apps in toss

Menu On this page

React Native로 개발하기

준비가 필요해요

프로젝트를 스캐폴딩하고 서비스를 실행하려면 환경설정이 필요해요. 아래 가이드를 먼저 확인해 주세요.

- iOS 환경설정 문서 바로가기
- Android 환경설정 문서 바로가기

Bedrock을 사용해 "Hello Bedrock!"페이지가 표시되는 서비스를 만들어볼게요. 이를 통해 로컬 서버를 연결하는 방법과 파일 기반 라우팅을 배울 수 있어요.

스캐폴딩

앱을 만들 위치에서 다음 명령어를 실행하세요.

이 명령어는 프로젝트를 초기화하고 필요한 파일과 디렉토리를 자동으로 생성해요.

npm pnpm yarn

sh

```
$ npm create bedrock-app@latest
```

1. 앱 이름 지정하기

앱 이름은 kebab-case 형식으로 만들어 주세요. 예를 들어, 아래와 같이 입력해요.

shell

```
# 예시
my-bedrock-app
```

2. 도구 선택하기

`bedrock` 에서는 프로젝트를 생성할 때 필요한 도구를 선택할 수 있어요. 현재 제공되는 선택지는 다음 두 가지예요. 둘 중 한 가지 방법을 선택해서 개발 환경을 세팅하세요.

- `prettier + eslint` : 코드 포맷팅과 린팅을 각각 담당하며, 세밀한 설정과 다양한 플러그인으로 유연한 코드 품질 관리를 지원해요.
- `biome` : Rust 기반의 빠르고 통합적인 코드 포맷팅과 린팅 도구로, 간단한 설정으로 효율적인 작업이 가능해요.

3. 의존성 설치하기

프로젝트 디렉터리로 이동한 뒤, 사용 중인 패키지 관리자에 따라 의존성을 설치하세요.

npm pnpm yarn

sh

```
$ cd my-bedrock-app
$ npm install
```

스캐폴딩 전체 예시

아래는 `my-bedrock-app` 이라는 이름으로 새로운 앱을 스캐폴딩한 결과예요.

스캐폴딩을 마쳤다면 프로젝트 구조가 생성돼요.

0:01 / 0:12

로컬에서 프로젝트 실행하기

이제 여러분만의 Hello Bedrock 페이지를 만들 준비가 끝났어요. 🎉 다음으로 로컬에서 `my-bedrock-app` 서비스를 실행해 볼게요.

시뮬레이터 및 기기에서 실행하기

앱 실행 환경을 먼저 설정하세요.

- Android 환경설정
- iOS 환경설정

1. 개발 서버 실행하기

스캐폴딩된 프로젝트 디렉터리로 이동한 뒤, 선택한 패키지 매니저를 사용해 `dev` 스크립트를 실행하세요. 이렇게 하면 개발 서버가 시작돼요.

npm pnpm yarn

sh

```
$ cd my-bedrock-app
$ npm run dev
```

참고하세요

개발 서버 실행 중 too many open files 에러가 발생한다면, node_modules 디렉터리를 삭제한 뒤 다시 의존성을 설치해 보세요.

sh

```
rm -rf node_modules
npm install  # 또는 yarn, pnpm에 맞게
```

2. 개발 서버 연결하기

개발 서버에 연결해서 로컬 환경에서 애플리케이션을 실행할 수 있어요.
연결 방법은 개발 서버 연결 가이드를 참고하세요.

3. 앱 스킴(URL) 입력하기

접속할 스킴(URL)을 입력해 디바이스에서 앱을 실행하세요.
서비스 이름에 해당하는 스킴을 입력하면 연결이 완료돼요.

```
intoss://my-bedrock-app
```

4. Metro 서버 연결 확인하기

Metro 서버는 React Native에서 번들링 작업을 수행하는 도구예요. 아래 명령어를 실행하면 Metro 서버가 연결되고, 디바이스에서 번들링된 화면을 확인할 수 있어요.

개발 환경이 올바르게 설정됐다면, Metro 서버가 자동으로 실행되고 화면이 나타날 거예요.

코드 확인해보기

프로젝트의 `_app.tsx` 파일에 다음과 같은 코드가 들어있을 거예요.

_app.tsx

```tsx
import { AppsInToss } from '@apps-in-toss/framework';
import { Bedrock } from "react-native-bedrock";
import { PropsWithChildren } from "react";
import { context } from "../require.context";

function App({ children }: PropsWithChildren) {
  return <>{children}</>;
}

export default AppsInToss.registerApp(App, {
  appName: "my-bedrock-app",
  context,
});
```

스캐폴딩 된 코드 알아보기

스캐폴딩 명령어를 실행하면 다음과 같은 파일이 생성돼요.

