앱개발 3주차



02 오늘 배울 내용

* 리액트 필수지식 활용
* Expo 기능 사용
* 페이지 이동/적용

**리엑트 필수 지식**

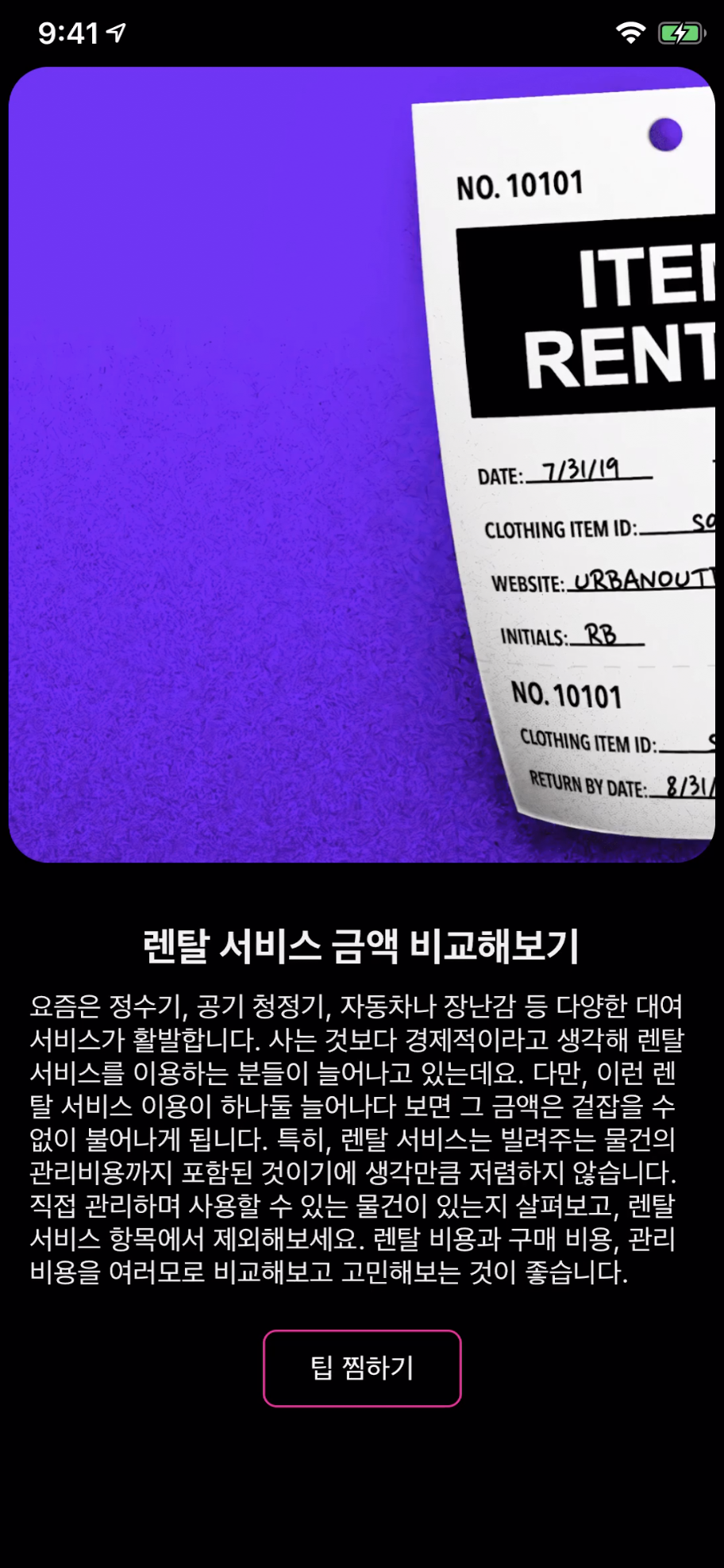
* 컴포넌트(Component)
* 상태(State, useState)
* 속성(props)
* useEffect

목표: 시중의 앱과 유사한 상태의 앱 개발

**Expo 기능 사용**

* 링크, 페이지 이동, 상태바 등 앱 다운 앱 만드는 기능

**[나만의 꿀팁 앱 상세화면 만들어 보기]**



DetailPage.js를 만들고, 이를 호출하여 사용한다

| app.js |
| --- |
| DetailPage를 호출하도록 수정  \* import를 제대로 하지 않을 경우 다음과 같은 에러를 만나게 되는 점에 주의  (import DetailPage from './pages/DetailPage';)  ***ReferenceError: Can't find variable: DetailPage*** |
| Detail.js |
| 페이지 상세 처리 |

