React Native

expo-router의 Stack, Tab Nagivation

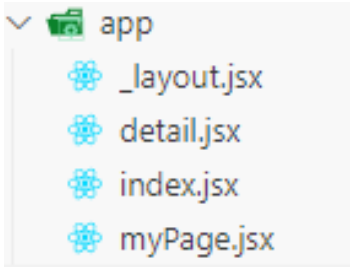
React Native | 주의사항

우리는 React Native 프로젝트 생성 시 'npx create-expo-app@latest'라는 명령어를 사용하여 프로젝트를 생성한다.
명령어에 보면 'expo'라는 단어가 보이는데, 이는 react native 프로젝트를 생성하는 두 가지 방법 중 expo를 사용하여 프로젝트를 만든다는 것을 뜻한다.
native 프로젝트를 생성하는 방법은 위에서 이야기한 expo를 사용하는 방식과 cli를 이용한 방식(순수한 react native 프로젝트 생성)이 있다.
cli를 이용하여 react native 프로젝트를 생성하면 기능 구현을 위해 필요한 라이브러리 설치, 라이브러리 버전 관리 등 프로젝트의 전반적인 모든 설정을 개발자가 해줘야 한다. 단, 이렇게 고생해서 프로젝트를 생성하면 앱의 고유한 기능(카메라, 자이로 센서 등)을 활용하여 기능을 개발하기 용이하다.
expo를 이용하여 프로젝트를 만들면 cli를 이용해 만들 때 개발자가 해줘야 했던 모든 프로젝트의 설정이 갖춰진채로 프로젝트가 만들어진다. 예전 버전에서는 expo로 만든 프로젝트에서 휴대폰의 고유한 기능 사용이 다소 제한적이라 사용에 불편함이 많았지만 현재는 많이 개선되어, react native 측에서도 cli 방식보다 현재는 expo를 이용한 프로젝트 생성 방식을 권장하고 있다.

이렇게 react native 프로젝트를 생성하는 방법에 대해 이야기를 한 이유는 expo 프로젝트와 cli 프로젝트에서 사용하는 문법이 조금씩 다르기 때문이다.
이번 ppt에서는 route(페이지 이동) 기능에 대해 알아볼 것이다. 페이지 이동 방식은 순수한 react Native 방식의 문법을 활용하여 구현할 수도 있으며, expo에서 권장하는 문법을 사용하여 구현할 수도 있다. 우리는 expo 방식으로 프로젝트를 생성하기에 expo-router를 이용한 페이지 이동 방식을 알아보겠다.

expo 프로젝트의 파일 구조

expo 프로젝트는 파일기반 routing 구조를 갖는다. 파일 기반이란 파일이 만들어진 폴더, 만들어진 폴더명을 기준으로 페이지 이동 경로가 결정된다는 것을 말한다. 아래는 stack navigation을 연습할 기본적인 파일 구조이다.



- _layout : 같은 폴더 내의 레이아웃을 구성하는 파일. 레이아웃을 결정하는 파일명은 '_layout'으로 정해져 있음
- index.js : 같은 폴더 내에서 최초 실행되는 파일. 최초 실행되는 파일의 이름은 'index.js'로 정해져 있음.(url : '/')
- detail.js : 상세화면 (url : '/detail')
- myPage.js : 내정보 화면 (url : '/myPage')

위 구조에서 보듯 파일명이 곧 url 경로가 된다. 이러한 구조를 파일기반 routing구조라 한다.

Stack Navigation 사용하기

먼저 레이아웃을 결정하는 '_layout.js' 파일에 Stack Navigation을 사용을 위해 다음과 같이 작성한다.

```
//_layout.js
const HomeLayout = () => {
  return (
    <Stack />
  )
}

export default HomeLayout

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

'_layout.js'에 위와 같이 작성하면 자동으로 app 폴더 안의 모든 파일(index, detail, myPage)들이 stack navigation 구조를 갖는다.

Stack Navigation 사용하기

앱 실행 시 최초 실행되는 파일인 'index.js'에는 다음과 같이 작성하였다.

