apps in toss

Menu On this page

화면 이동하기

Bedrock 애플리케이션에서 새로운 화면으로 이동하거나, 화면 히스토리를 제어하는 등의 라우팅 작업을 쉽게 처리할 수 있어요.

아래 예제 코드를 통해 다양한 라우팅 기능을 설명할게요.

> INFO
>
> Bedrock은 React Navigation을 기반으로 동작해요

예제 코드

예제 코드는 총 3개의 페이지로 구성되어 있고, 구조는 아래와 같아요.

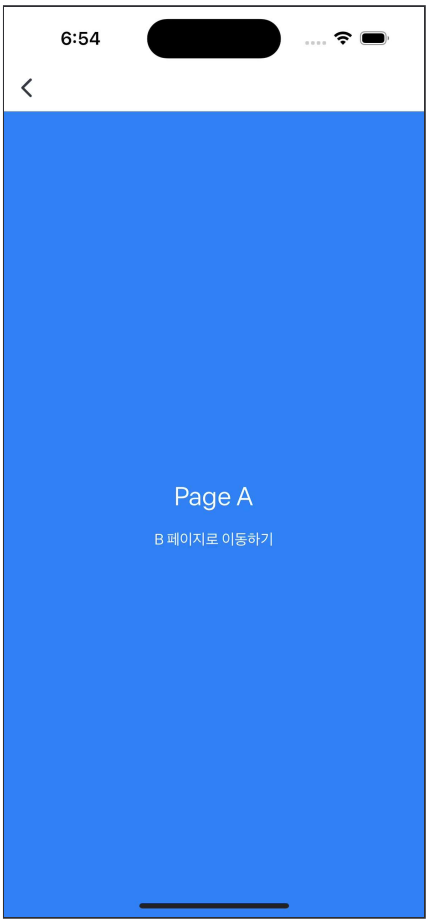
```
root
├── pages
│   ├── page-a.tsx
│   ├── page-b.tsx
│   └── page-c.tsx
└── src
    └── ...
```

▶ page-a.tsx 소스코드

▶ page-b.tsx 소스코드

▶ page-c.tsx 소스코드

페이지 A: 화면 이동하기



useNavigation 은 화면 간 이동을 처리할 때 사용해요.

navigate 메서드로 이동할 화면의 경로와 필요한 데이터를 함께 전달할 수 있어요.

tsx

```tsx
// page-a.tsx
import { BedrockRoute, useNavigation } from "react
import { StyleSheet, View, Text, Pressable } from

export const Route = BedrockRoute("/page-a", {
  validateParams: (params) => params,
  component: PageA,
});

function PageA() {
  const navigation = useNavigation();
  const handlePress = () => {
    navigation.navigate("/page-b");
  };

  return (
    <View style={[styles.container, { backgroundCo
      <Text style={styles.text}>Page A</Text>
      <Pressable onPress={handlePress}>
        <Text style={styles.buttonLabel}>B 페이지로
      </Pressable>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    gap: 16,
    padding: 16,
  },
  text: {
    color: "white",
```

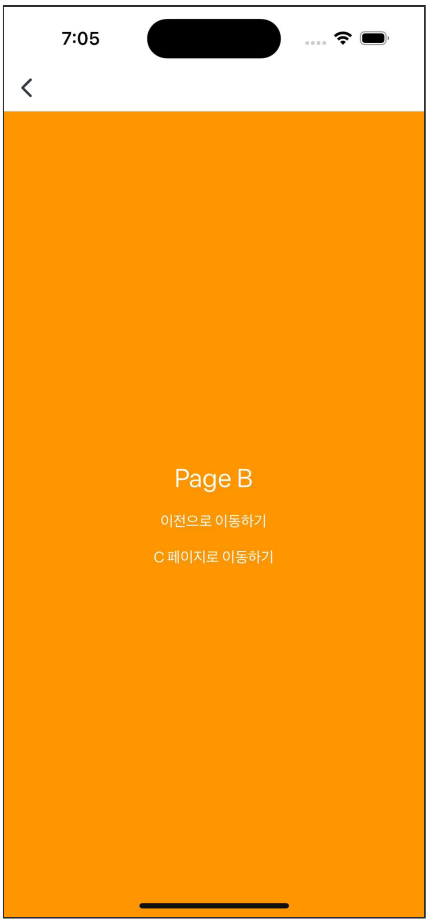
```
    fontSize: 24,
  },
  buttonLabel: {
    color: "white",
  },
});
```

주요 포인트

- `useNavigation` 훅을 사용해 `navigation` 객체를 가져와요.
- `navigation.navigate('/page-b')` 를 호출하면 'B' 페이지로 이동해요.

