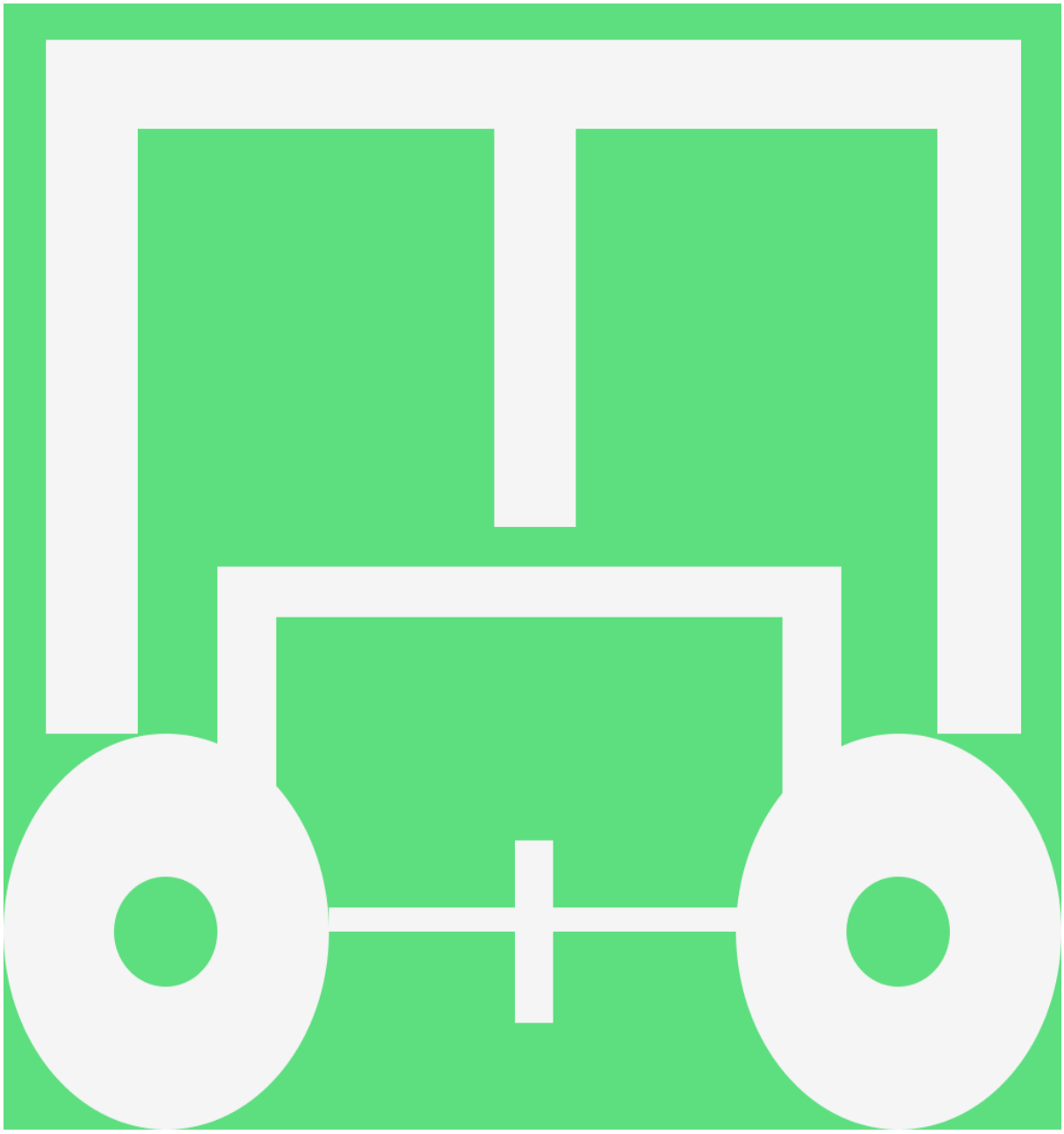
“탈 수 이써!” 최종 보고서

(오픈소스SW개론 002반 1조)



18011684 김건영

18011664 서동원

18011651 이태화

목차

A)변경점들

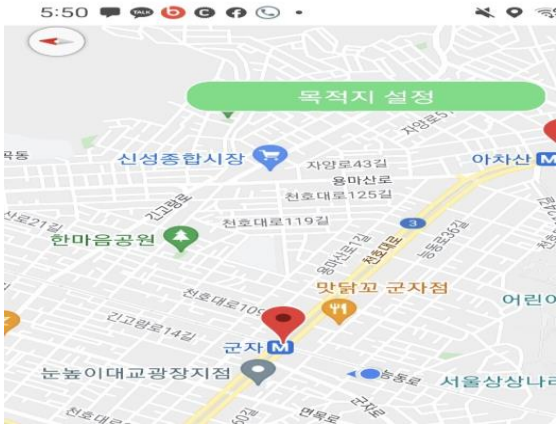
가)회원가입 후 현 위치 및 목적지 출력 -> Google Map 마커 생성 및 목적지 설정 버튼 추가

-현 위치와 목적지를 홈 화면의 좌상단에 띄울 계획이었으나 목적지 설정 버튼 추가 및 Google Map을 이용해 마커로 현재 위치를 표시하는 방향으로 변경

-현 위치 주소를 가져오기로 계획했으나 API 사용과 관련해 마커 터치 시 시작역으로 설정하는 것으로 변경

변경 전 화면

최종 화면



나-1)택시 소요시간 출력 -> WalkScreen, BikeScreen에서 도보 및 자전거 이용 시간 출력

-홈화면에서 현위치에서 목적지까지의 택시 탑승 시 예상 소요시간을 띄울 계획이었으나 택시 API 사용에 있어 제한점(유료 및 네이버 API에서 크롤링 불가)의 문제로 WalkScreen, BikeScreen을 추가해 도보와 자전거 이용 시 예상 소요시간을 불러오는 것으로 변경

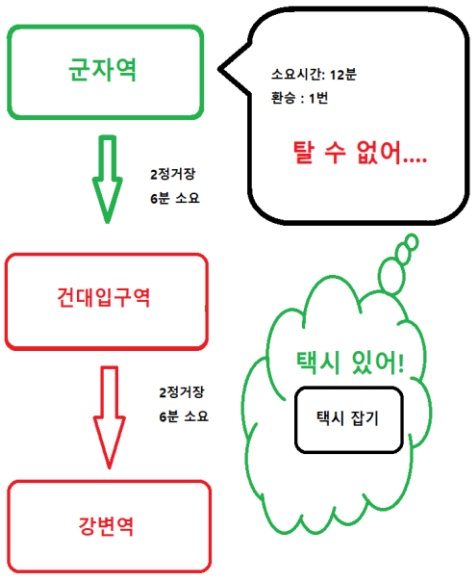
변경 전 화면

최종 화면

나-2) 택시 호출 버튼 -> WalkScreen, BikeScreen으로 도보 및 자전거 이용시간 출력

-이동경로를 보여주는 화면에서 택시 호출버튼 생성을 계획했으나 제한점(유료 및 데이터 연결 제한)으로 WalkScreen, BikeScreen 버튼 추가로 변경

변경 전 화면



최종화면

다) 거리 및 탑승 가능여부에 따른 역 디자인 차별화 -> 삭제

-네이버 API 크롤링으로 마커 생성 시 환승역은 720여개의 API를 불러와야 탑승가능여부를 판단할 수 있기에, 앱 동작 시 속도 지연 및 데이터 충돌로 인해 삭제