간단하다 싶지만 절대 만만하지가 않았다. import 구문을 넣지 않아 1시간 허비ㅠㅠ

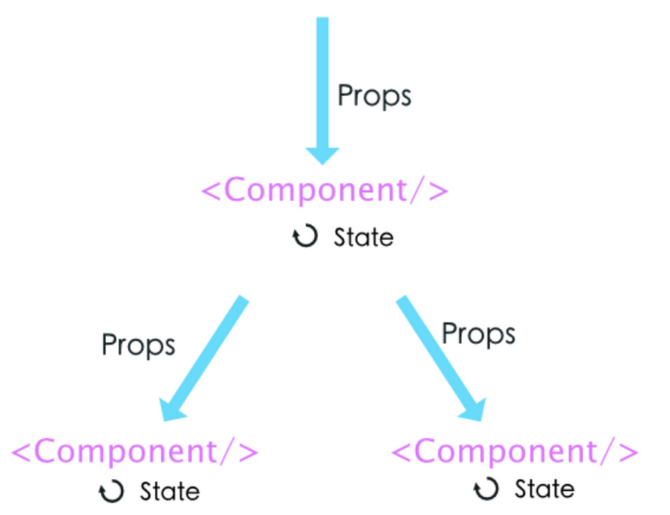
복붙만 하다보면 이런 사소한 실수를 하게 됨을…

완성된 코드는 아래

| **app.js** |
| --- |
| import React from 'react'  import MainPage from './pages/MainPage';  import AboutPage from './pages/AboutPage';  import DetailPage from './pages/DetailPage';  export default function App(){  // return (<MainPage/>)  // return (<AboutPage/>)  return (<DetailPage/>)  } |
| **Detail.js** |
| import React from 'react';  import { StyleSheet, Text, View, Image, ScrollView,TouchableOpacity,Alert } from 'react-native';  export default function DetailPage() {  const tip = {  "idx":9,  "category":"재테크",  "title":"렌탈 서비스 금액 비교해보기",  "image": "https://storage.googleapis.com/sparta-image.appspot.com/lecture/money1.png",  "desc":"요즘은 정수기, 공기 청정기, 자동차나 장난감 등 다양한 대여서비스가 활발합니다. 사는 것보다 경제적이라고 생각해 렌탈 서비스를 이용하는 분들이 늘어나고 있는데요. 다만, 이런 렌탈 서비스 이용이 하나둘 늘어나다 보면 그 금액은 겉잡을 수 없이 불어나게 됩니다. 특히, 렌탈 서비스는 빌려주는 물건의 관리비용까지 포함된 것이기에 생각만큼 저렴하지 않습니다. 직접 관리하며 사용할 수 있는 물건이 있는지 살펴보고, 렌탈 서비스 항목에서 제외해보세요. 렌탈 비용과 구매 비용, 관리 비용을 여러모로 비교해보고 고민해보는 것이 좋습니다. ",  "date":"2020.09.09"  }  const popup = () => {  Alert.alert("팝업!!")  }  return (  // ScrollView에서의 flex 숫자는 의미가 없습니다. 정확히 보여지는 화면을 몇등분 하지 않고  // 화면에 넣은 컨텐츠를 모두 보여주려 스크롤 기능이 존재하기 때문입니다.  // 여기선 내부의 컨텐츠들 영역을 결정짓기 위해서 height 값과 margin,padding 값을 적절히 잘 이용해야 합니다.  <ScrollView style={styles.container}>  <Image style={styles.image} source={{uri:tip.image}}/>  <View style={styles.textContainer}>  <Text style={styles.title}>{tip.title}</Text>  <Text style={styles.desc}>{tip.desc}</Text>  <TouchableOpacity style={styles.button} onPress={()=>popup()}><Text style={styles.buttonText}>팁 찜하기</Text></TouchableOpacity>  </View>    </ScrollView>    )  }  const styles = StyleSheet.create({  container:{  backgroundColor:"#000"  },  image:{  height:400,  margin:10,  marginTop:40,  borderRadius:20  },  textContainer:{  padding:20,  justifyContent:'center',  alignItems:'center'  },  title: {  fontSize:20,  fontWeight:'700',  color:"#eee"  },  desc:{  marginTop:10,  color:"#eee"  },  button:{  width:100,  marginTop:20,  padding:10,  borderWidth:1,  borderColor:'deeppink',  borderRadius:7  },  buttonText:{  color:'#fff',  textAlign:'center'  }  }) |

**[앱 필수 기초지식01]컴포넌트**

**리액트의 기본 구조**



* 컴포넌트(component): 화면 구성요소
* 상태(State): 데이터
* 속성(Props): 태그 속성 및 값,
* useEffect: 화면내 함수처리

컴포넌트는 코드 재사용성을 극대화하는 것으로, 코드 중 일부를 분리해 별도 코드(컴포넌트)로 분리한다. 호출 프로그램도 간단해 지고, 피호출 프로그램(컴포넌트)는 다른 프로젝트에서도 재활용될 수 있도록 하는 것이 컴포넌트의 사용 목적이다.

아래에서 볼 수 있듯, View 컴포넌트 자체를 외부 Card 컴포넌트로 대체되었다.

| <As-Is>  <View style={styles.cardContainer}>  {/\* 하나의 카드 영역을 나타내는 View \*/}  {  tip.map((content,i)=>{  return (<View style={styles.card} key={i}>  <Image style={styles.cardImage} source={{uri:content.image}}/>  <View style={styles.cardText}>  <Text style={styles.cardTitle} numberOfLines={1}>{content.title}</Text>  <Text style={styles.cardDesc} numberOfLines={3}>{content.desc}</Text>  <Text style={styles.cardDate}>{content.date}</Text>  </View>  </View>)  })  }  </View> |
| --- |
| <To-Be>  <View style={styles.cardContainer}>  {/\* 하나의 카드 영역을 나타내는 View \*/}  {  tip.map((content,i)=>{  // return (<View style={styles.card} key={i}>  // <Image style={styles.cardImage} source={{uri:content.image}}/>  // <View style={styles.cardText}>  // <Text style={styles.cardTitle} numberOfLines={1}>{content.title}</Text>  // <Text style={styles.cardDesc} numberOfLines={3}>{content.desc}</Text>  // <Text style={styles.cardDate}>{content.date}</Text>  // </View>  // </View>)  return (<Card content={content} key={i}/>)  })  }    </View> |

**[앱 필수 기초지식 02] 속성(Probs)**

아래와 같이 컴포넌트간 값을 전달하는 방식을 이해할 수 있다.

기존의 객체기반 방식과 동일한 흐름을 보여 주고 있다.

앱 개발이 레고 조립과 같이 쉬워진다고 하는 튜터님의 말씀이 바로 이것이었다.

| <컴포넌트 호출> |
| --- |
| tip.map((content,i)=>{  return (<Card content={content} key={i}/>)  }) |
| < 피호출 컴포넌트> |
| import React from "react"  import {View,Text,Image,StyleSheet} from "react-native";  //비구조 할당 방식으로 넘긴 속성 데이터를 꺼내 사용함  export default function Card({content}) {  return (<View style={styles.card}>  <Image style={styles.cardImage}source={{uri:content.image}}/>  <View style={styles.cardText}>  <Text style={styles.cardTitle} numberOfLines={1}>{content.title}</Text>  <Text style={styles.cardDesc} numberOfLines={3}>{content.desc}</Text>  <Text style={styles.cardDate}>{content.date}</Text>  </View>  </View>)  } |

**[앱 필수 기초지식03] 상태(useState) 와 효과(UseEffect)**

* 상태(State, useState)

React Native에서의 유명한 공식

**UI = Component(state)**

즉, 화면은 데이터(state)가 변경됨에 따라 바뀌는 컴포넌트다.

| **<기존 개발방식, Web 등>**  데이터가 바뀌면 사용자가 계속 Refresh |
| --- |
| **<React Native>**  데이터가 바뀌면 자동으로 바뀌어짐 |

useEffect: 화면이 그려진 다음 가장 먼저 실행되어야 할 코드 작성공간

**useEffect(()=>{**

**“[])**

화면상에서는 이렇게 처리된다. 하지만 실행시 오류가 난다. 이유는?