```
//index.js
const Home = () => {
  const router = useRouter();

  return (
    <View style={styles.homeContainer}>
      <Text>홈 화면</Text>

      <Pressable
        onPress={(e) => router.push('/detail')}
      >
        <Text>상세페이지로 이동</Text>
      </Pressable>

      <Pressable
        onPress={(e) => router.push('/myPage')}
      >
        <Text>마이페이지 이동</Text>
      </Pressable>
    </View>
  )
}
```

- useRouter()는 react에서 useNavigate()와 같은 역할은 하는 hook이다.
- useRouter()로 생성된 객체는 push(), replace(), navigate() 세 가지 함수를 통해 화면 전환을 할 수 있다.
- push() : 새로운 화면을 스택에 추가하여 이동
- replace() : 현재 화면을 새 화면으로 교체
- navigate() : 해당 경로가 있으면 이동만 하고 스택에 추가하지 않음. 해당 경로가 없다면 스택에 추가하여 이동.
- push('/detail')은 app폴더 내의 detail.js를 실행한다는 의미이다.

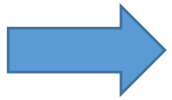
Stack Navigation 사용하기

화면 전환 시 데이터를 가져갈 때는 아래와 같이 작성한다.

pathname에는 이동 경로를 작성하고, params에는 전달

할 데이터를 객체 형식으로 작성한다.

```
<Pressable
  onPress={(e) => router.push({
    pathname : '/myPage',
    params : {
      id : 'abc',
      age : 20
    }
  })}
>
  <Text>마이페이지 이동</Text>
</Pressable>
```



```
//myPage.js
const MyPage = () => {
  const params = useLocalSearchParams();

  return (
    <View>
      <Text>myPage</Text>
      <Text>id : {params.id} / age : {params.age}</Text>
    </View>
  )
}

export default MyPage
```

route를 통해 전달된 데이터는 useLocalSearchParams()를

사용하면 객체 형식으로 받을 수 있다.

Stack Navigation 사용하기

stack 네비게이션을 사용하면 화면 상단에 현재 이동하는 페이지 정보가 뜬다. 만약 페이지 정보를 없애고 싶다면 index.js 파일을 다음과 같이 수정한다.

```
//_layout.js
const HomeLayout = () => {
  return (
    <Stack
      screenOptions={{headerShown:false}}
    />
  )
}

