```
import React, { Component, useState, useEffect } from 'react';
import {
    View,
    Text,
    StyleSheet,
    Dimensions,
    PermissionsAndroid,
    Platform,
    TouchableOpacity,
    Image
} from 'react-native';

import {
    GEOCODER_KEY
  } from 'react-native-dotenv'

import Toast from 'react-native-simple-toast';
import firebase from 'firebase';
import { CountDown } from 'react-native-countdown-component';
import { Stopwatch, Timer } from 'react-native-stopwatch-timer';
import humanize from 'humanize-plus';
import Geocoder from 'react-native-geocoding';
import ButtonComponent from '../components/ButtonComponent';
import Select2 from 'react-native-select-two';
import _ from 'lodash';

import { Icon } from 'react-native-elements';
import { getSunrise, getSunset } from 'sunrise-sunset-js'; //sunrise-sunset task

import { ScrollView, FlatList } from 'react-native-gesture-handler';
import JourneyOption from '../components/JourneyOption';

const haversine = require('haversine')

//global variables
var db_input = {}
var provLicence = false
var vehicleClass = null
var olderCar = false
var coverage = ''

//initiate geocoder api
// with env.GEOCODER_KEY
Geocoder.init(GEOCODER_KEY)

class TrackJourney extends Component {
    constructor(props) {
        super(props);
        this.state = {
            running: false,

            stopwatchStart: false,
            stopwatchReset: false,
            showCountdown: false,
            showStopwatch: false,
            timerStart: false,
            timerReset: false,

            distanceTravelled: 0,
```

```
            speed: 0,
            error: '',
            prevTimestamp: 0,
            prevCoords: {},
            journeyCost: 0,
            acceleration: 0,

            updatesEnabled: true,
            location: {},
            nightdrive: false,
            address: null,
            currentTime: 0,
            billing_month: '',

            vehiclelist: [],
            vehiclename: '',
            vehicletype: '',
            vehicleyear: 0,
            vehiclekey: '',
            hardBraking: false
        };
    }

    async componentDidMount() {
        this.hasLocationPermission()
        this.getLocation()

        //get database algorithm conditions
        await firebase.database().ref(`/algorithm/`).once('value')
            .then((snapshot) => {
                db_input.distance = snapshot.val().distance
                db_input.nightdrive_multiplier = snapshot.val().nightdrive_multiplier
                db_input.lightclass = snapshot.val().lightclass
                db_input.middleclass = snapshot.val().middleclass
                db_input.heavyclass = snapshot.val().heavyclass
                db_input.provisional_licence = snapshot.val().provisional_licence
                db_input.age_conditional = snapshot.val().age_conditional
                db_input.age_addition = snapshot.val().age_addition
                db_input.acceleration_conditional =
snapshot.val().acceleration_conditional
                db_input.hard_braking_penalty = snapshot.val().hard_braking_penalty
                db_input.coverage_1 = snapshot.val().coverage_1
                db_input.coverage_2 = snapshot.val().coverage_2
                db_input.coverage_3 = snapshot.val().coverage_3
            })
            .catch((error) => {
                console.log(error)
            })

        const { currentUser } = firebase.auth();

        //EVENT: firebase call
        //get user vehicle list
        firebase.database().ref(`/users/${currentUser.uid}/vehicles/`).on('value',
snapshot => {
            var vehicle_list = []
            snapshot.forEach((childSub) => {
                vehicle_list.push({id: childSub.key, name: childSub.val().name, type:
childSub.val().type, year: Number(childSub.val().year)})
```

```
            })
            this.setState({vehiclelist: vehicle_list})
        })

        //EVENT: firebase call
        //get licence variable
        //get coverage variable
        firebase.database().ref(`users/${currentUser.uid}/`).once('value').then(snapshot
=> {
            if(snapshot.val().licence == 'Provisional Licence'){
                provLicence = true
            }
            coverage = snapshot.val().coverage
        })
    }

    //run permission check
    hasLocationPermission = async () => {
        if (Platform.OS === 'ios' ||
            (Platform.OS === 'android' && Platform.Version < 23)) {
        return true;
        }

        const hasPermission = await PermissionsAndroid.check(
        PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION
        );

        //has permission already
        if (hasPermission) return true;

        //prompt permission request
        const status = await PermissionsAndroid.request(
        PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION
        );

        //permission granted
        if (status === PermissionsAndroid.RESULTS.GRANTED) return true;

        //permission denied
        if (status === PermissionsAndroid.RESULTS.DENIED) {
            Toast.show('Location permission denied by user.', Toast.LONG);
        } else if (status === PermissionsAndroid.RESULTS.NEVER_ASK_AGAIN) {
            Toast.show('Location permission revoked by user.', Toast.LONG);
        }

        return false;