데이터가 빠져 있다! 해결은 다음 시간에…

| import React,{useState,useEffect} from 'react';  import { StyleSheet, Text, View, Image, TouchableOpacity, ScrollView} from 'react-native';  const main = 'https://storage.googleapis.com/sparta-image.appspot.com/lecture/main.png'  import data from '../data.json';  import Card from '../components/Card';  export default function MainPage() {  //useState 사용법  //[state,setState] 에서 state는 이 컴포넌트에서 관리될 상태 데이터를 담고 있는 변수  //setState는 state를 변경시킬때 사용해야하는 함수  //모두 다 useState가 선물해줌  //useState()안에 전달되는 값은 state 초기값  const [state,setState] = useState([])    //하단의 return 문이 실행되어 화면이 그려진다음 실행되는 useEffect 함수  //내부에서 data.json으로 부터 가져온 데이터를 state 상태에 담고 있음  useEffect(()=>{  setState(data)  },[])  // let tip = data.tip;  //data.json 데이터는 state에 담기므로 상태에서 꺼내옴  let tip = state.tip; |
| --- |

**[앱 필수기초지식 응용]컴포넌트와 상태를 이용한 로딩화면 만들기**

일단 바꾸니 Main.js를 보면

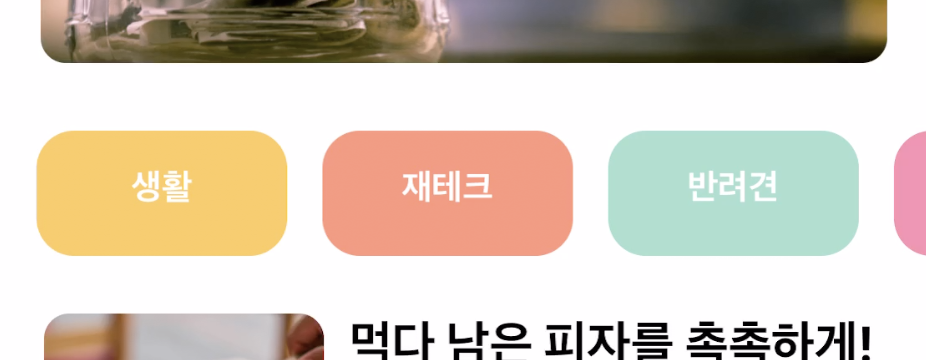
| import React,{useState,useEffect} from 'react';  import { StyleSheet, Text, View, Image, TouchableOpacity, ScrollView} from 'react-native';  const main = 'https://storage.googleapis.com/sparta-image.appspot.com/lecture/main.png'  import data from '../data.json';  import Card from '../components/Card';  import Loading from '../components/Loading';  export default function MainPage() {  //useState 사용법  //[state,setState] 에서 state는 이 컴포넌트에서 관리될 상태 데이터를 담고 있는 변수  //setState는 state를 변경시킬때 사용해야하는 함수  //모두 다 useState가 선물해줌  //useState()안에 전달되는 값은 state 초기값  const [state,setState] = useState([])    //하단의 return 문이 실행되어 화면이 그려진다음 실행되는 useEffect 함수  //내부에서 data.json으로 부터 가져온 데이터를 state 상태에 담고 있음  const [ready,setReady] = useState(true)  useEffect(()=>{    //뒤의 1000 숫자는 1초를 뜻함  //1초 뒤에 실행되는 코드들이 담겨 있는 함수  setTimeout(()=>{  setState(data)  setReady(false)  },1000)      },[])  //data.json 데이터는 state에 담기므로 상태에서 꺼내옴  let tip = state.tip;  let todayWeather = 10 + 17;  let todayCondition = "흐림"  //return 구문 밖에서는 슬래시 두개 방식으로 주석  return ready ? <Loading/> : (  /\*  return 구문 안에서는 {슬래시 + \* 방식으로 주석  \*/  <ScrollView style={styles.container}>  <Text style={styles.title}>나만의 꿀팁</Text>  <Text style={styles.weather}>오늘의 날씨: {todayWeather + '°C ' + todayCondition} </Text>  <Image style={styles.mainImage} source={{uri:main}}/>  <ScrollView style={styles.middleContainer} horizontal indicatorStyle={"white"}>  <TouchableOpacity style={styles.middleButton01}><Text style={styles.middleButtonText}>생활</Text></TouchableOpacity>  <TouchableOpacity style={styles.middleButton02}><Text style={styles.middleButtonText}>재테크</Text></TouchableOpacity>  <TouchableOpacity style={styles.middleButton03}><Text style={styles.middleButtonText}>반려견</Text></TouchableOpacity>  <TouchableOpacity style={styles.middleButton04}><Text style={styles.middleButtonText}>꿀팁 찜</Text></TouchableOpacity>  </ScrollView>  <View style={styles.cardContainer}>  {/\* 하나의 카드 영역을 나타내는 View \*/}  {  tip.map((content,i)=>{  return (<Card content={content} key={i}/>)  })  }    </View>    </ScrollView>)  }  const styles = StyleSheet.create({  …중략…  }); |
| --- |

다음과 같은 순서로 처리가 된다

| 1. setReady 수행 |
| --- |
| const [ready,setReady] = useState(true) |
| 2. <Loading/> 함수 실행(ready == true이므로, Loading.js를 import하였고. 이를 화면에 출력) |
| import Loading from '../components/Loading';  return ready ? <Loading/> : ( |
| 3. 화면이 그려지고 난 다음, useEffect() 함수가 실행됨. 즉, 1초 후 state 데이터가 채워지고, 변경된다(setState(data))  변경이 완료되면 화면이 다시 컴포넌트 호출(setReady(false)) |
| useEffect(()=>{    //뒤의 1000 숫자는 1초를 뜻함  //1초 뒤에 실행되는 코드들이 담겨 있는 함수  setTimeout(()=>{  setState(data)  setReady(false)  },1000) |
| 4. 화면이 다시 실행될 때(setReady(false)) ready값이 false이므로, return ready ? <Loading/> : ( 즉, <Loading/> 대신 이후 코드들이 수행됨(화면에 뿌려줌) |
| return ready ? <Loading/> : (  /\*  return 구문 안에서는 {슬래시 + \* 방식으로 주석  \*/  <ScrollView style={styles.container}>  <Text style={styles.title}>나만의 꿀팁</Text>  <Text style={styles.weather}>오늘의 날씨: {todayWeather + '°C ' + todayCondition} </Text>  <Image style={styles.mainImage} source={{uri:main}}/>  <ScrollView style={styles.middleContainer} horizontal indicatorStyle={"white"}>  <TouchableOpacity style={styles.middleButton01}><Text style={styles.middleButtonText}>생활</Text></TouchableOpacity>  <TouchableOpacity style={styles.middleButton02}><Text style={styles.middleButtonText}>재테크</Text></TouchableOpacity>  <TouchableOpacity style={styles.middleButton03}><Text style={styles.middleButtonText}>반려견</Text></TouchableOpacity>  <TouchableOpacity style={styles.middleButton04}><Text style={styles.middleButtonText}>꿀팁 찜</Text></TouchableOpacity>  </ScrollView>  <View style={styles.cardContainer}>  {/\* 하나의 카드 영역을 나타내는 View \*/}  {  tip.map((content,i)=>{  return (<Card content={content} key={i}/>)  })  }    </View>    </ScrollView>) |

**[앱 필수지식응용] state를 이용한 카테고리 기능 넣기**

이제 버튼을 눌러 다른 데이터를 화면에 뿌려주는 실습이다.