변경 전 화면

*가장 가까운 역은 마커의 테두리를 두껍게, 거리가 먼 역은 테두리를 얇게 표시

*탑승가능여부(첫 역에서 끊김, 중간에 끊김)에 따라 색깔 표시

=> 삭제

B) Source Code

가) 사용 오픈소스 및 데이터

a) 네이버 지도 API 웹 크롤링

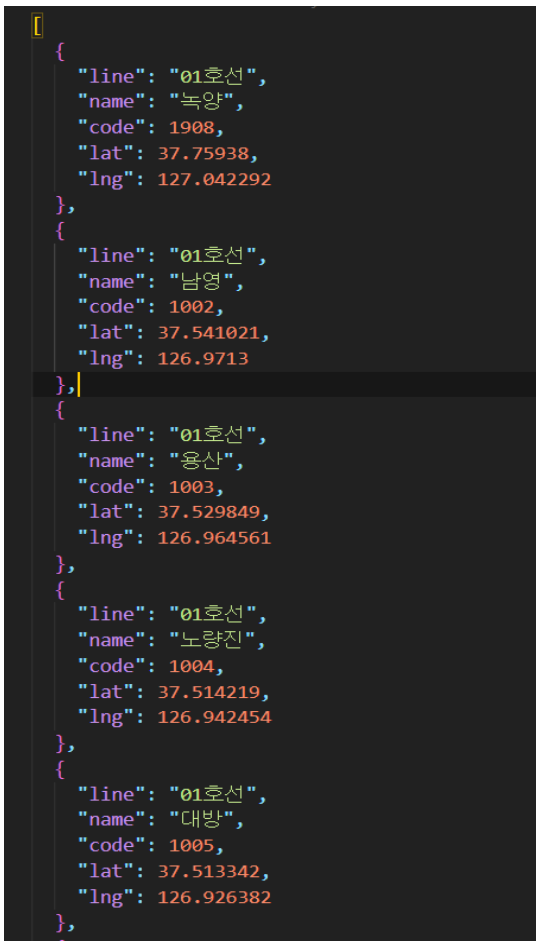
-네이버 지도의 지하철, 도보, 자전거 길찾기 API에서 데이터를 불러오기 위해 웹 크롤링 사용

b) 공공데이터 포털의 지하철 역 JSON 파일

-'서울시 지하철역 정보 검색(역명).JSON' :
지하철역명, 지하철역 코드, 지하철역 호선, 외부코드

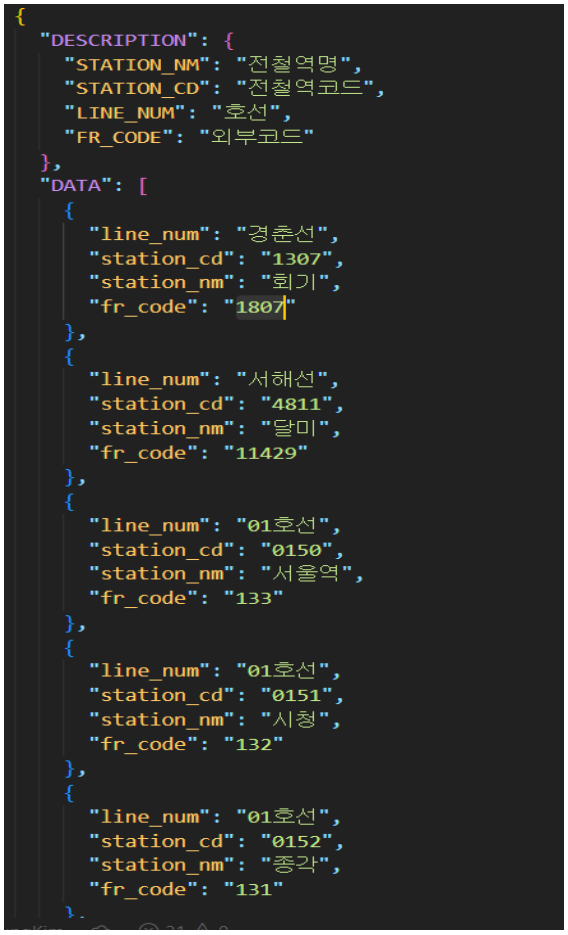
-'metro.JSON':
지하철역명, 외부코드, 지하철역의 위도/경도, 지하철역 호선

metro.JSON



```
[
  {
    "line": "01호선",
    "name": "녹양",
    "code": 1908,
    "lat": 37.75938,
    "lng": 127.042292
  },
  {
    "line": "01호선",
    "name": "남영",
    "code": 1002,
    "lat": 37.541021,
    "lng": 126.9713
  },
  {
    "line": "01호선",
    "name": "용산",
    "code": 1003,
    "lat": 37.529849,
    "lng": 126.964561
  },
  {
    "line": "01호선",
    "name": "노량진",
    "code": 1004,
    "lat": 37.514219,
    "lng": 126.942454
  },
  {
    "line": "01호선",
    "name": "대방",
    "code": 1005,
    "lat": 37.513342,
    "lng": 126.926382
  },
```

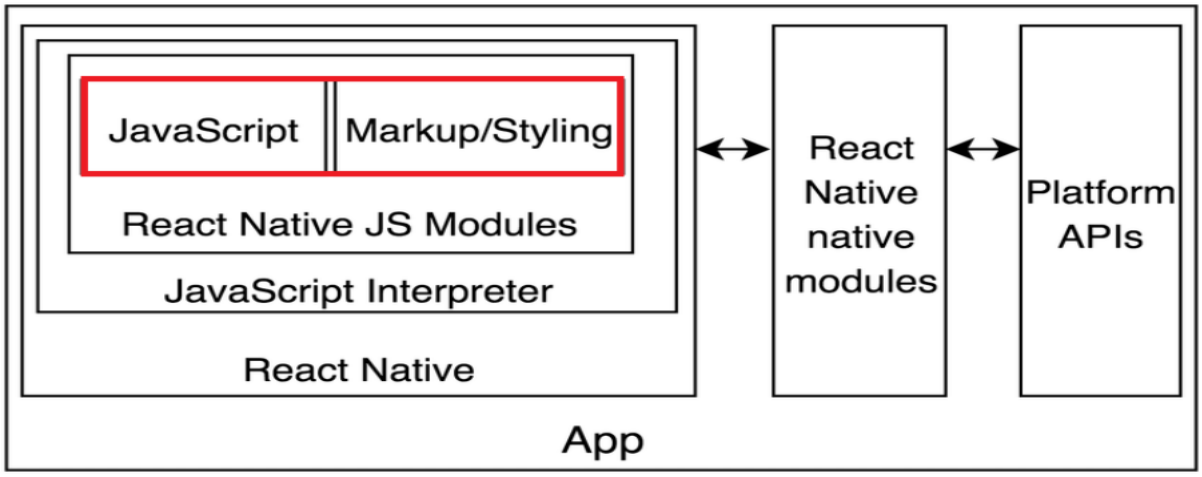

서울시 지하철역 정보 검색(역명).JSON'



```
{
  "DESCRIPTION": {
    "STATION_NM": "전철역명",
    "STATION_CD": "전철역코드",
    "LINE_NUM": "호선",
    "FR_CODE": "외부코드"
  },
  "DATA": [
    {
      "line_num": "경춘선",
      "station_cd": "1307",
      "station_nm": "회기",
      "fr_code": "1807"
    },
    {
      "line_num": "서해선",
      "station_cd": "4811",
      "station_nm": "달미",
      "fr_code": "11429"
    },
    {
      "line_num": "01호선",
      "station_cd": "0150",
      "station_nm": "서울역",
      "fr_code": "133"
    },
    {
      "line_num": "01호선",
      "station_cd": "0151",
      "station_nm": "시청",
      "fr_code": "132"
    },
    {
      "line_num": "01호선",
      "station_cd": "0152",
      "station_nm": "종각",
      "fr_code": "131"
    },
```

c) expo cli

-expo: 애플리케이션을 위한 프레임워크 및 플랫폼. React Native를 이용해 앱 개발 JavaScript와 Style(CSS) 부분의 수정 제외 필요 모듈과 platfrom에서의 구동을 해결. 무료 내장 모듈을 이용해 앱 개발에 사용(ex) Ionicons, expo-location)