    }

    //run one location update
    getLocation = async () => {
        const hasLocationPermission = await this.hasLocationPermission();

        if (!hasLocationPermission) return;

        this.setState({ loading: true }, () => {
          navigator.geolocation.getCurrentPosition(
            (position) => {
              this.setState({
                    location: position,
```

```
                    loading: false,
                    prevTimestamp: position.timestamp,
                    prevCoords: {
                        latitude: position.coords.latitude,
                        longitude: position.coords.longitude
                    }
                });
                //get journey starting address
                //stored in $address
              Geocoder.from(position.coords.latitude, position.coords.longitude)
                .then(json => {
                    const json_address = json.results[0].address_components
                    var address_component = json_address[3].long_name + ", " +
json_address[4].long_name
                    this.setState({
                        address: address_component
                    })
                })
                .catch(error => console.warn(error))
            },
            (error) => {
              this.setState({ location: error, loading: false });
              console.log(error);
            },
            { enableHighAccuracy: true, timeout: 15000, maximumAge: 10000,
distanceFilter: 50, forceRequestLocation: true }
          );
        });
    }

    //start stream - fetch location updates
    //trigger - geolocation.watchPosition
    getLocationUpdates = async () => {
        const hasLocationPermission = await this.hasLocationPermission();

        if (!hasLocationPermission) return;

        //on location change
        this.setState({ updatesEnabled: true }, () => {
          this.watchId = navigator.geolocation.watchPosition(
            (position) => {
              this.setState({ location: position });
              const { distanceTravelled, prevCoords, prevTimestamp, speed } = this.state

              const newCoords = {
                  latitude: position.coords.latitude,
                  longitude: position.coords.longitude
              }

              const distance = this.calcDistance(
                  prevCoords,
                  newCoords
              )

              const currentTimestamp = position.timestamp

              const previous_speed = speed

              const current_speed = this.calcSpeed(
```

```
                prevTimestamp,
                currentTimestamp,
                distance
            )

            const acceleration = this.calcAcceleration(previous_speed, current_speed,
currentTimestamp - prevTimestamp)

            this.setState({
                distanceTravelled: distanceTravelled + distance,
                prevCoords: newCoords,
                speed: current_speed,
                prevTimestamp: currentTimestamp,
                acceleration: acceleration
            })

            //update cost total
            this.calcCost(this.state.distanceTravelled, acceleration)
          },
          (error) => {
            this.setState({ location: error });
            console.log(error);
          },
          { enableHighAccuracy: true, distanceFilter: 5.0, interval: 1000,
fastestInterval: 2000 }
        );
      });
  }

  //stop location stream
  //trigger - geolocation.clearWatch
  removeLocationUpdates = () => {
      if (this.watchId !== null) {
          navigator.geolocation.clearWatch(this.watchId);
          this.setState({ updatesEnabled: false })
      }
  }

  //stopwatch and timer section
  getFormattedTime(time) {
      this.currentTime = time;
  };

  //start journey event
  startJourney() {
      const { prevCoords  } = this.state

      //start stopwatch
      //hide countdown
      this.setState({
          running: true,
          stopwatchStart: true,
          stopwatchReset: false,
          showCountdown: false,
      })

      //make single location call
      this.getLocation()
```

```
        //check driving time is unsafe
        const currentTime = new Date().getTime()
        if (currentTime > getSunset(prevCoords.latitude, prevCoords.longitude) ||
currentTime < getSunrise(prevCoords.latitude, prevCoords.longitude)) {
            this.setState({ nightdrive: true })
        }

        //assign selected vehicle to journey
        this.state.vehiclelist.forEach((child, index) => {
            if(child.checked == true){
                this.setState({ vehiclename: child.name, vehicletype: child.type,
vehicleyear: child.year })
            }
        })

        //assign vehicle type
        switch (this.state.vehicletype) {
            case 'SUV':
            case 'Pickup':
            case 'Minivan':
                vehicleClass = 'heavy_class'
                break;
            case 'Coupe':
            case 'Roadster':
            case 'Sedan':
                vehicleClass = 'middle_class'
                break;
            case 'Hatchback':
                vehicleClass = 'light_class'
                break;
            default:
                vehicleClass = 'error'
                break;
        }

        //check age of car against conditional