바뀐 코드는 다음과 같다.

| import React,{useState,useEffect} from 'react';  import { StyleSheet, Text, View, Image, TouchableOpacity, ScrollView} from 'react-native';  const main = 'https://storage.googleapis.com/sparta-image.appspot.com/lecture/main.png'  import data from '../data.json';  import Card from '../components/Card';  import Loading from '../components/Loading';  export default function MainPage() {  //useState 사용법  //[state,setState] 에서 state는 이 컴포넌트에서 관리될 상태 데이터를 담고 있는 변수  //setState는 state를 변경시킬때 사용해야하는 함수  //모두 다 useState가 선물해줌  //useState()안에 전달되는 값은 state 초기값  const [state,setState] = useState([])  const [cateState,setCateState] = useState([])  //하단의 return 문이 실행되어 화면이 그려진다음 실행되는 useEffect 함수  //내부에서 data.json으로 부터 가져온 데이터를 state 상태에 담고 있음  const [ready,setReady] = useState(true)  useEffect(()=>{    //뒤의 1000 숫자는 1초를 뜻함  //1초 뒤에 실행되는 코드들이 담겨 있는 함수  setTimeout(()=>{  setState(data.tip)  setCateState(data.tip)  setReady(false)  },1000)      },[])  const category = (cate) => {  if(cate == "전체보기"){  //전체보기면 원래 꿀팁 데이터를 담고 있는 상태값으로 다시 초기화  setCateState(state)  }else{  setCateState(state.filter((d)=>{  return d.category == cate  }))  }  }  //data.json 데이터는 state에 담기므로 상태에서 꺼내옴  // let tip = state.tip;  let todayWeather = 10 + 17;  let todayCondition = "흐림"  //return 구문 밖에서는 슬래시 두개 방식으로 주석  return ready ? <Loading/> : (  /\*  return 구문 안에서는 {슬래시 + \* 방식으로 주석  \*/  <ScrollView style={styles.container}>  <Text style={styles.title}>나만의 꿀팁</Text>  <Text style={styles.weather}>오늘의 날씨: {todayWeather + '°C ' + todayCondition} </Text>  <Image style={styles.mainImage} source={{uri:main}}/>  <ScrollView style={styles.middleContainer} horizontal indicatorStyle={"white"}>  <TouchableOpacity style={styles.middleButtonAll} onPress={()=>{category('전체보기')}}><Text style={styles.middleButtonTextAll}>전체보기</Text></TouchableOpacity>  <TouchableOpacity style={styles.middleButton01} onPress={()=>{category('생활')}}><Text style={styles.middleButtonText}>생활</Text></TouchableOpacity>  <TouchableOpacity style={styles.middleButton02} onPress={()=>{category('재테크')}}><Text style={styles.middleButtonText}>재테크</Text></TouchableOpacity>  <TouchableOpacity style={styles.middleButton03} onPress={()=>{category('반려견')}}><Text style={styles.middleButtonText}>반려견</Text></TouchableOpacity>  <TouchableOpacity style={styles.middleButton04} onPress={()=>{category('꿀팁 찜')}}><Text style={styles.middleButtonText}>꿀팁 찜</Text></TouchableOpacity>  </ScrollView>  <View style={styles.cardContainer}>  {/\* 하나의 카드 영역을 나타내는 View \*/}  {  cateState.map((content,i)=>{  return (<Card content={content} key={i}/>)  })  }    </View>    </ScrollView>)  } |
| --- |

컴포넌트는 다음과 같이 수행된다

| 초기 로딩시에 카테고리 처리가 추가되었다 |
| --- |
| const [state,setState] = useState([])  const [cateState,setCateState] = useState([]) |
| 그리고, useEffect에도 전체 카테고리내용을 표시하는 기능이 추가되었다 (setCateState(data.tip)) |
| useEffect(()=>{    //뒤의 1000 숫자는 1초를 뜻함  //1초 뒤에 실행되는 코드들이 담겨 있는 함수  setTimeout(()=>{  setState(data.tip)  setCateState(data.tip)  setReady(false)  },1000) |
| 버튼 클릭시 호출 부분이 수정되었다 |
| <TouchableOpacity style={styles.middleButton02} onPress={()=>{category('재테크')}}><Text style={styles.middleButtonText}>재테크</Text></TouchableOpacity> |
| 호출시 수행되는 함수가 추가되었다 |
| const category = (cate) => {  if(cate == "전체보기"){  //전체보기면 원래 꿀팁 데이터를 담고 있는 상태값으로 다시 초기화  setCateState(state)  }else{  setCateState(state.filter((d)=>{  return d.category == cate  }))  }  } |

즉, 화면이 로딩되고 나면 전체 카테고리 리스트가 화면에 나타나게 되는 것이다.

버튼이 선택되면, 개별 버튼별로 지정된 category 함수를 호출하게 되고,

category 함수는 전체보기일 경우 전체 데이터를, 아닐 경우에는 해당 카테고리(cate)와 데이터의 category를 비교한 후 해당 데이터만을 선택(filter)하여 보여주게 되는 것이다.

**[Expo 앱다운 앱기능]앱 상태바(Status Bar) 관리**

이번에는 Expo 상태바를 중심으로 앱 다운 앱을 구현하는 Expo의 기능을 배우게 된다.

**Expo 상태바 설치**

우선 터미널을 통해 Expo 상태바를 설치하여야 한다.

**이 때, 반드시 해당 프로젝트 폴더로 이동한 후 설치하여야 한다는 점에 주의해야 한다.**

| **C:\Users\kkk\Desktop\sparta\_study\(프로젝트 폴더)>expo install expo-status-bar** |
| --- |

.