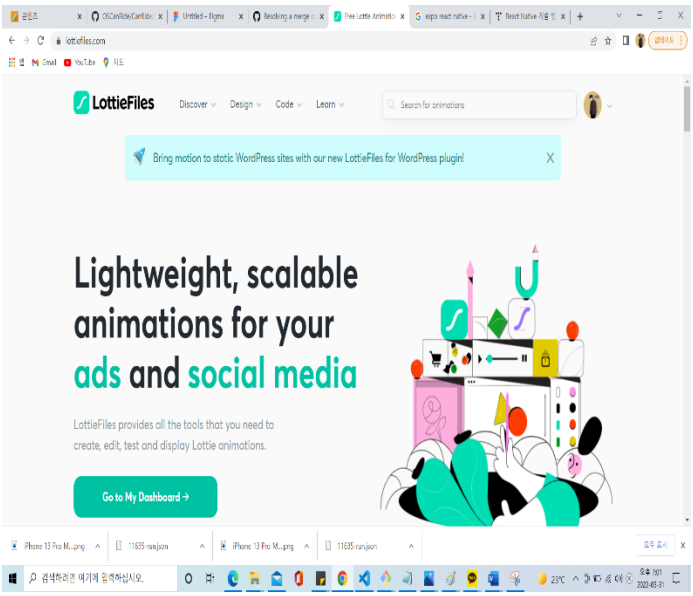


d)Lottie animation(https://lottiefiles.com)

-앱 내 애니메이션 구현을 위한 애니메이션 파일을 JSON 형태의 파일로 구현해놓은 오픈소스

-Splash Screen 이후 로딩화면, WalkScreen/BikeScreen의 애니메이션 재생에 사용

lottieFiles 홈페이지



BikeScreen의 사용한 JSON 파일

나) 전체 코드 개요

a)App.js

-react-navigation의 NativeStackNavigator 패키지를 통해 화면 전환을 Stack.Screen으로 구현

- IOS, Android Platform에서의 통일성을 위해 자동으로 생성되는 header삭제, 각 screen 별 버튼(MapScreen의 마커, DetailsScreen의 walk, bike 버튼) 터치시 화면 전환 구현

*App.js의 모든 Screen을 담은 NavigationContainer

*초기 화면(initialRouteName)을 MapScreen으로 지정

로딩이 완료되면 MapScreen을 사용자에게 보여줌.

b)폴더

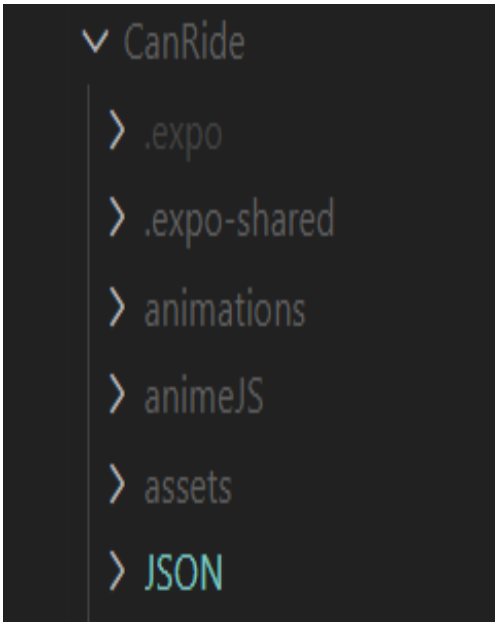
*JSON: API 크롤링과 연관 시 필요한 데이터 담겨있는 JSON 파일 저장 폴더

*animations: lottie animation JSON 파일 저장 폴더

*animeJS: animations의 JSON 파일을 불러와 LottieView를 컴포넌트화 시킨 js 파일 저장 폴더

*screens: NavigationContainer 안의 화면 구현을 위한 js 파일 저장 폴더

*assets: 앱 첫 로딩 시의 Splash Screen을 저장하는 폴더



c) styles.js

-Screen 별 컴포넌트의 style prop을 저장해놓은 js 파일

```
import { StyleSheet } from "react-native";

const styles = StyleSheet.create({
  overlay: {
    flex: 1,
    justifyContent: "flex-end",
    backgroundColor: "rgba(0, 0, 0, 0.4)",
  },
  background: {
    flex: 0,
  },
  bottomSheetContainer: {
    height: 300,
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: "white",
    borderTopLeftRadius: 10,
    borderTopRightRadius: 10,
  },
  rootContainer: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
  },
  map: {
    flex: 1,
    width: "100%",
    height: "100%",
  },
```

***screen 폴더의 각 화면의 js 파일을 import 해와 App.js의 NavigationContainer 안에서 각 화면 전환이 이루어지게 된다.**

```
import MapScreen from "./screens/MapScreen";
import DetailsScreen from "./screens/DetailsScreen";
import WalkScreen from "./screens/WalkScreen";
import BikeScreen from "./screens/BikeScreen.js";
import LottieView from "lottie-react-native";
```

다) Screens

a)Splash Screen 과 loading 화면

1)Splash Screen

-expo cli가 제공하는 expo-splash-screen을 통해 app.json을 이용해 splash screen을 변경한다.

app.json

*image를 assets 폴더 안의 rideSplashScreen.png로 변경

*resizeMode: cover로 설정, 화면 크기에 상관없이 모든 화면을 덮게 설정

```
"splash": {
  "image": "./assets/rideSplashScreen.png",
  "resizeMode": "cover",
  "backgroundColor": "#ffffff"
},
```

App.js

*delay_splash function 안에서 sleep function을 이용해 splashscreen을 2초 동안 보여주고 App.js로 넘어가도록 설정

```
function sleep(ms) {
  return new Promise((resolve) => setTimeout(resolve, ms));
}

async function delay_splash() {
  await SplashScreen.preventAutoHideAsync();
  await sleep(2000);
  await SplashScreen.hideAsync();
}

function app() {
  delay_splash();
```

SplashScreen 실행 화면

2) loading 화면

App.js

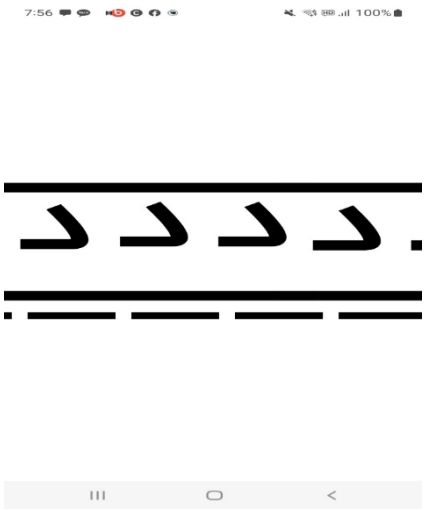
```
function app() {
  delay_splash();

  const [loaded, setLoaded] = useState(false);
  if (loaded == false) {
    return (
      <View
        style={{
          flex: 1,
          alignItems: "center",
          margin: 0,
        }}
      >
        <LottieView
          source={require("./animations/loadingTrain.json")}
          autoPlay
          loop={false}
          resizeMode="cover"
          onAnimationFinish={() => {
            setLoaded(true);
          }}
        />
      </View>
```

- *react의 useState 패키지를 이용해 loaded, setLoaded 변수를 선언.
- *loaded의 값이 false일 경우animations 폴더의 lottie animation (loadingTrain.json)를 실행시키는 LottieView를 렌더링.
- animation의 종료 후 setLoaded를 통해 loaded의 값을 true로 변경.

```
  else {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Map">
        <Stack.Screen
          name="Map"
          component={MapScreen}
          options={{
            headerShown: false,
          }}
        />
        <Stack.Screen
          name="Details"
          component={DetailsScreen}
          options={{
            headerShown: false,
          }}
        />
```

- loaded의 값이 true일 경우 NavigationContainer를 렌더링.
- initialRouteName을 Map으로 설정하여 로딩 화면 이후 MapScreen을 바로 화면에 출력

*** SplashScreen 이후의 실행 화면**



3)MapScreen.js

a)import, 변수, 함수 및 초기 작업

1.import

```
import * as React from "react";
import {
  Text,
  View,
  Dimensions,
  Alert,
  TextInput,
  TouchableOpacity,
  Modal,
  TouchableWithoutFeedback,
} from "react-native";
import styled from "styled-components";
import { useState, useEffect } from "react";
import MapView, { PROVIDER_GOOGLE, Marker } from "react-native-maps";
import * as Location from "expo-location";
import metro from "../JSON/metro.json";

import code from "../JSON/서울시 지하철역 정보 검색 (역명).json";
import axios from "axios";
import haversine from "haversine-distance";

import styles from "../styles";
```

*해당 import문으로 앱 제작에 필요한 JSON파일과, 모듈을 불러왔다.