        var year = new Date().getFullYear()
        if((year - this.state.vehicleyear) > db_input.age_conditional){
            olderCar = true
        }

        //start location stream
        this.getLocationUpdates()
    }

    //end journey event
    endJourney() {
        //stop tracking location
        this.removeLocationUpdates()

        const duration = this.currentTime;
        const distance = Number(this.state.distanceTravelled).toFixed(3);
        const cost_total = Number(this.calcCost(distance, 0)); //get final cost
        const nightdrive = this.state.nightdrive
        const address = this.state.address
        const vehiclename = this.state.vehiclename
        const vehiclekey = this.state.vehiclekey

        const hours = new Date().getHours()
```

```
        const date = new Date().getDate()
        const month = new Date().getMonth()
        const year = new Date().getFullYear()

        //turn hour date month year into human friendly string
        const humanized_string = this.humanizeDate(hours, date, month);

        const billing_month = this.state.billing_month

        const date_string = `${date}/${month}/${year}`
        this.journeyCreate(address, distance, duration, cost_total, date_string,
humanized_string, nightdrive, vehiclename, vehiclekey, billing_month);

        //reset variables
        this.setState({
            distanceTravelled: 0,
            speed: 0,
            prevCoords: {},
            prevTimestamp: 0,
            running: false,
            stopwatchStart: false,
            stopwatchReset: true,
            speed: 0,
            nightdrive: false,
            updatesEnabled: false
        });
    }

    //update current cost state
    calcCost(distance, acceleration){
        //driving after sunset or before sunrise multiplier (percentage %)
        var nightdrive_addition = 0
        if (this.state.nightdrive) {
            nightdrive_addition = distance * db_input.nightdrive_multiplier/100
        }

        //vehicle type multiplier (percentage %)
        switch (vehicleClass) {
            case 'heavy_class':
                var vehicletype_addition = distance * (db_input.heavyclass / 100)
                break;
            case 'middle_class':
                var vehicletype_addition = distance * (db_input.middleclass / 100)
                break;
            case 'light_class':
                var vehicletype_addition = distance * (db_input.lightclass / 100)
                break;
            default:
                console.warn('ERROR: Vehicle type error in calculation')
                var vehicletype_addition = 0
                break;
        }

        //hard braking detection
        var hardBrakingPenalty = 0
        if(acceleration < (db_input.acceleration_conditional * -1) &&
!this.state.hardBraking){
            hardBrakingPenalty = db_input.hard_braking_penalty / 100
            this.setState({hardBraking: true})
```

```
            setTimeout(() => {
                this.setState({hardBraking: false})
            }, 5000);
        }

        //Car age addition
        var age_addition = 0
        if(olderCar){
            age_addition = distance * (db_input.age_addition / 100)
        }

        //Provisional License addition
        var licence_addition = 0
        if(provLicence){
            licence_addition = distance * (db_input.provisional_licence / 100)
        }

        //distance multiplier (cents/km)
        const distance_addition = distance * db_input.distance / 100

        var total = distance_addition + nightdrive_addition + vehicletype_addition +
licence_addition + age_addition + hardBrakingPenalty

        //multiply total by insurance plan
        switch (coverage) {
            case 'Third Party Insurance':
                total = total * db_input.coverage_1
                break;
            case 'Third Party Fire & Theft':
                total = total * db_input.coverage_2
                break;
            case 'Comprehensive':
                total = total * db_input.coverage_3
                break;
            default:
                console.warn('ERROR: Coverage type error in calculation')
                break;
        }

        //update current total
        this.setState({journeyCost: total})
        return total.toFixed(2)
    }

    //return spped in km/hr
    calcSpeed(prevTimestamp, newTimestamp, distance) {
        //get delta time in milliseconds
        const unix_time = new Date(newTimestamp).getTime() - new
Date(prevTimestamp).getTime()

        var speed = 0
        if(distance != 0){
            //speed = distance / time
            speed = distance / ((((unix_time)/1000)/60)/60) //convert to km/hr
        }

        return speed
    }
```

```
    //return distance in km
    calcDistance(start, end){
        //distance = difference in longitude and latitude
        //using haversine algorithm
        const distance = haversine(start, end, {unit: 'km'}) || 0
        return distance
    }

    //return acceleration in m/s^2
    calcAcceleration(v_0, v, t){
        //acceleration = v/seconds - v_0/seconds / time elapsed
        return ((v/3.6) - (v_0/3.6))/t
    }

    //EVENT: firebase call
    //push new journey to db
    journeyCreate(address, distance, duration, cost_total, date, humanized_date,
