India
Malleshwaram Dose Corner, 17th cross, Bengaluru, Karnataka 560003, India
Malleshwaram Underpass, Dathathreya Temple, 560003, Bengaluru, Bengaluru Urban, Karnataka, India
Malleshwaram Market, Sampige Road, Bengaluru, Karnataka 560003, India

mg road bengaluru

Mahatma Gandhi Road, 560001, Ashok Nagar, Bengaluru, Bengaluru Urban, Karnataka, India
Mahatma Gandhi Road, Someshwarpura, 560008, Ulsoor, Bengaluru, Bengaluru Urban, Karnataka, India
MG Road, 205 rue Saint-Martin, Paris, 75003, France
MG Road, Puducherry, Puducherry 605001, India
MG Road, Mehrauli Gurgaon Road, Gurugram, Haryana 122002, India

Let's Ride

Choose Ride for Driver:

8:40

Pickup =
Destination =

I AM READY FOR THIS RIDE

Pickup =
Destination =

I AM READY FOR THIS RIDE

Pickup =
Destination =

I AM READY FOR THIS RIDE

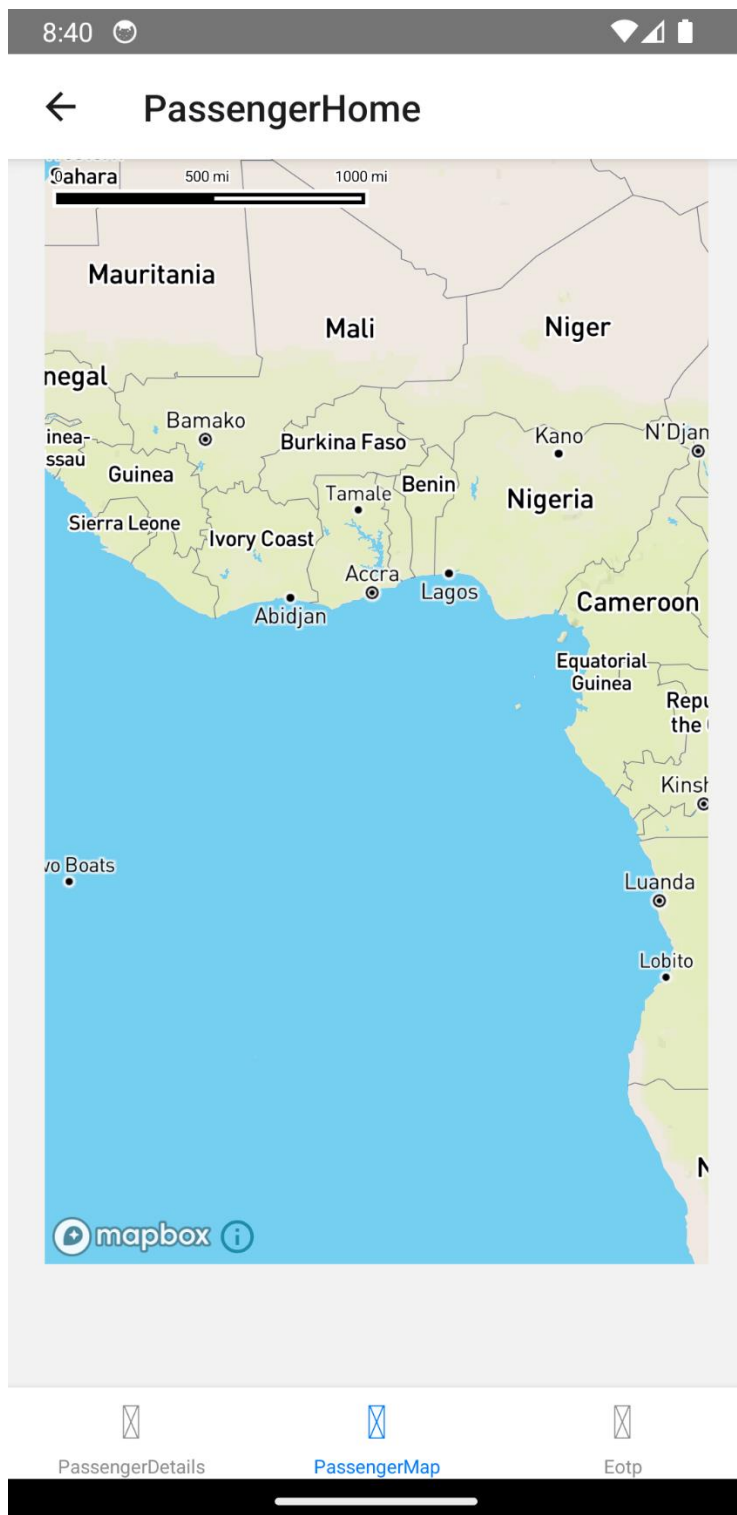
Pickup =
Destination =

I AM READY FOR THIS RIDE

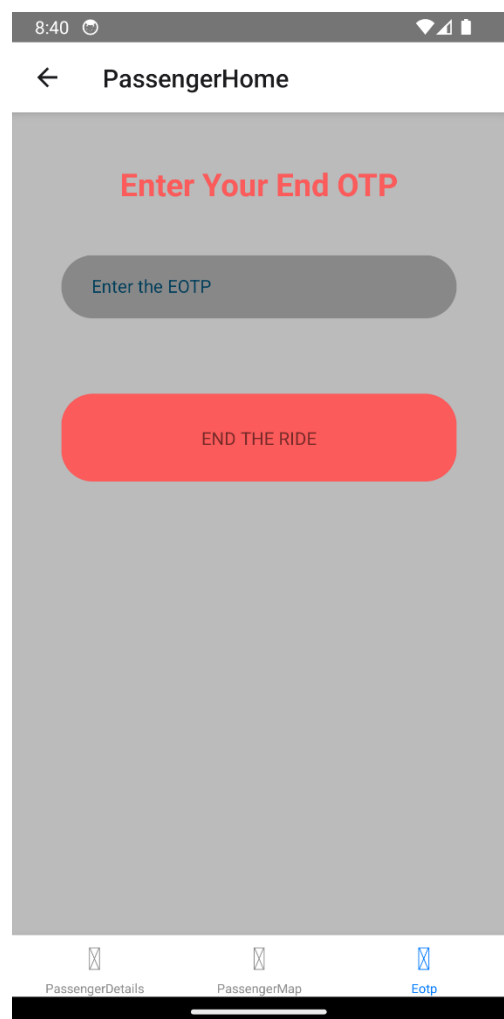
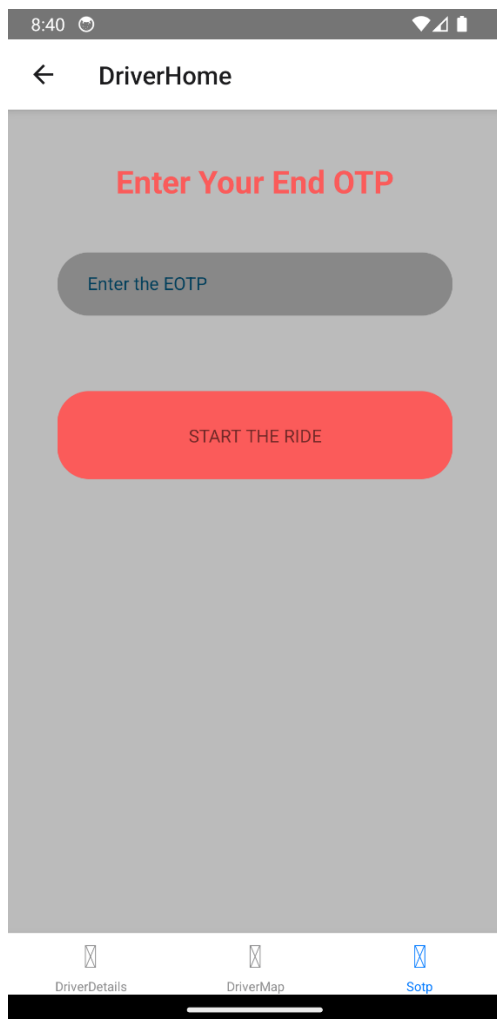
Pickup =
Destination =