- Import MapView, {PROVIDER_GOOGLE, Marker} from "react-native-maps"는 리액트 네이티브에서 제공해주는 구글 맵을 불러오기 위해서 import해준 패키지
- Axios는 네이버 웹에서 지하철의 경로를 크롤링 해오기 위해서 import해준 패키지

2.변수 선언

```
30    const { width: SCREEN_WIDTH, height: SCREEN_HEIGHT } = Dimensions.get('
```

*IOS/ Android 디바이스 별로 화면 크기가 다르기에 유동적으로 지도를 띄워 주기 위한 화면 크기 불러오기

```
const [initialRegion, setInitialRegion] = useState({
  latitude: 37.548014,
  longitude: 127.074658,
  latitudeDelta: 0.03,
  longitudeDelta: 0.03,
});
```

*useState 이용, 초기 좌표값인 initialRegion에 값 할당. 세종대학교를 초기 위치로 설정 (latitudeDelta, longitudeDelta는 지도의 축척을 의미.)

```
const [mapWidth, setMapWidth] = useState("99%");
const [location, setLocation] = useState(0);
const [errorMsg, setErrorMsg] = useState(null);
const [inputText, setInputText] = useState("");
const [destination, setDestination] = useState("");
const [desName, setDesName] = useState("");
const [deslat, setDeslat] = useState("");
const [deslng, setDeslng] = useState("");
const [searchValue, setSearchValue] = useState("");
const ccode = {};
let ccodelat = "";
let ccodelng = "";
let endlat = "";
let endlng = "";
const [modalVisible, setModalVisible] = useState(true);
```

*MapScreen에서 사용되는 State와 변수 선언.

- -location, setLocation: GoogleMap 이용 시 현재 위치값 받아오기 위한 변수
- -inputText, setInputText: 목적지 설정 시 TextInput component 안의 값 저장을 위한 변수
- -destination, setDestination: 목적지 설정 후 서울시 지하철 정보 검색 (역명).json에서 도착역에 대한 코드 저장을 위한 변수
- -desName, setDesName: 목적지 설정 후 서울시 지하철 정보 검색 (역명).json에서 도착역의 이름 저장을 위한 변수
- -deslat, deslng: 목적지 설정 후 metro.json에서 도착역의 위도,경도 값 저장을 위한 변수
- -modalVisible, setModalVisible: MapScreen으로 전환 후 바로 목적지 입력창을 띄우기 위한 변수
- -errorMsg: 앱 실행 중 error 메시지 출력을 위한 변수

3.함수 선언

```
71    const updateMapStyle = () => {
72      setMapWidth("100%");
73    };
74
75    const set_fistDestination = () => {
76      setInputText("");
77    };
78    const handleDestination = () => {
79      setDestination("");
80      setDesName("");
```

- set_firstDestination : 해당 목적지를 초기화 시켜주는 함수. InputText를 빈텍스트로 만들어준다.
- handleDestination : 목적지를 저장해주는 함수. 입력된 Text값을 기반으로 위에서 선언한 지하철 역의 이름, 역의 고유 코드, 역의 호선이 담긴 JSON파일과 비교하여 inputtext와 객체와 동일할 경우에 그것에 대한 정보를 useState에 저장해줌. 또한 walkScreen에서 사용될 목적지 역의 좌표를 metro.json에서 불러온다.

```
148    const pressButton = () => {
149      setModalVisible(true);
150    };
151
152    const closeModal = () => {
153      setModalVisible(false);
154    };
```

- Loading 화면 이후 MapScreen으로 전환 시 목적지 입력창을 바로 띄워주기 위한 함수

b)MapScreen 동작 방식

1.현재 위치 정보 불러오기

```
// Get current location information
useEffect(() => {
  (async () => {
    let { status } = await Location.requestForegroundPermissionsAsync(
    if (status !== "granted") {
      setErrorMsg("Permission to access location was denied");
      return;
    }

    let location = await Location.getCurrentPositionAsync({});
    setLocation(location);
    for (var i = 0; i < array.length; i++) {
      const a = {
        latitude: location.coords.latitude,
        longitude: location.coords.longitude,
      };
      const b = {
        latitude: array[i].lat,
        longitude: array[i].lng,
      };
```

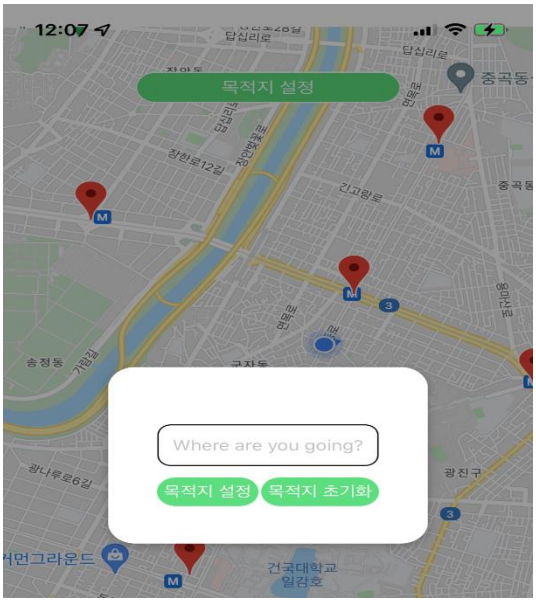
- *현재 앱에서 Google Map을 사용하기 위해 사용자의 현재 위치의 위도, 경도값을 가져온다. 이때 useEffect를 통해 화면이 전환될 때마다 계속 위도, 경도값을 가져오는 것을 막고 MapScreen으로 화면 전환 시 한번만 사용자의 현재 위치 정보를 가져오도록 설정한다.
- *async/ await function을 이용해 status에 현재 위치를 가져오는 작업에 대한 허용상태를 반환하는 requestForegroundPermissionsAsync을 status에 할당한다. 이때 status가 위치 정보를 가져오는 사용자로부터 받지 못한다면 error 메시지를 출력한다.
- *location 변수에 getCurrentPositionAsync 함수를 통해 현재 위치의 위도, 경도값을 저장한다.

2. 목적지 입력창(Modal)

```
157     'rapper style={{ backgroundColor: "white" }}>
158     <View style={styles.centeredView}>
159       <Modal animationType="fade" transparent={true} visible={modalVisible}
160         <View style={styles.overlay}>
161           <TouchableWithoutFeedback onPress={closeModal}>
162             <View style={styles.background} />
163           </TouchableWithoutFeedback>
164           <View style={styles.centeredView}>
165             <View style={styles.modalView}>
166               <TextInput
167                 style={[styles.TextInput, { width: "80%" }]}
168                 placeholder="Where are you going?"
169                 value={inputText}
170                 onChangeText={setInputText}
171               />
172               <View style={{ flexDirection: "row", alignSelf: "center" }}>
173                 <TouchableOpacity
174                   style={styles.button}
175                   onPress={() => {
176                     if (inputText === "") {
177                       Alert.alert("도착지 이름을 입력해주세요!");
178                     } else {
179                       handleDestination();
180                       setModalVisible(!modalVisible);
181                     }
182                   }}
183                 >
184                   <Text style={{ textAlign: "center", color: "white" }}>
185                     목적지 설정
186                   </Text>
187                 </TouchableOpacity>
188                 <TouchableOpacity
189                   style={styles.button}
190                   onPress={set_fistDestination}
191                 >
192                   <Text style={{ textAlign: "center", color: "white" }}>
193                     목적지 초기화
194                   </Text>
195                 </TouchableOpacity>
```

- Modal을 사용함으로써 앱 실행 시 처음에 도착지에 대한 역을 입력 받아 버튼을 누르면 handleDestination 함수를 실행하여 도착지에 대한 정보를 저장. 그 후 modal 꺼지고 본격적인 지도 표출.
- 버튼의 디자인을 위한 <Button> 태그가 아닌 <TouchableOpacity> 사용.
- 미리 선언한 handleDestination, setModalVisible, set_firstDestination 함수 사용