nightdrive, vehiclename, vehiclekey, billing_month){
        const data = {
            address: address,
            distance: distance,
            duration: duration,
            cost: cost_total,
            date: date,
            humanized_date: humanized_date,
            nightdrive: nightdrive,
            vehiclename: vehiclename,
            vehiclekey: vehiclekey,
            billing_month: billing_month
        }
        const { currentUser } = firebase.auth();
        console.log(data)
        firebase.database().ref(`users/${currentUser.uid}/journeys`)
            .push(data)
    }

    //return human firendly date string
    humanizeDate(hours, date, month){
        switch (hours > 12) {
            case true:
                hours = hours - 12 + "pm"
                break;
            case false:
                if(hours == 12){
                    hours = "12pm"
                    break;
                }

                if(hours == 0){
                    hours = "12am"
                    break;
                }

                hours = hours + "am"
            default:
                break;
        }

        date = humanize.ordinal(date)
```

```
    switch (month) {
        case 0:
            month = "Jan"
            break;
        case 1:
            month = "Feb"
            break;
        case 2:
            month = "Mar"
            break;
        case 3:
            month = "Apr"
            break;
        case 4:
            month = "May"
            break;
        case 5:
            month = "Jun"
            break;
        case 6:
            month = "Jul"
            break;
        case 7:
            month = "Aug"
            break;
        case 8:
            month = "Sep"
            break;
        case 9:
            month = "Oct"
            break;
        case 10:
            month = "Nov"
            break;
        case 11:
            month = "Dec"
            break;
        default:
            break;
    }
    this.setState({billing_month: month})
    return `${date} ${month} ${hours}`
}

//3 2 1 countdown section
countdown() {
    this.setState({showCountdown: true});
}

//cancel journey
cancelPressed() {
    this.setState({
        showCountdown: false
    })
}

round(n) {
    if (!n) {
```

```
          return 0;
        }

        return Math.floor(n * 100) / 100;
      }

    render() {
        const { hardBraking } = this.state
        return(
            <View style={styles.container}>
                <View style={styles.contentContainer}>
                    <Text
                        ref={(info) => this.errorMessage = info}
                    />
                    {this.state.showCountdown ? (
                        <View style={{flex: 1, justifyContent: 'space-around'}}>
                            <Image
                                    source={require('../../assets/key.png')}
                                    style={styles.image}
                                    alignSelf='center'
                                />
                            <Text style={styles.logo}>Beginning Journey</Text>
                            <CountDown
                                until={3}
                                size={50}
                                digitStyle={{backgroundColor: ''}}
                                onFinish={() => this.startJourney()}
                                digitTxtStyle={{color: '#2E6CB5'}}
                                timeToShow={['S']}
                                timeLabels={{s: ' '}}
                            />
                        </View>

                    ) : (
                        this.state.running ? (
                        <View style={styles.summary}>
                            <View style={styles.summaryHeaderContainer}>
                                <View style={{flex:1}}></View>
                                <View style={{flex: 2}}>
                                    <Image
                                        source={require('../../assets/gps.png')}
                                        style={styles.placeholder}
                                        alignSelf='center'
                                    />
                                </View>

                                <View style={{flex:3, borderBottomColor: '#84828C',
borderBottomWidth: 1, justifyContent: 'space-evenly'}}>
                                    <Text style=
{styles.amount}>€{this.state.journeyCost.toFixed(2).toString()}</Text>
                                    <Stopwatch
                                        start={this.state.stopwatchStart}
                                        reset={this.state.stopwatchReset}
                                        options={stopwatchOptions}
                                        getTime={(time) => this.getFormattedTime(time)}
                                    />
                                </View>
                            </View>
```

```
                            <View style={{flex: 3}}>
                                <ScrollView>
                                    <JourneyOption
                                        icon='map-marker-distance'
                                        type='material-community'