export default HomeLayout
```

Stack 컴포넌트의 screenOptions를 이용하면 헤더를 숨기거나, 색상 및 폰트 변경 등 헤더에 대한 내용을 변경할 수 있다.

Stack Navigation 사용하기

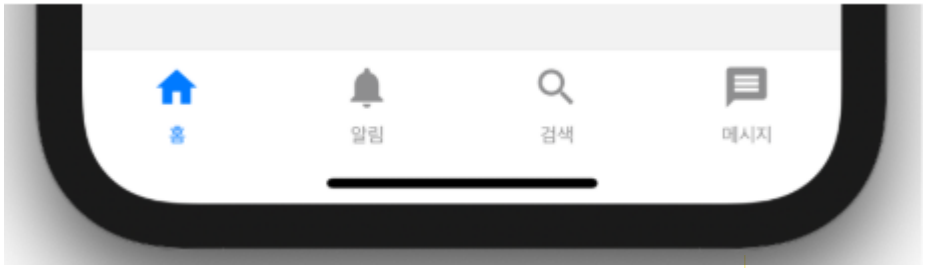
다음은 <Stack> 컴포넌트의 screenOptions에 사용할 수 있는 내용과 예시이다.

옵션	설명
`headerShown`	헤더 표시 여부 (`true` / `false`)
`title`	헤더 타이틀 텍스트
`headerTitleAlign`	타이틀 정렬 방식 (`center`, `left`)
`headerStyle`	헤더 배경 스타일 (배경색 등)
`headerTintColor`	헤더의 텍스트, 버튼 색상
`headerTitleStyle`	타이틀 텍스트 스타일 (폰트 크기 등)
`headerBackTitleVisible`	iOS에서 뒤로가기 버튼의 텍스트 숨김 여부
`headerRight`	헤더 오른쪽에 들어갈 컴포넌트
`headerLeft`	헤더 왼쪽에 들어갈 컴포넌트
`gestureEnabled`	제스처(스와이프 등)로 화면 이동 가능 여부
`gestureDirection`	제스처 방향 (`horizontal`, `vertical`)
`animation`	화면 전환 애니메이션 종류 (`fade`, `slide_from_right`, `none` 등)
`presentation`	모달 방식 (`card`, `modal`, `transparentModal`, `fullScreenModal`)
`contentStyle`	화면 전체의 스타일
`animationDuration`	화면 전환 애니메이션 시간 (ms)
`statusBarStyle`	상태바 스타일 (`light`, `dark`, `auto`)
`statusBarColor`	안드로이드에서 상태바 배경색
`orientation`	화면 방향 설정 (`portrait`, `landscape`)

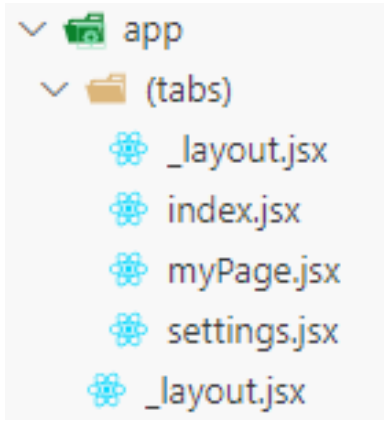
```jsx
<Stack screenOptions={{
  headerShown: true,
  title: '기본 제목',
  headerStyle: {
    backgroundColor: '#f4511e',
  },
  headerTintColor: '#fff',
  headerTitleStyle: {
    fontWeight: 'bold',
  },
  gestureEnabled: true,
  animation: 'slide_from_right'
}}/>
```

tab Navigation 사용하기

tab 네비게이션을 사용하면 앱 화면 상단 혹은 하단에 메뉴를 만들수 있다.



다음은 탭 네비게이션을 구성할 파일 구조이다. expo 프로젝트는 파일기반 라우팅을 지원하기 때문에 생성한 폴더명과 파일명이 화면 전환 경로가 됨을 잊지말자.



- app/_layout : 같은 폴더 내의 레이아웃을 구성하는 파일. 레이아웃을 결정하는 파일명은 '_layout'으로 정해져 있음
- app/(tabs) : 폴더를 만들 때 ()를 붙이면 파일 경로에서 제외된다.
  특수한 문법 : 폴더를 소괄호 () 안에 넣으면 경로를 생략한다는 문법임
- app/(tabs)/_layout : 레이아웃을 구성하는 파일로, 해당 파일에 탭 네비게이션을 구성한다.
- app/(tabs)/index : 탭 레이아웃이 실행되면 기본적으로 화면에 보여지는 파일
- app/(tabs)/myPage : 마이 페이지 (url : '/detail')
- app/(tabs)/setting : 세팅 페이지 (url : '/settings')

tab Navigation 사용하기

먼저 , app 폴더 안의 _layout.jsx 파일을 다음과 같이 작성한다.

```jsx
//_layout.jsx
const HomeLayout = () => {
  return (
    <Stack screenOptions={{headerShown:false}}/>
  )
}

export default HomeLayout

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

- 전체적으로는 스택 구조를 이루며, 스택은 (tabs)폴더 하나로만 이루어진 구조이다.
- 스택으로 (tabs) 폴더 하나만 사용하기 때문에, 스택으로 구성한 것은 큰 의미를 부여하지 않아도 된다.
- screenOptions을 통해 스택 구조의 헤더 정보를 보이지 않게 구성하였다.

tab Navigation 사용하기

(tabs) 폴더 안의 _layout.jsx 파일에는 탭 네비게시션을 다음과 같이 만든다.

```jsx
// /(tabs)/_layout.jsx
const TabLayout = () => {
  return (
    <Tabs screenOptions={{tabBarInactiveTintColor: 'blue'}}
    >
      <Tabs.Screen
        name='index'
        options={{
          title:'Home',
          tabBarIcon : () => <MaterialIcons name="home" size={24} color="black" />
        }}
      />
      <Tabs.Screen
        name='myPage'
        options={{
          title:'내 정보',
          tabBarIcon : () => <MaterialIcons name="account-circle" size={24} color="black" />
        }}
      />
      <Tabs.Screen name='settings' options={{title:'Settings'}}/>
    </Tabs>
  )
}
```

- <Tabs.Screen> 컴포넌트 하나하나가 각각 탭 메뉴가 된다.
- <Tabs.Screen> 안의 name 속성은 탭과 연결될 jsx 파일명이다.
- <Tabs.Screen> 안의 options에는 탭과 관련한 여러 속성값을 넣을 수 있다. 현재는 탭의 제목과 탭에 표시할 아이콘을 작성한 것이다.
- npm install @expo/vector-icons 명령어로 라이브러리를 실치하면 간단하게 아이콘을 사용할 수 있다.