***실행화면**



3. MapView

```
<MapView
  initialRegion={initialRegion}
  style={[styles.map, { width: SCREEN_WIDTH, zIndex: 1 }]}
  provider={PROVIDER_GOOGLE}
  showsUserLocation={true}
  showsMyLocationButton={true}
  onMapReady={() => {
    updateMapStyle();
  }}
>
  {metro.map((marker, index) => {
    return (
      <Marker
        key={index}
        coordinate={{
          latitude: marker.lat ? marker.lat : 0,
          longitude: marker.lng ? marker.lng : 0,
        }}
        onPress={() => {
          if (inputText === "") {
            Alert.alert("도착지에 대한 정보가 없습니다!");
          } else {
            var station_code = code.DATA;
            var ccode = "";
            var ccodeName = "";
            for (var i = 0; i < station_code.length; i++) {
              if (station_code[i]["station_nm"] === marker.name) {
                ccode = station_code[i]["fr_code"];
                ccodeName = station_code[i]["station_nm"];
                ccodelat = marker.lat;
                ccodelng = marker.lng;
              }
            }
            for (var j = 0; j < metro.length; j++) {
              if (metro[j].name === inputText) {
                endlat = metro[j].lat;
                endlng = metro[j].lng;
              }
            }
```

- 화면에 맵과 마커를 띄우는 코드. Map함수를 사용하여 metro에 담겨있는 지하철 역의 위도와 경도를 인자로 받아. 마커를 그 위치에 띄움.
- 마커 터치 시, text에 입력한 정보가 없다면 Alert창을 띄움.
- 도착지가 입력되었다면 마커에 담겨있는 그 마커가 해당하는 지하철 역의 이름과 json파일의 역 이름을 비교하여 같을시 json파일에 있는 역 이름과 역 고유코드, 위도 경도를 가져옴. Walkscreen, BikeScreen에서 사용할 도착지에 대한 좌표도 가져옴.

```
259    const URL = `https://map.naver.com/v5/api/transit/directions/subway?st
260    //const test = allclear;
261    //const test = transnogo;
262    //const test = none;
263    //const test = stopsameline;
264    //const test = ziseok;
265
266    if (destination === "") {
267      Alert.alert("도착지에 대한 정보가 없습니다!");
268    } else {
269      axios.all([axios.get(URL)]).then(
270        axios.spread((data) => {
271          let cnt = 0; //환승역 갯수
272          let count = 0; //출발지와 목적지
273          let ccnt = 0;
274          let ccc = 0;
275          let placelist = [];
276          let list = [];
277          let stationcntlist = [];
278          let starthourlist = [];
279          let startMinlist = [];
280          let startseclist = [];
281          let codelist = [];
282          let listColor = [];
283          const legs = data.data.paths[0].legs[0];
```

- 마커 터치 시 json에서 얻어온 출발지에 대한 정보(역 이름, 역 고유 코드)와 초반 도착지 설정으로 가져온 도착지에 대한 정보(역 이름, 역 고유 코드)와 현재 시간을 인자로 받아 URL 완성

```
const URL =
`https://map.naver.com/v5/api/transit/directions/subway?start=${ccode}&goal=${destination}&departureTime=${y
ear}-${month}-${date}T${hours}%3A${minutes}%3A${seconds}`;
```

- axios를 사용하여 URL에 접근하여 웹 크롤링을 시도함. URL의 정보는 data변수에 저장됨.

```
14   "paths": [
15     {
16       "idx": 1,
17       "mode": "TIME",
18       "type": "SUBWAY",
19       "optimizationMethod": "MINIMUM_DURATION",
20       "labels": ["최적"],
21       "departureTime": "2022-05-25T22:50:50",
22       "arrivalTime": "2022-05-25T23:56:00",
23       "alarmDepartureTime": "2022-05-25T22:50:50",
24       "alarmArrivalTime": "2022-05-25T23:56:00",
25       "duration": 66,
26       "intercityDuration": 0,
27       "walkingDuration": 2,
28       "waitingDuration": 4,
29       "distance": 35650,
```

- 웹 크롤링시 가져오는 정보를 담은 JSON 파일(보고서 제출을 위해 정적인 파일로 만들어 놓은 모습)

```
placelist[ccnt] = legs.steps[0].stations[0].placeId;
ccnt++;
placelist[ccnt] =
  legs.steps[legs.steps.length - 1].stations[
    legs.steps[legs.steps.length - 1].stations
      .length - 1
  ].placeId;
for (i = 0; i < legs.steps.length; i++) {
  if (legs.steps[i].type === "SUBWAY") {
    if (i === legs.steps.length - 1) {
      transcount = transcount - 1;
    }
    if (data.data.paths[0].shutdown === true) {
      startOK = false;
    } else if (
      data.data.paths[0].legs[0].steps[i].shutdown ===
      true
    ) {
      transferOK[transcount] = false;
    } else {
    }
    listColor[ccc] =
      data.data.paths[0].fares[0].routes[
        ccc
      ][0].type.color;
    ccc++;
```

- 출발지와 도착지에 해당하는 역의 장소번호를 placelist에 저장
- 크롤링을 해오면 지하철을 타는 Step이 있는데 이 Step은 환승역에서 지하철을 갈아 탈 때 환승역에서 걸어가는 것도 나오므로 우리에게는 필요가 없는 정보이다. 따라서 type이 SUBWAY일 때만 for문을 실행한다라고 조건문을 걸어줌. Transount는 환승역의 갯수를 나타냄.
- Shutdown은 지하철의 막차가 끊겼는지 알려주는 객체, Shoutdown의 위치에 따라 지하철이 환승역에서 끊겼는지 처음부터 끊겼는지 파악 가능. Listcolor는 해당 지하철의 호선의 색깔을 뜻함.

```
      hour: legs.steps[i].departureTime.substring(
        11,
        13
      ),
      minute: legs.steps[i].departureTime.substring(
        14,
        16
      ),
      second: legs.steps[i].departureTime.substring(
        17,
        19
      ),
    };

    starthourlist[count] = setTime.hour;
    startMinlist[count] = setTime.minute;
    startseclist[count] = setTime.second;
    stationcntlist[count] =
      legs.steps[i].stations.length;
    count++;
```

- 환승역과 환승역 또한 환승역과 도착역, 환승역과 출발역 사이의 걸리는 시간을 알아내기 위한 코드.

```
            for (
              j = 0;
              j < legs.steps[i].stations.length;
              j++
            ) {
              console.log(legs.steps[i].stations[j].name);
            }
            if (transferOK[transcount] === false) {
              transferOKOK[tc] = transferOK[transcount];
              tc++;
            }
            if (i + 1 != legs.steps.length) {
              list[cnt] = legs.steps[i].stations[j - 1].name; //환승역 이름
              codelist[cnt] =
                legs.steps[i].stations[j - 1].displayCode; //환승역 코드 담
              cnt++;
            } else {
              starthourlist[count] = legs.steps[
                i
              ].arrivalTime.substring(11, 13);
              startMinlist[count] = legs.steps[
                i
              ].arrivalTime.substring(14, 16);
              startseclist[count] = legs.steps[
                i
              ].arrivalTime.substring(17, 19);
              count++;
            }
            transcount++;
          } else {
          }
        }
        if (startOK === false) {
          transferOKOK[tc] = false;
        }
```

- 거쳐가는 지하철 역을 알려주는 코드. 지하철을 거쳐갈 때 환승역에서 지하철이 끊기는지 조건문으로 판단해주고, list에는 환승역에 대한 이름을 저장. codeList에는 환승역의 고유 코드를 저장해줌.

```
navigation.navigate("Details", {
  S: ccode,
  SName: ccodeName,
  Lcolor: listColor,
  cnt: cnt,
  Tname: list,
  T: codelist,
  E: destination,
  EName: desName,
  //건영이 추가
  PT: placelist,
  sLat: ccodelat,
  sLng: ccodelng,
  eLat: endlat,
  eLng: endlng,
  startOKColor: startOK,
  transferOKColor: transferOKOK,
  stationcntlist: stationcntlist,
  starthourlist: starthourlist,
  startMinlist: startMinlist,
```

- 마커 터치 시 MapView에서 저장된 데이터를 detailscreen으로 넘겨준다.