text={this.state.distanceTravelled.toFixed(2).toString() + " Km"}
                                        addition={db_input.distance}
                                        mileage={true}
                                    />

                                    { this.state.nightdrive ?
                                        (<JourneyOption
                                            icon='moon'
                                            type='feather'
                                            text='Night drive'
                                            addition={db_input.nightdrive_multiplier}
                                        />)
                                        :
                                        (<JourneyOption
                                            icon='sun'
                                            type='feather'
                                            text='Day drive'
                                            addition={null}
                                        />)
                                    }

                                    { provLicence ? (
                                        <JourneyOption
                                            icon='drivers-license-o'
                                            type='font-awesome'
                                            text="Provisional Licence"
                                            addition={db_input.provisional_licence}
                                        />
                                    ) : (
                                        <JourneyOption
                                            icon='drivers-license-o'
                                            type='font-awesome'
                                            text="Full Licence"
                                            addition={null}
                                        />
                                    ) }

                                    {
                                        vehicleClass == 'heavy_class' ? (
                                            <JourneyOption
                                                icon='car-pickup'
                                                type='material-community'
                                                text={this.state.vehiclename + ' (Heavy
Class)'}

                                                addition={db_input.heavyclass}
                                            />
                                        ) : vehicleClass == 'middle_class' ? (
                                            <JourneyOption
                                                icon='car-sports'
                                                type='material-community'
                                                text={this.state.vehiclename + ' (Middle
Class)'}

                                                addition={db_input.middleclass}
```

```
                                        />
                                    ) : vehicleClass == 'light_class' ? (
                                        <JourneyOption
                                            icon='car-hatchback'
                                            type='material-community'
                                            text={this.state.vehiclename + ' (Light
Class)'}
                                            addition={db_input.lightclass}
                                        />
                                    ) : (
                                        <JourneyOption
                                            icon='drivers-license-o'
                                            type='font-awesome'
                                            text="Vehicle Class Error"
                                            addition={null}
                                        />
                                    )
                                }

                                { olderCar ? (
                                    <JourneyOption
                                        icon='calendar'
                                        type='antdesign'
                                        text={"Car older than " +
db_input.age_conditional + " years"}
                                        addition={db_input.age_addition}
                                    />
                                ):(
                                    null
                                )}

                                {
                                    coverage == 'Third Party Insurance' ? (
                                        <JourneyOption
                                            icon='attach-money'
                                            type='material'
                                            text={coverage}
                                            addition={null}
                                        />
                                    ) : coverage == 'Third Party Fire & Theft' ? (
                                        <JourneyOption
                                            icon='attach-money'
                                            type='material'
                                            text={coverage}
                                            addition={null}