페이지 B: 이전 화면으로 돌아가기

`goBack` 메서드를 사용하면 이전 화면으로 돌아갈 수 있어요. 하지만 이전 화면 기록이 없는 경우에는 에러가 발생할 수 있으니, `canGoBack` 으로 먼저 확인해야 해요.



tsx

```
// page-b.tsx
import { BedrockRoute, useNavigation } from "react
import { StyleSheet, View, Text, Pressable } from

export const Route = BedrockRoute("/page-b", {
  validateParams: (params) => params,
  component: PageB,
});

function PageB() {
  const navigation = useNavigation();

  // 이전 화면으로 돌아가는 함수예요.
  const handlePressBackButton = () => {
    if (navigation.canGoBack()) {
      navigation.goBack();
    } else {
      console.warn("이전 화면으로 이동할 수 없습니다
    }
  };
```

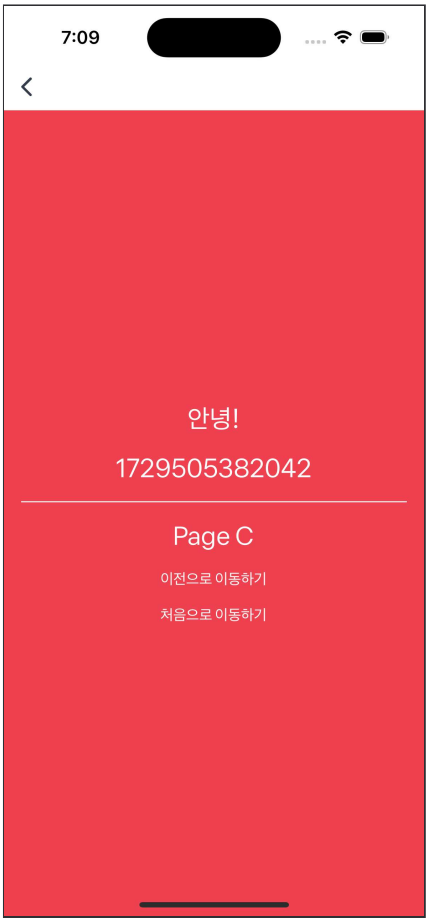
```
  const handlePressNextButton = () => {
    navigation.navigate("/page-c", {
      message: "안녕!",
      date: new Date().getTime(),
    });
  };

  return (
    <View style={[styles.container, { backgroundCo
      <Text style={styles.text}>Page B</Text>
      <Pressable onPress={handlePressBackButton}>
        <Text style={styles.buttonLabel}>이전으로
      </Pressable>
      <Pressable onPress={handlePressNextButton}>
        <Text style={styles.buttonLabel}>C 페이지로
      </Pressable>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    gap: 16,
    padding: 16,
  },
  text: {
    color: "white",
    fontSize: 24,
  },
  buttonLabel: {
    color: "white",
  },
});
```

주요 포인트

- `canGoBack()` 으로 이전 화면이 있는지 확인하고, 있으면 `goBack()` 을 호출해요.

- `navigate('/page-c', { message: '안녕!', date: new Date().getTime() })` 로 데이터를 전달하면서 'C' 페이지로 이동해요.

페이지 C: 전달받은 데이터 사용하기



`Route.useParams` 훅은 다른 화면에서 전달된 데이터를 가져올 때 사용해요.

이때, `BedrockRoute.validateParams` 옵션을 설정하면 전달된 데이터를 타입 검증(Type-Safe)하면서 접근할 수 있어요. 이를 통해 잘못된 데이터 형식으로 인한 에러를 방지할 수 있어요.

tsx

```tsx
// page-c.tsx
import { BedrockRoute, useNavigation } from "react
import { CommonActions } from "@react-native-bedrc
import { StyleSheet, View, Text, Pressable } from

export const Route = BedrockRoute("/page-c", {
  validateParams: (params) => params as { message:
  component: PageC,
});

function PageC() {
  const navigation = useNavigation();
  const params = Route.useParams();
  // 또는 아래와 같이 사용할 수 있어요.
  // import { useParams } from 'react-native-bedrc
  //
  // const params = useParams({
  //   from: '/page-b',
  // });

  const handlePressHomeButton = () => {
    navigation.dispatch((state) => {
      return CommonActions.reset({
        ...state,
        index: 0,
        routes: state.routes.filter((route) => rou
      });
    });
```

```
  };

  return (
    <View style={[styles.container, { backgroundCo
      <Text style={styles.text}>{params.message}</
      <Text style={styles.text}>{params.date}</Tex
      <View style={styles.line} />
      <Text style={styles.text}>Page C</Text>
      <Pressable onPress={handlePressHomeButton}>
        <Text style={styles.buttonLabel}>처음으로
      </Pressable>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    gap: 16,
    padding: 16,
  },
  text: {
    color: "white",
    fontSize: 24,
  },
  buttonLabel: {
    color: "white",
  },
});
```