➢ startOKColor : 출발지에서 지하철이 끊겼는지 안 끊겼는지 판단 후 true,false로 나눠준 변수

➢ transferOKColor : 환승역마다 지하철이 끊겼는지 안 끊겼는지 판단 후 true,false로 나눠준 리스트

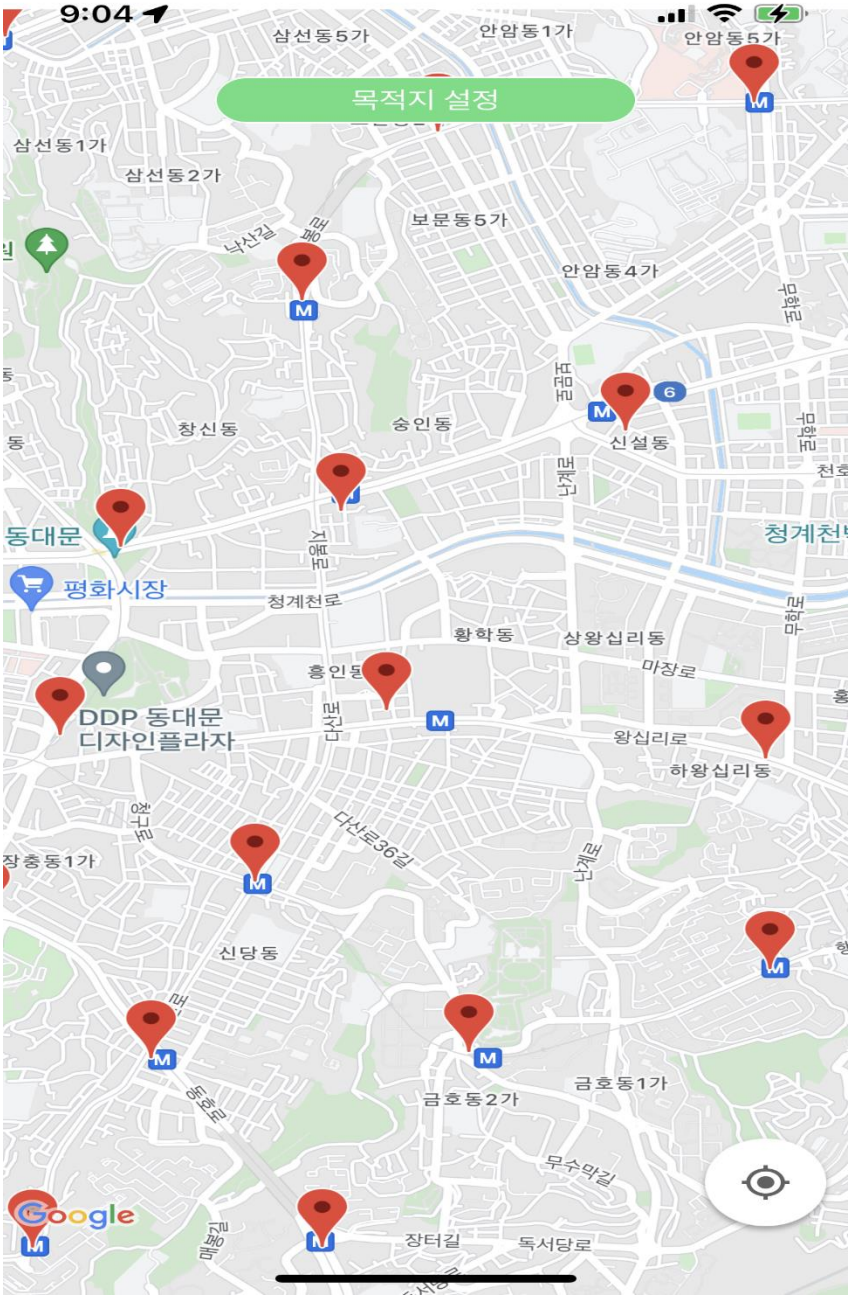
➢ stationcntlist : 환승역과 환승역 사이의 지하철 역 갯수 리스트

➢ stationhourlist : 환승역과 환승역 사이의 걸리는 시간에서 시 리스트

➢ stationMinlist : 환승역과 환승역 사이의 걸리는 시간에서 분 리스트

➢ stationSeclist : 환승역과 환승역 사이의 걸리는 시간에서 초 리스트

4.목적지 입력 후 전체 실행 화면



4)DetailsScreen.js

a)MapScreen으로부터의 인자 전달

```
const DetailsScreen = ({ route, navigation }) => {
  const {
    S,
    SName,
    Lcolor,
    cnt,
    Tname,
    T,
    E,
    EName,
    startOKColor,
    transferOKColor,
    //건영이 추가
    PT,
    sLat,
    sLng,
    eLat,
    eLng,
    stationcntlist,
    starthourlist,
    startMinlist,
    startseclist,
  } = route.params;
```

- MapScreen으로부터 전달받은 인자
- startOKColor는 첫 역에서부터 지하철을 탈 수 있는지 없는지 여부를 저장, transferOKColor는 환승역에서 탈 수 없다면 false 값을 저장하는 배열.
- LColor에는 각 역의 호선의 색깔을 저장하는 배열

```
if (startOKColor == true && transferOKColor.length === 0) {

  Possible = "탈 수 있어  !";
} else {
  Possible = "탈 수 없어 ..";
}
```

- startOkColor가 true이며 transferOKColor.length === 0 인 경우 Possible 변수에 "탈 수 있어" 저장, 반대의 경우에는 Possible에 "탈 수 없어"를 저장

b) 화면 출력

*역의 경로와 탈 수 있는지의 여부/WalkScreen,BikeScreen을 따로 담을 수 있는 화면 2개를 만들기 위해 styles.js 파일에서 leftWrap과 rightWrap을 나눈 후 각각을 자식 컴포넌트로 묶어줌.

```
return (
  <View style={styles.DetailContainer}>
    <View style={styles.leftWrap}>
      <View style={styles.start}>
        <Text
          style={
            startOKColor
              ? [styles.startName, { color: Lcolor[0] }]
              : [styles.startName, { color: Lcolor[0] }, styles.fail]
          }
        >
          {SName}
        </Text>
        <View style={{ flexDirection: "row" }}>
          <Text
            style={
              startOKColor ? styles.arrow : [styles.arrow, styles.failArrow]
            }
          >
            &darr;
          </Text>
          <View style={{ flexDirection: "column", marginTop: 20 }}>
            <Text style={{ fontSize: 15 }}>
              {stationcntlist[0] - 1}정거장
            </Text>
            <Text style={{ fontSize: 15 }}>
              {time[0]}분 소요
            </Text>
          </View>
```

- Lcolor를 이용해 첫 역부터 도착역까지 모든 경로의 지하철 호선 색깔로 Text안의 역 이름을 화면에 출력
- HTML의 &darr과 style에 arrow를 생성하고 크기와 색상을 설정하여 아래를 가리키는 화살표를 출력
- 정거장 수는 MapScreen에서 전달받은 거쳐가는 역의 개수를 저장하는 배열인 stationcntlist의 값을 이용하여 출력
- style.js에 fail과 failArrow를 생성하고 <Text> 안에서 삼항연산자를 이용하여 조건이 참인 경우 초기 설정으로 출력하고 거짓일 경우 fail과 failArrow style로 출력하는 방식으로 구현

```
let time = [];
for (let i = 0; i <= cnt; i++) {
  if (startMinlist[i + 1] - startMinlist[i] < 0) {
    time[i] = startMinlist[i + 1] - startMinlist[i] + 60;
  }
  else {
    time[i] = startMinlist[i + 1] - startMinlist[i];
  }
}
```