  (https://icons.expo.fyi/Index)

tab Navigation 사용하기

다음은 <Tabs> 컴포넌트의 screenOptions에 적용될 수 있는 옵션과 예시이다. 이 곳에 작성하는 옵션은 모든 탭에 적용된다.

옵션 이름	설명
`tabBarLabel`	탭에 표시될 텍스트 (string 또는 함수 가능)
`tabBarIcon`	탭에 표시될 아이콘 컴포넌트 (React component 리턴하는 함수)
`tabBarBadge`	탭 아이콘 옆에 표시되는 뱃지 (숫자 또는 문자열)
`tabBarActiveTintColor`	활성화된 탭의 아이콘 및 텍스트 색상
`tabBarInactiveTintColor`	비활성 탭의 아이콘 및 텍스트 색상
`tabBarActiveBackgroundColor`	활성 탭의 배경색
`tabBarInactiveBackgroundColor`	비활성 탭의 배경색
`tabBarHideOnKeyboard`	키보드가 열릴 때 탭바 숨김 여부 (Android에 유용)
`headerShown`	해당 스크린의 헤더를 표시할지 여부
`tabBarStyle`	탭 바의 전체 스타일 객체
`tabBarLabelStyle`	탭 라벨 텍스트의 스타일
`tabBarIconStyle`	탭 아이콘의 스타일
`tabBarItemStyle`	각 탭 항목(버튼)의 스타일
`unmountOnBlur`	탭이 포커스를 잃을 때 컴포넌트를 언마운트할지 여부

```
<Tabs
  screenOptions={{
    tabBarActiveTintColor: 'tomato',
    tabBarInactiveTintColor: 'gray',
    tabBarStyle: { backgroundColor: '#fff', height: 60 },
    headerShown: false,
    tabBarIcon: ({ color, size }) => (
      <Ionicons name="home" color={color} size={size} />
    ),
  }}
>
  <Tabs.Screen name="home" component={HomeScreen} />
  <Tabs.Screen name="settings" component={SettingsScreen} />
</Tabs>
```

tab Navigation 사용하기

다음은 <Tabs.Screen> 컴포넌트의 options에 적용될 수 있는 옵션과 예시이다. 이 곳에 작성하는 옵션은 해당 탭에만 적용된다.

옵션	설명
`title`	헤더 및 탭 라벨 텍스트 설정 (`tabBarLabel` 없을 때 사용됨)
`tabBarLabel`	탭 바에 보여질 라벨 텍스트 (string 또는 function)
`tabBarIcon`	탭 아이콘 설정 (React 컴포넌트를 리턴하는 함수)
`tabBarBadge`	탭에 표시할 배지 (숫자 또는 문자열)
`tabBarVisible`	탭 바 숨기기 (현재는 `tabBarStyle: { display: 'none' }` 사용 권장)
`headerShown`	헤더 보여줄지 여부 (기본은 true)
`headerTitle`	헤더에 표시될 제목
`headerRight`	헤더 오른쪽에 들어갈 컴포넌트
`headerLeft`	헤더 왼쪽에 들어갈 컴포넌트
`tabBarActiveTintColor`	이 탭이 활성화되었을 때 아이콘/텍스트 색
`tabBarInactiveTintColor`	비활성화 시 색
`tabBarButton`	탭 자체를 커스텀 버튼으로 구성할 수 있음
`tabBarStyle`	특정 스크린에서만 탭 바 스타일 변경
`unmountOnBlur`	이 스크린에서 다른 탭으로 이동 시 컴포넌트 언마운트 여부

```
<Tabs.Screen
  name="home"
  component={HomeScreen}
  options={{
    title: '홈',
    tabBarLabel: '홈화면',
    tabBarIcon: ({ color, size }) => (
      <Ionicons name="home" size={size} color={color} />
    ),
    tabBarBadge: 3,
    headerShown: false,
    tabBarActiveTintColor: 'tomato',
    tabBarInactiveTintColor: 'gray',
  }}
/>
```