주요 포인트

- `Route.useParams` 훅을 사용하면 URL에서 전달된 데이터(매개변수)에 접근할 수 있어요.
- `BedrockRoute.validateParams` 옵션을 설정하면 데이터 타입을 검증하면서(Type-Safe) 안전하게 사용할 수 있어요.

화면 파라미터 타입 정의하기

페이지마다 아래와 같은 Route 컴포넌트를 정의해요. 여기서 `validateParams` 옵션은 해당 화면에서 받을 파라미터의 타입을 정의해요.

tsx

```tsx
export const Route = BedrockRoute("/page-c", {
  validateParams: (params) => params as { message: string; date: number },
  component: PageC,
});
```

위 코드에서 `validateParams` 는 `message` 와 `date` 라는 두 필드를 포함한 매개변수를 타입으로 정의해요.

이를 통해 다른 코드에서 `useNavigate` 나 `useParams` 를 사용할 때, 타입 검사를 통해 필요한 경로와 전달해야 할 데이터를 명확히 알 수 있어요. 이렇게 하면 코드의 안전성과 가독성이 높아져요.

자동 타입 정의 생성

아래 명령어를 실행하면 정의된 `Route` 를 기반으로 타입 정의가 자동으로 생성돼요.

개발 모드에서는 `pages/` 디렉토리에 파일이 추가되면 자동으로 타입 정의가 생성되므로 별도의 명령어를 실행하지 않아도 돼요.

npm pnpm yarn

sh

```sh
$ npm run bedrock routegen
```

생성된 파일 예시

자동으로 생성된 파일은 다음과 같아요. 이 파일은 자동 생성되므로 수동으로 수정할 필요가 없어요.

tsx

```tsx
// src/router.gen.ts

/* eslint-disable */
// This file is auto-generated by react-native-bedrock. DO NOT EDIT.
import { Route as _PageARoute } from "../pages/page-a";
import { Route as _PageBRoute } from "../pages/page-b";
import { Route as _PageCRoute } from "../pages/page-c";

declare module "react-native-bedrock" {
```

```
    interface RegisterScreen {
      "/page-a": ReturnType<typeof _PageARoute.useParams>;
      "/page-b": ReturnType<typeof _PageBRoute.useParams>;
      "/page-c": ReturnType<typeof _PageCRoute.useParams>;
    }
  }
```