- 소요시간은 각 역의 탑승하는 시간 중 분을 저장하는 배열인 startMinlist에 저장되어 있는 값을 이용하여 다음 인덱스에서 현재 인덱스의 값을 빼서 0이상이면 그대로 출력하고 0미만일 경우 60을 더하는 방식으로 구현

```
<View style={styles.start}>
  <Text
    style={
      startOKColor
        ? [styles.startName, { color: Lcolor[0] }]
        : [styles.startName, { color: Lcolor[0] }, styles.fail]
    }
  >
    {SName}
  </Text>

  <View key={index} style={[styles.middle, { flex: index - 1 }]}>
    <Text
      key={index}
      style={
        transferOKColor[index] || transferOKColor.length === 0

          ? [styles.transferName, { color: Lcolor[index + 1] }]
          : [
            styles.transferName,
            { color: Lcolor[index + 1] },
            styles.fail,
          ]
      }
    >
      {Tname[index]}
  <View style={styles.end}>
    <Text
      style={
        startOKColor && transferOKColor.length === 0
          ? [styles.endName, { color: Lcolor[cnt] }]
          : [styles.endName, { color: Lcolor[cnt] }, styles.fail]
      }
    >
      {EName}
    </Text>
  </View>
</View>
```

- Tname: 환승역에 대한 정보를 담고 있는 배열
- 탑승이 가능한 역은 초록색, 탑승이 불가능한 역은 빨간색으로 출력하기 위해 시작역의 조건은 startOKColor의 참,거짓을 이용

- 환승역은 transferOKColor의 length가 0이거나 transferOKColor[index]의 참인 경우 초록색, 아닌 경우 빨간색으로 출력되도록 구현
- 도착역은 startOKColor가 참이면서 transferOKColor의 length가 0이면 초록색, 아닌 경우 빨간색으로 출력되도록 구현

c) WalkScreen, BikeScreen 전환 버튼

```
<TouchableOpacity
  style={styles.button}
  onPress={() => {
    let sLine = "";
    let eLine = "";
    const station_code = code.DATA;
    //let distance = 0;
    for (var i = 0; i < station_code.length; i++) {
      if (station_code[i]["fr_code"] === S) {
        sLine = station_code[i]["line_num"];
      }
      if (station_code[i]["fr_code"] === E) {
        eLine = station_code[i]["line_num"];
      }
    }
    const desID = PT[1];
    const walk_url = encodeURI(
      `https://map.naver.com/v5/api/dir/findwalk?lo=ko&st=1&o=all&l=${sLng},${sLat},${SName}역%2${sLine},${PT[0]};${
    );
    axios.get(walk_url).then((data) => {
      let distance = data.data.routes[0].summary.distance;
      let duration = data.data.routes[0].summary.duration;
      let stepCount = data.data.routes[0].summary.stepCount;
      navigation.navigate("Walk", {
        S: S,
        E: E,
        sLat: sLat,
        sLng: sLng,
        eLat: eLat,
        eLng: eLng,
        PT: PT,
        SName: SName,
        EName: EName,
        distance: distance,
<TouchableOpacity
  style={styles.button}
  onPress={() => {
    let sLine = "";
    let eLine = "";
    const station_code = code.DATA;
    const desID = PT[PT.length - 1];
    const bike_url = encodeURI(
      `https://map.naver.com/v5/api/dir/findbicycle?start=${sLng},${sLng}
    );
    //let distance = 0;
    for (var i = 0; i < station_code.length; i++) {
      if (station_code[i]["fr_code"] === S) {
        sLine = station_code[i]["line_num"];
      }
      if (station_code[i]["fr_code"] === E) {
        eLine = station_code[i]["line_num"];
      }
    }
    axios.get(bike_url).then((data) => {
      //console.log(data.data.routes[0].summary);
      let distance = data.data.routes[0].summary.distance;
      let duration = data.data.routes[0].summary.duration;
      let taxi_fare = data.data.routes[0].summary.taxi_fare;
      navigation.navigate("Bike", {
        distance: distance,
        duration: duration,
      });
    });
  }}
>
  <Ionicons name="bicycle" size={80} color="white" />
</TouchableOpacity>
```

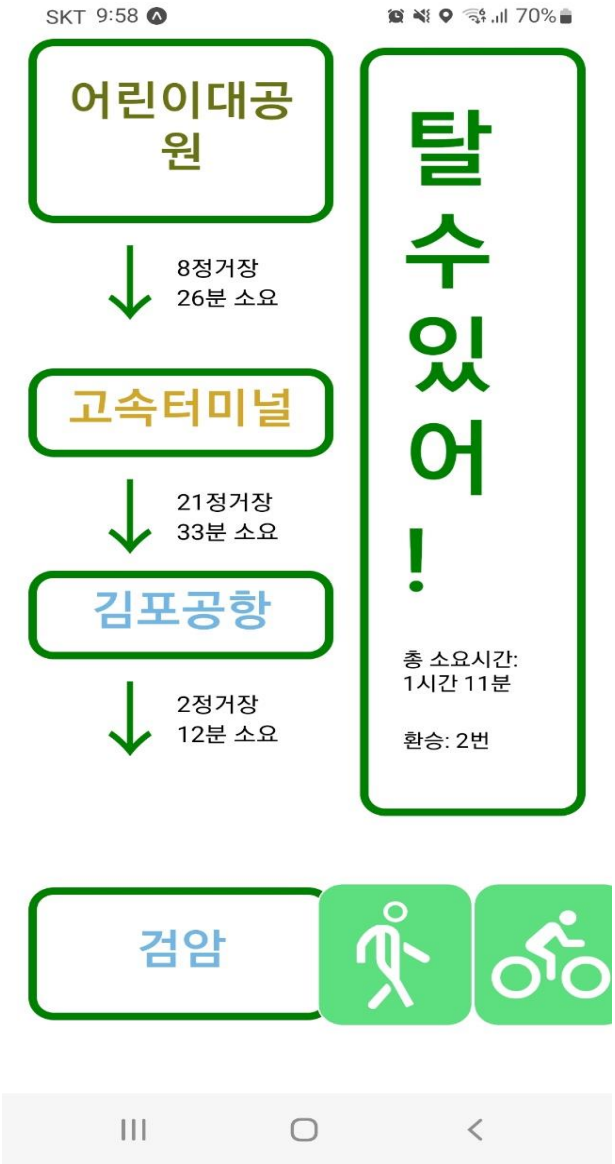
- TouchableOpacity 컴포넌트를 이용해 버튼 생성
- 각각의 버튼에 onpress 속성에 navigation.navigate를 추가해 각각 walkscreen과

bikescreen으로 전환될 수 있게 작성

- 도보, 자전거 경로의 경우 출발,도착지의 위도/경도, 호선 추가로 지하철 API에서 크롤링 해온 장소 고유의 ID를 인자로 넘겨주어 크롤링을 실행
  - 호선의 경우 sLine, eLine 변수에 각각 할당한다.
- 크롤링 후 거리, 예상 소요시간, 도보수의 인자를 각각 WalkScreen, BikeScreen으로 넘겨준다.

d) 4가지 경우의 실행 화면

***처음부터 탈 수 있는 경우**



***지하철이 끊기는 경우(환승)**

*** 지하철이 끊긴 경우(환승 X)**

***출발역부터 지하철이 끊긴 경우**

5) Walk/BikeScreen.

a)함수

```
const hourormin = () => {
  if (duration > 3600) {
    return (
      <Text style={styles.walkdistanceText}>
        소요시간: {(duration / 3600).toFixed(1)}시간
      </Text>
    );
  } else {
    return (
      <Text style={styles.walkdistanceText}>
        소요시간: {(duration / 60).toFixed(1)}분
      </Text>
    );
  }
};

const howHard = () => {
  {
    if (distance > 6000) {
      return <Text style={[{ color: "red" }]}>멀어요!</Text>;
    } else if (distance <= 6000 && distance >= 2000) {
      return <Text style={[{ color: "orange" }]}>할만해요!</Text>;
    } else {
      return <Text style={[{ color: "green" }]}>코앞이네!</Text>;
    }
  }
};
```