I AM READY FOR THIS RIDE

Maps :



Otp entering for passenger and driver:



5. CONCLUSION

In conclusion, the React Native Ride-Hailing App represents a significant milestone in modern transportation solutions. This documentation has provided an in-depth insight into the development, features, and functionality of the app, aiming to empower developers, stakeholders, and users alike with the knowledge needed to understand and utilize this innovative platform.

Throughout this documentation, we have explored the primary modules and system requirements that make up this robust ride-hailing application. From user management and ride booking to driver management, navigation, and payment processing, every aspect has been meticulously designed to ensure a seamless and enjoyable user experience.

This app, built on the foundation of React Native and integrated with the powerful Mapbox Maps API, not only exemplifies the fusion of cutting-edge technologies but also encapsulates the spirit of convenience, safety, and efficiency that modern commuters demand.

As the ride-hailing landscape continues to evolve, we believe that this app has the potential to redefine the way people move around cities, bridging the gap between convenience and reliability. The journey from concept to creation has been one of dedication and innovation, and we look forward to seeing this app serve as a valuable tool for users and a source of opportunity for drivers.

In the ever-changing world of technology, the React Native Ride-Hailing App stands as a testament to what can be achieved when vision, creativity, and expertise come together. As this documentation concludes, we extend our gratitude to all those who have been part of this journey, and we eagerly anticipate the future possibilities that this application will bring to the world of urban mobility.

Future Enhancements:

1. **Multi-Language Support:** Implement multi-language support to cater to a global user base. Allow users to select their preferred language within the app.
2. **Scheduled Rides:** Add the ability for users to schedule rides in advance, making it convenient for planning trips to airports, appointments, or events.
3. **Carpooling and Ride Sharing:** Introduce carpooling options to encourage ride-sharing and reduce traffic congestion. Users can opt to share rides with others traveling in the same direction.
4. **Ride Discounts and Promotions:** Create a promotional system that offers discounts, promo codes, and referral rewards to both passengers and drivers to incentivize usage and growth.
5. **Loyalty Program:** Implement a loyalty program where frequent riders can earn points or rewards for using the service regularly.
6. **AI-Powered Predictive Features:** Use artificial intelligence to predict user preferences, such as preferred pick-up locations and frequently visited destinations, to expedite the booking process.
7. **Offline Mode:** Enable an offline mode that allows users to book rides and view trip details even when they have limited or no internet connectivity.
8. **Accessibility Features:** Improve accessibility by adding features for users with disabilities, such as voice commands, screen readers, and adjustable font sizes.
9. **Advanced Driver Tools:** Enhance driver tools and analytics, allowing drivers to view earnings breakdowns, track performance metrics, and access educational resources.
10. **Safety Features:** Continue to invest in safety features, such as real-time background checks for drivers and additional emergency services integration.
11. **Electric and Eco-Friendly Vehicles:** Promote the use of electric and eco-friendly vehicles by allowing users to select them as ride options, contributing to sustainability efforts.

6. REFERENCES

- [1] Marijn Haverbeke - “Eloquent JavaScript Third Edition .
- [2] <https://www.w3schools.com/REACT/DEFAULT.ASP>
- [3] <https://www.tutorialspoint.com/expressjs/index.htm>
- [4] <https://www.w3schools.com/javascript>
- [5] <https://www.w3schools.com/mongodb>
- [6] <https://reactnavigation.org/>
- [7] <https://reactnative.dev/>
- [8] <https://www.mapbox.com/>