                                        />
                                    ) : coverage == 'Comprehensive' ? (
                                        <JourneyOption
                                            icon='attach-money'
                                            type='material'
                                            text={coverage}
                                            addition={null}
                                        />
                                    ) : ( null )
                                }

                                { hardBraking ? (
                                    <View style={{flexDirection: "row",
justifyContent: 'space-evenly', paddingVertical: '2%'}}>
```

```
                                <Icon
                                    name='warning'
                                    type='antdesign'
                                    color='#FFCC00'
                                />
                                <Text>Hard Braking Detected</Text>
                            </View>
                        ) : (
                            <View>

                            </View>
                        )}

                    </ScrollView>
                </View>
            </View>
        ) : (
            <View style={{paddingTop: '40%'}}>
                <Image
                    source={require('../../assets/speedometer.png')}
                    style={styles.image}
                    alignSelf='center'
                />
                <Text style={styles.logo}>Track Journey</Text>
                <Select2
                    isSelectSingle
                    style={{ borderRadius: 5, width: '80%' }}
                    colorTheme={'#2E6CB5'}
                    popupTitle='Select vehicle'
                    title='Select vehicle'
                    data={this.state.vehiclelist}
                    onSelect={key => {
                        this.setState({ vehiclekey: key.toString() });
                    }}
                    onRemoveItem={key => {
                        this.setState({ vehiclekey: key.toString() });
                    }}
                    searchPlaceHolderText='Search Vehicle'
                    cancelButtonText='Cancel'
                    selectButtonText='Choose'
                    listEmptyTitle='No vehicles to show'

                />
            </View>
        )
        )}
    </View>

    <View style={styles.buttonContainer}>
    {
            !this.state.running ? (this.state.showCountdown ? (
                <ButtonComponent
                    text="Cancel"
                    icon="minuscircleo"
                    type="antdesign"
                    onPress={() => this.cancelPressed()}
                />
            ) : (
                <ButtonComponent
```

```
                                text="Start Journey"
                                icon="pluscircleo"
                                type="antdesign"
                                onPress={() => {
                                    if(this.state.vehiclekey == '') {
                                        Toast.show("Please select a vehicle first")
                                        return
                                    }
                                    requestAnimationFrame(() =>
this.setState({showCountdown: true}))
                                }}
                            />
                            )) : (
                                <ButtonComponent
                                    text="End Journey"
                                    icon="closecircleo"
                                    type="antdesign"
                                    onPress={() => requestAnimationFrame(() =>
this.endJourney())}
                                />
                        )
                    }
                </View>
            </View>
        )}
}

const stopwatchOptions = {
    container: {
        alignSelf: 'center',
        borderRadius: 5,
        width: '100%'
    },
    text: {
        fontSize: 28,
        color: '#84828C',
        alignSelf: 'center'
    }
};


//center icons
const styles = StyleSheet.create({
    container: {
        flex: 1,
      },
    summary: {
        width: '100%',
        flex: 1,
    },
    logo:{
        fontWeight:"bold",
        fontSize:32,
        color:"#373E45",
        marginBottom:40,
        textAlign: 'center'
    },
    image:{
        flex: 1,
```

```
        height: 150,
        width: 150,
        resizeMode: 'contain'
    },
    placeholder: {
        flex: 5,
        height: 150,
        width: 150,
        resizeMode: 'contain'
    },
    buttonContainer: {
        flex: 1,
    },
    contentContainer: {
        flex: 6,
        alignItems: 'center',
    },
    amount: {
        fontSize: 40,
        textAlign: 'center',
        color: '#2E6CB5'
    },
    summaryHeaderContainer:{
        flex: 2,
        justifyContent: 'space-between'
    }
})

export default TrackJourney;

{/* <Timer
    start={this.state.timerStart}
    reset={this.state.timerReset}
    totalDuration={3000}
    handleFinish={() => {
        this.setState({hardBraking: false, resetTimer: true, timerStart: false})
    }}
    options={stopwatchOptions}
/> */}
```