- 크롤링을 통해 가져오는 예상 소요시간과 거리의 경우 각각 초단위, 미터단위로 가져옴. 화면에 출력되는 값의 통일성을 위해 hourormin 함수로 1시간이 넘는다면 시간을, 1시간 미만이라면 분으로 소요시간을 변환해서 렌더링함.
- HowHard 함수를 통해 거리에 따라 사용자에게 경로의 강도를 색깔과 텍스트로 표시함

b)시간에 따른 인터페이스 변화

```
let today = new Date();
let hours = today.getHours();
```

- 현재 시간을 hours에 저장

```
if ((hours >= 0 && hours <= 5) || (hours <= 23 && hours >= 19)) {
  return (
    <View style={[styles.walkcontainer, { backgroundColor: "grey" }]}>
      <View style={styles.distance}>
        <Text style={[styles.walkdistanceText, { color: "yellow" }]}>
          거리: {(distance / 1000).toFixed(1)}km
        </Text>
        <Text style={{ color: "yellow" }}>{hourormin()}</Text>
      </View>
      <View
        style={{
          flex: 1,
          alignItems: "center",
          justifyContent: "center",
          backgroundColor: "grey",
        }}
      >
        <Text style={{ fontSize: 20 }}>{howHard()}</Text>
        <BikeMan />
      </View>
    </View>
  );
} else {
  return (
    <View style={[styles.walkcontainer, { backgroundColor: "#50d3eb" }]}>
      <View style={styles.distance}>
        <Text style={[styles.walkdistanceText, { color: "white" }]}>
          거리: {(distance / 1000).toFixed(1)}km
        </Text>
        <Text style={{ color: "white" }}>{hourormin()}</Text>
      </View>
      <View
        style={{
          flex: 1,
          alignItems: "center",
          justifyContent: "center",
          backgroundColor: "#50d3eb",
        }}
      >
```

- Hours에 저장된 현재 시간 변수를 이용해 시간에 따른 배경색과 글자색의 변화 도출
- 시간이 19시부터 새벽 5시라면 회색 화면과 노란 글씨를, 그 이외의 시간이라면 하늘색 배경과 하얀 글씨를 출력

c) Walkman, BikeMan animation 추가

```jsx
import React, { useRef, useEffect } from "react";
import { StyleSheet, View } from "react-native";
import LottieView from "lottie-react-native";

function WalkMan() {
  const animation = useRef(null);
  useEffect(() => {}, []);

  return (
    <View style={styles.animationContainer}>
      <LottieView
        autoPlay
        loop
        ref={animation}
        style={{
          width: 200,
          height: 200,
          //backgroundColor: "#eee",
        }}
        source={require("../animations/walkman.json")}
      />
    </View>
  );
}

const styles = StyleSheet.create({
  animationContainer: {
    //backgroundColor: "#fff",
    alignItems: "center",
    justifyContent: "center",
    flex: 1,
  },
});

export default WalkMan;
```

```jsx
import React, { useRef, useEffect } from "react";
import { StyleSheet, View } from "react-native";
import LottieView from "lottie-react-native";

function BikeMan() {
  const animation = useRef(null);
  useEffect(() => {}, []);

  return (
    <View style={styles.animationContainer}>
      <LottieView
        autoPlay
        loop
        ref={animation}
        style={{
          width: "100%",
          height: 200,
          //backgroundColor: "#eee",
        }}
        source={require("../animations/bikeman.json")}
      />
    </View>
  );
}

const styles = StyleSheet.create({
  animationContainer: {
    //backgroundColor: "#fff",
    alignItems: "center",
    justifyContent: "center",
    flex: 1,
  },
});

export default BikeMan;
```

- lottie 홈페이지에서 walkscreen,
  bikescreen에 추가하고 싶은 animation.json 파일 다운로드
- 컴포넌트 형식으로 생성하기 위해 Walkman.js, Bikeman.js 파일 생성

```
import BikeMan from "../animeJS/BikeMan";
```

```
import WalkMan from "../animeJS/WalkMan";
```

```
    <Text style={{ fontSize
    <WalkMan />
  </View>
</View>
```

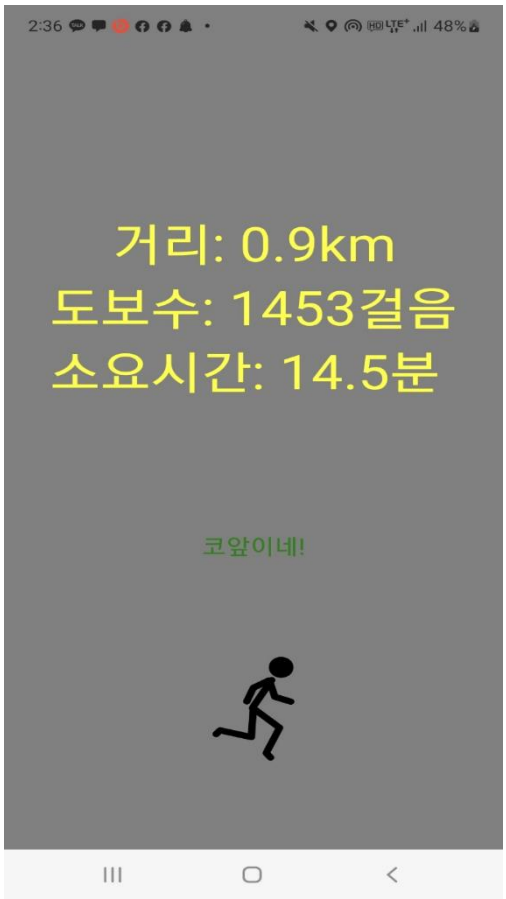
```
    <Text style={{ fontSize: 20
    <BikeMan />
  </View>
</View>
```

- WalkScreen, BikeScreen에서 WalkMan.js, BikeMan.js 파일 import
- <WalkMan/>, <BikeMan/>의 컴포넌트 형식으로 각각의 Screen에 render

d) 실행화면

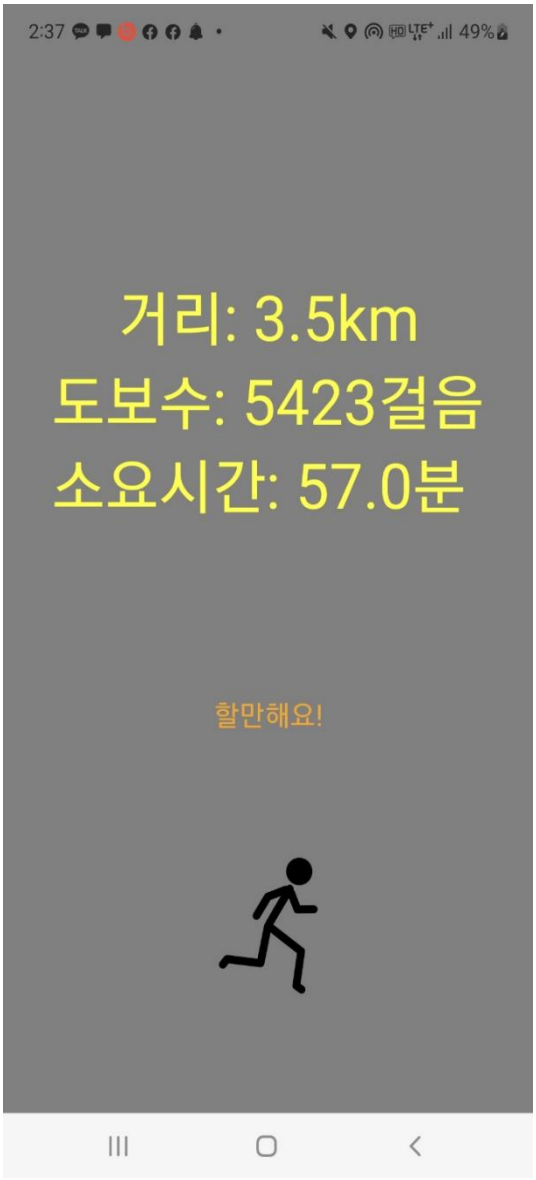
***낮 시간 동안의 실행화면(WalkScreen)**

***저녁,밤 시간 동안의 실행화면(WalkScreen)**



*낮 시간 동안의 BikeScreen

*밤 시간 동안의 BikeScreen



C) 팀원 기여도 및 Github 주소

- **이태화 - DetailScreen 전체, 전달받은 인자로 탑승여부에 따른 디자인, text, 출력**
- **서동원 - MapScreen 전체, 웹 크롤링, 지하철 탑승 여부 파악 하는 알고리즘 작성 후 detailscreen에 필요한 인자 전달,**
- **김건영 - 애니메이션 및 walkscreen, bikescreen 전체, splash screen, loading 화면 구현**

Github 주소

- ZeroKgun/OSCanRide (github.com)