**Main Page에 상태바 코드 추가**

| 상태바 import |
| --- |
| import { StatusBar } from 'expo-status-bar'; |
| View에 추가 |
| <ScrollView style={styles.container}>  **<StatusBar style="light" />**  <Text style={styles.title}>나만의 꿀팁</Text> |

**[앱페이지 적용]네비게이터**

네비게이터는 페이지간 이동을 다룬다

**네비게이션 설치**

터미널을 이용하여 네비게이션과 네비게이션 추가 설치를 한다

| **네비게이션 설치** |
| --- |
| C:\Users\kkk\Desktop\sparta\_study\(프로젝트폴더)>yarn add @react-navigation/native |
| **네비게이션 추가설치** |
| C:\Users\kkk\Desktop\sparta\_study\(프로젝트 폴더)>expo install react-native-gesture-handler react-native-reanimated react-native-screens react-native-safe-area-context @react-native-community/masked-view |

**[앱페이지 적용]설치 - 스택 네비게이션01**

스택 네비게이션은 페이지를 처리하고, 페이지간 이동을 처리하는 네비게이션

스택(Stack)은 페이지를 쌓아 놓는 것. (history.back())

**스택 설치**

터미널을 통해 스택을 설치한다

| C:\Users\kkk\Desktop\sparta\_study\(프로젝트 폴더)>yarn add @react-navigation/stack |
| --- |

네비게이션 처리시 다음과 같이 바뀌게 된다

| **StackNavigator.js**  모든 페이지를 이 파일에서 지정하게 된다. |
| --- |
| import React from 'react';  //설치한 스택 네비게이션 라이브러리를 가져옵니다  import { createStackNavigator } from '@react-navigation/stack';  //페이지로 만든 컴포넌트들을 불러옵니다  import DetailPage from '../pages/DetailPage';  import MainPage from '../pages/MainPage';  //스택 네비게이션 라이브러리가 제공해주는 여러 기능이 담겨있는 객체를 사용합니다  //그래서 이렇게 항상 상단에 선언하고 시작하는게 규칙입니다!  const Stack = createStackNavigator();  const StackNavigator = () =>{  return (  //컴포넌트들을 페이지처럼 여기게끔 해주는 기능을 하는 네비게이터 태그를 선언합니다.  //위에서 선언한 const Stack = createStackNavigator(); Stack 변수에 들어있는 태그를 꺼내 사용합니다.  //Stack.Navigator 태그 내부엔 페이지(화면)를 스타일링 할 수 있는 다양한 옵션들이 담겨 있습니다.  <Stack.Navigator  screenOptions={{  headerStyle: {  backgroundColor: "black",  borderBottomColor: "black",  shadowColor: "black",  height:100  },  headerTintColor: "#FFFFFF",  headerBackTitleVisible: false  }}    >  {/\* 컴포넌트를 페이지로 만들어주는 엘리먼트에 끼워 넣습니다. 이 자체로 이제 페이지 기능을 합니다\*/}  <Stack.Screen name="MainPage" component={MainPage}/>  <Stack.Screen name="DetailPage" component={DetailPage}/>  </Stack.Navigator>  )  }  export default StackNavigator;  **App.js**  이제 페이지 처리는 StackNavigation에서 하게 되고, App.js는 StackNavigator를 지정할 뿐, 수정할 필요가 없어진다. |
| import React from 'react';  //이제 모든 페이지 컴포넌트들이 끼워져있는 책갈피를 메인에 둘예정이므로  //컴포넌트를 더이상 불러오지 않아도 됩니다.  // import MainPage from './pages/MainPage';  // import DetailPage from './pages/DetailPage';  import { StatusBar } from 'expo-status-bar';  //메인에 세팅할 네비게이션 도구들을 가져옵니다.  import {NavigationContainer} from '@react-navigation/native';  import StackNavigator from './navigation/StackNavigator'  export default function App() {  return (  <NavigationContainer>  <StatusBar style="black" />  <StackNavigator/>  </NavigationContainer>);  } |

네비게이션을 처리하고 나면 화면은 다음과 같이 보여진다



**[앱 페이지 적용]페이지 헤더 스타일 수정**

상단의 Mainpage 헤더부분을 변경할 수도 있다.

| 바뀐 StackNavigator.js |
| --- |
| <Stack.Navigator  screenOptions={{  headerStyle: {  backgroundColor: "white",  borderBottomColor: "white",  shadowColor: "white",  height:100  },  //헤더의 텍스트를 왼쪾에 둘지 가운데에 둘지를 결정  headerTitleAlign:'left',  headerTintColor: "#000",  headerBackTitleVisible: false  }} |
| Mainpage.js 페이지 타이틀 수정(remark) |
| {/\* <Text style={styles.title}>나만의 꿀팁</Text> \*/} |

**[앱 페이지 적용]페이지 이동하기- 스택 네비게이션 03**

스택 네비게이션을 처리할 때는 다음과 같은 2개의 딕셔너리를 이용한다

* Navigation
* route

일반적으로 페이지 컴포넌트들을 네비게이션으로 페이지간 이동을 하게 되지만, 함수간 파라미터 전달을 통해 네비게이션 기능을 이용할 수 있다.

| 페이지 내의 호출부분 변경이 다음과 같이 변경된다.  즉, navigation과 route를 전달받는다 |
| --- |
| export default function MainPage({navigation,route}) { |
| navigation을 이용하여 페이지 타이틀 처리방법도 달라졌다(navigation.setOptions) |
| useEffect(()=>{    //뒤의 1000 숫자는 1초를 뜻함  //1초 뒤에 실행되는 코드들이 담겨 있는 함수  setTimeout(()=>{  //헤더의 타이틀 변경  navigation.setOptions({  title:'나만의 꿀팁'  })  setState(data.tip)  setCateState(data.tip)  setReady(false)  },1000)  },[]) |
| StackNavigation에 등록되지 않은 페이지도 navigation을 이용할 수 있다.  다만, 호출시 navigation을 함께 전달해야 한다  아래는 Card 선택시 호출되는 Card.js를 호출하는 함수이다. |
| <Card content={content} key={i} navigation={navigation}/> |
| 피호출 페이지에서는  전달받은 navigation을 이용하여 다른 페이지를 또 호출할 수 있다  아래는 Card.js에서 카드 선택시 상세 페이지(DetailPage)로 이동함을 알 수 있다  navigation.navigate('DetailPage') |
| export default function Card({content,navigation}){  return(  //카드 자체가 버튼역할로써 누르게되면 상세페이지로 넘어가게끔 TouchableOpacity를 사용  <TouchableOpacity style={styles.card} onPress={()=>{navigation.navigate('DetailPage')}}>  <Image style={styles.cardImage} source={{uri:content.image}}/>  <View style={styles.cardText}>  <Text style={styles.cardTitle} numberOfLines={1}>{content.title}</Text>  <Text style={styles.cardDesc} numberOfLines={3}>{content.desc}</Text>  <Text style={styles.cardDate}>{content.date}</Text>  </View>  </TouchableOpacity>  )  } |
| DetailPage에서는 Navigation을 이용하여 페이지 뒤로 가기도 가능해진다  (DetainPage.js에서는 다르게 Navigation처리도 하지 않았다) |