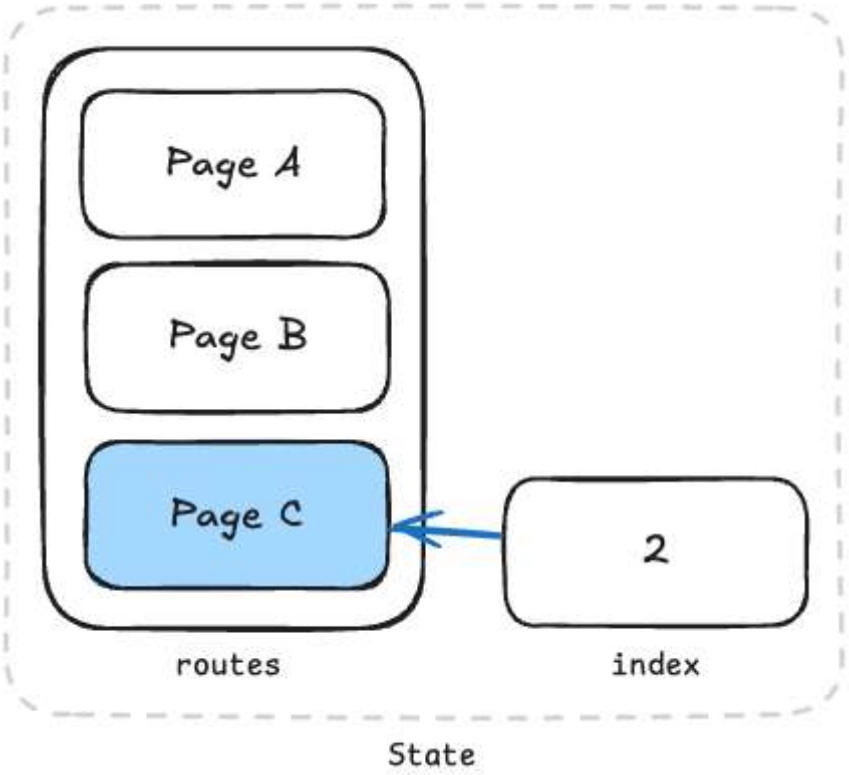
주요 포인트

- 각 화면에서 받을 파라미터의 타입을 `BedrockRoute.validateParams` 옵션으로 정의해두면, `navigate` 와 `params` 사용 시 타입 검사를 받을 수 있어 더 안전하게 코드를 작성할 수 있어요.
- 개발 모드에서는 `pages/` 디렉토리에 파일이 추가되면 타입 정의가 자동으로 생성되기 때문에 별도의 명령어 실행이 필요하지 않아요.

이렇게 React Navigation을 사용하면 화면 간 이동을 쉽게 처리할 수 있고, 데이터를 전달하거나 기록을 조작하는 기능을 통해 다양한 UX를 구현할 수 있어요. 또한 타입스크립트와 함께 사용하면 안전하고 견고한 코드를 작성할 수 있답니다.

라우팅 상태 초기화하기

페이지 A → 페이지 B → 페이지 C 순서로 이동한 직후의 상태는 아래와 그림과 같이 같이 나타낼 수 있어요.

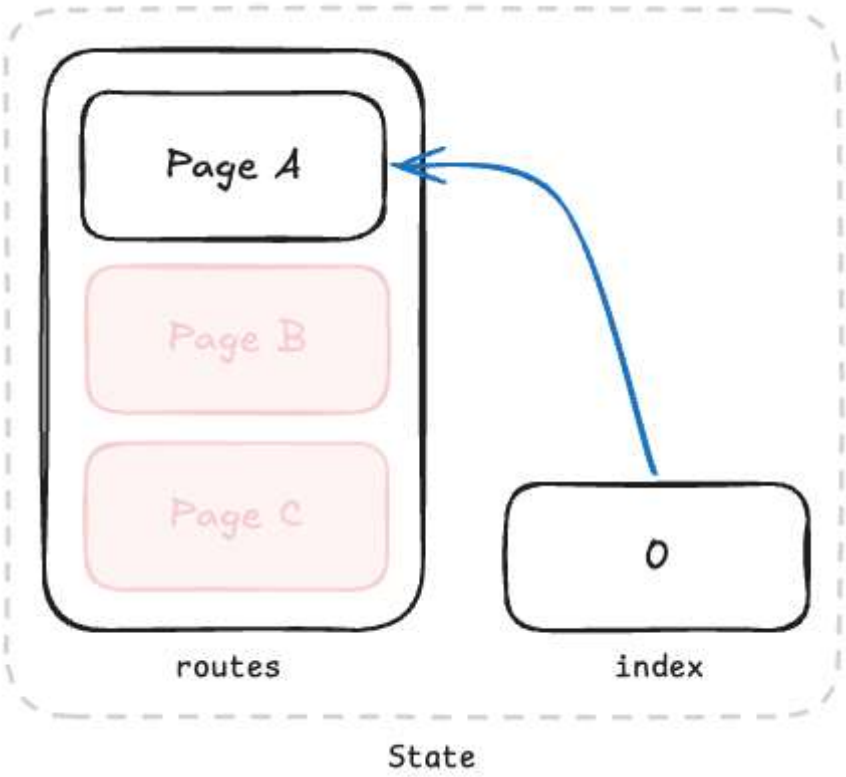


페이지 A, 페이지 B, 페이지 C가 순서대로 `routes` 기록에 남아 있고, `index` 값은 마지막으로 이동한 페이지 C의 위치인 2를 가리켜요.

`reset` 을 사용하면 화면 이동 기록을 초기화할 수 있어요. 예를 들어, '페이지 A → B → C'로 이동한 후에 '페이지 A'로 돌아가면서 B와 C 기록을 삭제하고 싶다면, `CommonActions.reset` 을 사용해요.

tsx

```
navigation.dispatch(
  CommonActions.reset({
    index: 0,
    routes: [{ name: "/page-a" }],
  })
);
```



주요 포인트

- `CommonActions.reset` 으로 특정 화면만 기록에 남기고 나머지 화면 기록을 삭제할 수 있어요.

레퍼런스

- React Navigation 공식 문서

이전 버전 문서가 필요할 때

이전 버전의 문서는 화면 이동하기에서 확인할 수 있어요.

Previous page
UI 표현하기

Next page
HTTP 통신하기