**페이지 이동시 데이터 전달하기**

Card.js 페이지와 DetailPage.js 페이지간 이동을 살펴본다

| Card.js에서는 상세 내용 선택시 DetailPage.js 페이지를 호출하게 된다.  이 경우, MainPage.js에서 전달받았던 Content 딕셔너리(상세 내용 포함)를 함께 전달한다 |
| --- |
| <TouchableOpacity style={styles.card} onPress={()=>{navigation.navigate('DetailPage',content)}}>  <Image style={styles.cardImage} source={{uri:content.image}}/>  <View style={styles.cardText}>  <Text style={styles.cardTitle} numberOfLines={1}>{content.title}</Text>  <Text style={styles.cardDesc} numberOfLines={3}>{content.desc}</Text>  <Text style={styles.cardDate}>{content.date}</Text>  </View>  </TouchableOpacity> |
| DetailPage.js  아래 사항들을 유념해서 볼 필요가 있다  1. 외부 페이지(card.js)로부터 navigation과 route를 전달받았다  **DetailPage({navigation,route})**  2. 에러 방지를 위해 tip 딕셔너리를 저장하고 있다. (앞에서 설명했듯 state에 초기값이 없으면 에러가 나기 때문에 초기값을 지정해 놓았다. state는 추후 content값으로 대체되게 된다. 페이지 처리 순서에 따라 처리되기 때문  3. 페이지 타이틀 처리도 달라졌다(title:route.params.title)  페이지 타이틀도 전달받은 content 딕셔너리(즉, content = route.params)의 title을 페이지 타이틀로 쓸 수 있다. 즉 전달받은 데이터를 활용할 수 있다.  4. 네비게이션과는 별도로 자체 페이지 꾸미기도 가능하다.(headerStyle: {)  전체 페이지 설정은 StackNavigation.js에서 지정이 가능하나, 개별 페이지별로도 설정할 수 있다는 것을 알 수 있다. |
| export default function DetailPage({navigation,route}) {    //초기 컴포넌트의 상태값을 설정  //state, setState 뿐 아니라 이름을 마음대로 지정할 수 있음!  const [tip, setTip] = useState({  "idx":9,  "category":"재테크",  "title":"렌탈 서비스 금액 비교해보기",  "image": "https://storage.googleapis.com/sparta-image.appspot.com/lecture/money1.png",  "desc":"요즘은 정수기, 공기 청정기, 자동차나 장난감 등 다양한 대여서비스가 활발합니다. 사는 것보다 경제적이라고 생각해 렌탈 서비스를 이용하는 분들이 늘어나고 있는데요. 다만, 이런 렌탈 서비스 이용이 하나둘 늘어나다 보면 그 금액은 겉잡을 수 없이 불어나게 됩니다. 특히, 렌탈 서비스는 빌려주는 물건의 관리비용까지 포함된 것이기에 생각만큼 저렴하지 않습니다. 직접 관리하며 사용할 수 있는 물건이 있는지 살펴보고, 렌탈 서비스 항목에서 제외해보세요. 렌탈 비용과 구매 비용, 관리 비용을 여러모로 비교해보고 고민해보는 것이 좋습니다. ",  "date":"2020.09.09"  })    useEffect(()=>{  console.log(route)  //Card.js에서 navigation.navigate 함수를 쓸때 두번째 인자로 content를 넘겨줬죠?  //content는 딕셔너리 그 자체였으므로 route.params에 고대~로 남겨옵니다.  //즉, route.params 는 content죠!  navigation.setOptions({  //setOptions로 페이지 타이틀도 지정 가능하고  title:route.params.title,  //StackNavigator에서 작성했던 옵션을 다시 수정할 수도 있습니다.  headerStyle: {  backgroundColor: '#000',  shadowColor: "#000",  },  headerTintColor: "#fff",  })  setTip(route.params)  },[])  const popup = () => {  Alert.alert("팝업!!")  }  return (  // ScrollView에서의 flex 숫자는 의미가 없습니다. 정확히 보여지는 화면을 몇등분 하지 않고  // 화면에 넣은 컨텐츠를 모두 보여주려 스크롤 기능이 존재하기 때문입니다.  // 여기선 내부의 컨텐츠들 영역을 결정짓기 위해서 height 값과 margin,padding 값을 적절히 잘 이용해야 합니다.  <ScrollView style={styles.container}>  <Image style={styles.image} source={{uri:tip.image}}/>  <View style={styles.textContainer}>  <Text style={styles.title}>{tip.title}</Text>  <Text style={styles.desc}>{tip.desc}</Text>  <TouchableOpacity style={styles.button} onPress={()=>popup()}><Text style={styles.buttonText}>팁 찜하기</Text></TouchableOpacity>  </View>    </ScrollView>    )  } |

**[Expo 앱다운 앱기능02]페이지 내용 공유하기**

페이지 공유하기 기능을 쉽게 적용이 가능하다

공유기능을 이용하려면

* import share
* share 함수 처리
* 공유버튼 처리

를 해야 한다.

| <Share 기본 함수 import>  import { StyleSheet, Text, View, Image, ScrollView,TouchableOpacity,Alert,Share } from 'react-native';  <함수 설정>  const share = () => {  Share.share({  message:`${tip.title} \n\n ${tip.desc} \n\n ${tip.image}`,  });  }  <호출하기>  <TouchableOpacity style={styles.button} onPress={()=>share()}><Text style={styles.buttonText}>팁 공유하기</Text></TouchableOpacity> |
| --- |

**[Expo 앱다운 앱기능02]외부 링크 클릭**

expo-linking 도구 설치

터미널을 이용하여 expo-linking을 설치한다

| C:\Users\kkk\Desktop\sparta\_study\(프로젝트 폴더)>expo install expo-linking |
| --- |

페이지에 적용하는 방법은 간단하다

| **함수 선언**  외부 링크 페이지를 함수로 선언해 준다 |
| --- |
| const link = () => { Linking.openURL("https://spartacodingclub.kr") } |
| **버튼 링크 적용**  해당 버튼에 선언한 함수를 호출한다 |
| <TouchableOpacity style={styles.button} onPress={()=>link()}><Text style={styles.buttonText}>외부 링크</Text></TouchableOpacity> </View> |

**[숙제: AboutPage 링크달기, 꿀팁 찜 페이지 추가]**

**1. AboutPage.js 링크 달고 외부 링크 추가하기**

React Navigation을 이용하여 해당 페이지 링크를 단 후 버튼에 외부 링크를 추가한다

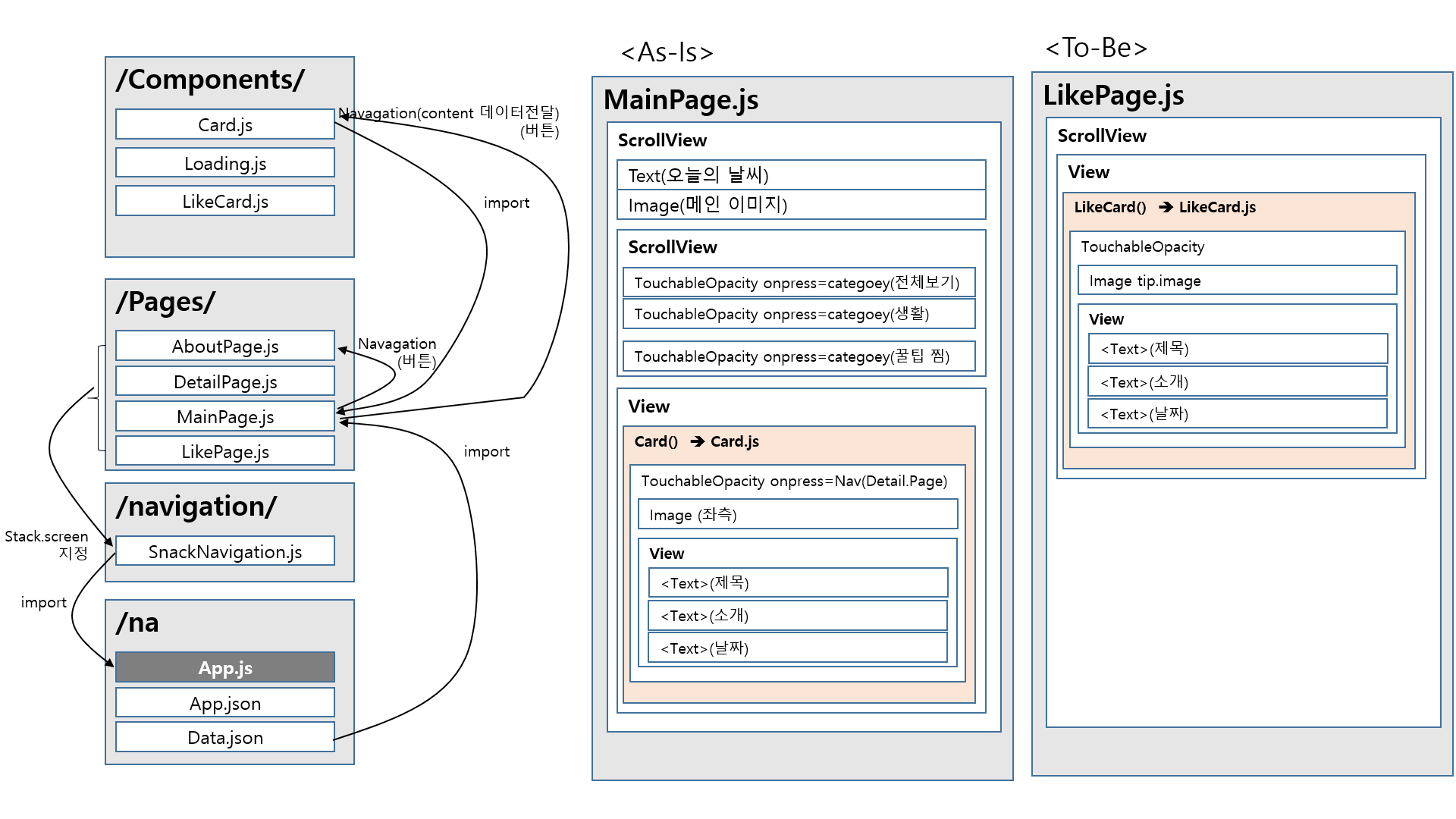
| (MainPage.js)1. MainPage 링크 수정  메인페이지 내 “소개 페이지” 링크를 네비게이션 처리로 변경한다 |
| --- |
| <TouchableOpacity style={styles.aboutButton} onPress={()=>{navigation.navigate('AboutPage')}}><Text style={styles.aboutButtonText}>소개 페이지</Text></TouchableOpacity> |
| 2. StackNavigation.js 에 AbouPage.js 페이지를 추가한다(StackNavigation.js)  import 후 screen을 추가해야 함에 유의 |
| import AboutPage from '../pages/AboutPage';  <Stack.Screen name="AboutPage" component={AboutPage}/> |
| 3. Aboutpage 호출함수 변경(MainPage.js)  Navigation을 이용할 경우 호출함수를 바꾸어주어야 한다 |
| export default function AboutPage({navigation,route}) { |
| 4. 외부 링크 함수 추가(MainPage.js) |
| const link = () => {  Linking.openURL("https://jason2022.tistory.com")  } |
| 5. 버튼 링크 변경(MainPage.js) |
| <TouchableOpacity style={styles.button} onPress={()=>link()}>  <Text style={styles.buttonText}>나의 블로그계정</Text>  </TouchableOpacity> |

**2. 꿀팁 찜 페이지 추가하기**

메인 페이지의 “꿀팁 찜” 버튼 선택시 “꿀팁 찜” 페이지가 나타나게 하는, 기능은 참으로 간단하다.

하지만, 3주차에 학습한 거의 모든 내용을 이해해야만 개발이 가능했다.

결국 프로젝트 내 모든 페이지 네비게이션을 분석하고야 말았다



복붙만 하다보니 에러가 속출했다. 어슬프게 이해한 탓이리라.

결국 3주차 전 과정을 재학습할 수 밖에 없었다.

그러고 나니 어느 정도 이해할 수 있었고, 다행히 정상적으로 완료할 수 있었다.

| **1. LikePage.js 코드 추가**  LikePage.js(꿀팁 찜) 페이지 코드를 추가한다 |
| --- |
| **import React,{useState,useEffect} from 'react';**  **import { StyleSheet, Text, View, Image, TouchableOpacity, ScrollView} from 'react-native';**  **import LikeCard from '../components/LikeCard';**  **import Loading from '../components/Loading';**  **import { StatusBar } from 'expo-status-bar';**  **export default function LikePage({navigation,route}) {**  **//data.json 데이터는 state에 담기므로 상태에서 꺼내옴**  **const [tip, setTip] = useState([{**  **"idx":3,**  **"category":"재테크",**  **"title":"잠자는 내 돈을 찾아라",**  **"image": "https://storage.googleapis.com/sparta-image.appspot.com/lecture/money1.png",**  **"desc":"‘새는 돈’에는 미처 몰랐던 카드 포인트, 휴면예금이나 환급금도 포함됩니다. 확실히 파악하지 못한 잠자는 돈을 찾아보고 자투리 돈들을 모으는 것도 중요합니다. 케이블방송, 위성방송 서비스를 이용하면서 중복 납부한 요금, 셋톱박스 보증금 등 돌려받지 않은 돈이 있는지 확인 해보세요. 또, 카드 포인트 통합 조회 서비스를 이용해 여러 개의 카드 포인트가 모두 얼마인지 체크해두는 것이 좋습니다. 보험해약 환급금, 휴면 보험금이나 휴면 예금을 찾아보고 돌려받는 일도 요즘에는 어렵지 않습니다.",**  **"date":"2020.09.09"**  **},**  **{**  **"idx":4,**  **"category":"재테크",**  **"title":"할인행사, 한정할인판매 문구의 함정 탈출!",**  **"image": "https://storage.googleapis.com/sparta-image.appspot.com/lecture/money2.png",**  **"desc":"‘안 사면 100% 할인’이라는 말 들어보셨나요? 견물생심, 좋은 물건을 보면 사고 싶기 마련입니다. 특히 대대적인 ‘할인 행사’ 중인 대형 마트에 갔을 때는 말할 것도 없겠죠. 따라서 생필품을 살 때, 한꺼번에 사서 사용하는 것보다 필요할 때 조금씩 구매하는 편이 좋습니다. 장을 보면서 대형마트에 자주 가다 보면 지금 필요한 것뿐 아니라 앞으로 필요할 것까지 사게 되어 지출이 커지기 때문입니다. 특히 할인 품목을 보면 뜻하지 않은 소비를 하는 경우도 많아진다. 홈쇼핑, 대형마트 등의 ‘할인행사’, ‘한정할인판매’ 등의 문구를 조심하세요. ",**  **"date":"2020.09.09"**  **}**  **])**  **useEffect(()=>{**    **//뒤의 1000 숫자는 1초를 뜻함**  **//1초 뒤에 실행되는 코드들이 담겨 있는 함수**  **setTimeout(()=>{**  **//헤더의 타이틀 변경**  **navigation.setOptions({**  **title:'꿀팁 찜'**  **})**  **setTip(tip)**  **setReady(false)**  **},1000)**      **},[])**  **//return 구문 밖에서는 슬래시 두개 방식으로 주석**  **return ready ? <Loading/> : (**  **/\***  **return 구문 안에서는 {슬래시 + \* 방식으로 주석**  **\*/**  **<ScrollView style={styles.container}>**  **<View style={styles.cardContainer}>**  **{/\* 하나의 카드 영역을 나타내는 View \*/}**  **{**  **tip.map((content,i)=>{**  **return (<LikeCard content={content} key={i} navigation={navigation}/>)**  **})**  **}**    **</View>**    **</ScrollView>)**  **}** |
| **2.꿀팀 찜 페이지, 네비게이션 추가**  StackNavigation.js에 LikePage.js 네비게이션을 추가한다 |
| **import LikePage from '../pages/LikePage';**  **<Stack.Screen name="LikePage" component={LikePage}/>** |
| **3. MainPage.js 버튼 수정**  메인페이지 내 “꿀팁 찜” 버튼 링크를 변경한다 |
| <TouchableOpacity style={styles.middleButton04} onPress={()=>{navigation.navigate('LikePage')}}><Text style={styles.middleButtonText}>꿀팁 찜</Text></TouchableOpacity> |
| 4. LikeCard.js 페이지 추가  LikePage.js 에서 호출할 꿀팁 찜 View 페이지를 추가한다 |
| import React from 'react';  import {View, Image, Text, StyleSheet,TouchableOpacity} from 'react-native'  //MainPage로 부터 navigation 속성을 전달받아 Card 컴포넌트 안에서 사용  export default function LikeCard({content,navigation}){  return(  //카드 자체가 버튼역할로써 누르게되면 상세페이지로 넘어가게끔 TouchableOpacity를 사용  <TouchableOpacity style={styles.card}>  <Image style={styles.cardImage} source={{uri:content.image}}/>  <View style={styles.cardText}>  <Text style={styles.cardTitle} numberOfLines={1}>{content.title}</Text>  <Text style={styles.cardDesc} numberOfLines={3}>{content.desc}</Text>  <Text style={styles.cardDate}>{content.date}</Text>  </View>  </TouchableOpacity>  )  } |

**3주차 수강 후기**

쉽지는 않겠거니 생각했지만 과연 쉽지는 않았다.

하지만 Expo 도구, React, React Native, Library, API Reference 등 앱 개발 도구들을 활용하여 코딩과 UI를 개발하는 것이 훨씬 수월해졌음을 체감할 수 있었다.

더군다나 Microsoft의 VisualCode 도구 또한 단단히 한 몫을 하고 있었음을 알 수 있었다.

또한, 튜터님은 짧은 시간 내에 많은 내용을 학습할 수 있게 학습과정을 정교하게 구성하였음을 알 수 있었다.

어쩔 수 없이 지난 과정을 반복 학습할 수 밖에 없게끔 숙제를 구성한 것도 흥미로웠다.

아직은 수박 겉핥기 식으로 학습하고 있지만, 새로운 개발 언어를 학습한다는 것 자체가 큰 변화를 수반할 수 밖에 없다는 것 또한 익히 알고 있던 터.

하지만 수년 전 개발 패턴을 고수하고 있던 나로서는 큰 변화가 오고 있음을 직감할 수 있었다.

지금까지의 학습과정을 살펴볼 때, 기존에 개발했던 프로젝트들은 거의 재개발 수준으로 개발해야 할 것 같다.

하지만, 코드 중복, 생산성 등을 감안할 때 가성비가 획기적으로 줄어들 것임에 틀림없어 보인다.

Facebook, Instagram이 학습한 React Native로 개발되었음을 감안할 때, Meta(구 Facebook)기 폐업하지 않는 한은 React Native는 계속 발전되어 나갈 것으로 보인다.

멈추었던 걸음을 다시 한번 재촉해 본다