/pages/index.tsx

tsx

```tsx
import React from "react";
import { Text, View } from "react-native";
import { BedrockRoute } from "react-native-bedrock";

export const Route = BedrockRoute("/", {
  validateParams: (params) => params,
  component: Index,
});

function Index() {
  return (
    <View>
      <Text>Hello Index</Text>
    </View>
  );
}
```

`intoss://my-bedrock-app` 스킴으로 라우팅하면 앱에서 `Index` 컴포넌트가 표시돼요.

만약 해당 화면에 필요한 파라미터가 있다면, `BedrockRoute.validateParams` 옵션을 사용해 파라미터를 정의할 수 있어요.

/pages/index.tsx

tsx

```tsx
import { Text, StyleSheet } from "react-native";

export const Route = BedrockRoute("/", {
  validateParams: (params) =>
    params as {
      age: number;
      name: string;
    },
  component: Index,
});
```

```
function Index() {
  const { age, name } = Route.useParams();
  // 또는 아래와 같이 사용할 수 있어요.
  // import { useParams } from 'react-native-bedrock';
  //
  // const params = useParams({
  //   from: '/',
  // });

  return (
    <Text>
      {name} is {age} years old
    </Text>
  );
}
```

파일 기반 라우팅 이해하기

Bedrock 개발 환경은 Next.js와 비슷한 파일 시스템 기반의 라우팅을 사용해요.

파일 기반 라우팅은 파일 구조에 따라 자동으로 경로(URL 또는 스킴)가 결정되는 시스템이에요. 예를 들어, pages라는 디렉토리에 `detail.ts` 파일이 있다면, 이 파일은 자동으로 `/detail` 경로로 연결돼요.

Bedrock 애플리케이션에서는 이 개념이 스킴과 연결돼요. 스킴은 특정 화면으로 연결되는 주소인데요. 예를 들어, `pages/detail.ts` 라는 파일은 자동으로 `intoss://my-bedrock-app/detail` 이라는 스킴으로 접근할 수 있는 화면이에요. 모든 Bedrock 화면은 `intoss://` 스킴으로 시작해요.

```
my-bedrock-app
└─ pages
    ├─ index.tsx       // intoss://my-bedrock-app
    ├─ detail.tsx      // intoss://my-bedrock-app/detail
    └─ item
        ├─ index.tsx    // intoss://my-bedrock-app/item
        └─ detail.tsx    // intoss://my-bedrock-app/item/detail
```

- `index.tsx` 파일: `intoss://my-bedrock-app`
- `detail.tsx` 파일: `intoss://my-bedrock-app/detail`
- `item/index.tsx` 파일: `intoss://my-bedrock-app/item`

- `item/detail.tsx` 파일: `intoss://my-bedrock-app/item/detail`

jsx

```
┌─ 모든 Bedrock 화면을 가리키는 스킴은
│  intoss:// 으로 시작해요
│
-------------
intoss://my-bedrock-app/detail
         ==============~~~~~~~
             │            └─ pages 하위에 있는 경로를 나타내요
             │
             └─ 서비스 이름을 나타내요
```

이렇게 개발자는 별도로 라우팅 설정을 하지 않아도, 파일을 추가하기만 하면 새로운 화면이 자동으로 설정돼요.

앱인토스

앱인토스를 사용해 번들 파일을 생성하고 출시하는 방법을 소개해요.

설치하기

앱인토스를 사용하려면 `@apps-in-toss/framework` 패키지를 설치해야 해요. 사용하는 패키지 매니저에 따라 아래 명령어를 실행하세요.

npm pnpm yarn

sh

```
$ npm install @apps-in-toss/framework
```

설정파일 구성하기

`ait init` 명령어로 앱 개발에 필요한 기본 환경을 구성할 수 있어요.

1. 아래 명령어 중 사용하는 패키지 관리자에 맞는 명령어를 실행하세요.

npm pnpm yarn

sh

```
npx ait init
```

2. 프레임워크를 선택하세요.

3. 앱 이름(`appName`)을 입력하세요.

   이 이름은 앱인토스 콘솔에서 앱을 만들 때 사용한 이름과 같아야 해요. 앱인토스 콘솔에서 앱 이름을 확인할 수 있어요.

모든 과정을 완료하면 프로젝트 루트에 `bedrock.config.ts` 파일이 생성돼요. 이 파일은 앱 설정을 관리하는 데 사용돼요.

bedrock.config.ts

ts

```ts
import { defineConfig } from "react-native-bedrock/config";
import { appsInToss } from "@apps-in-toss/framework/plugins";

export default defineConfig({
  appName: "<app-name>",
  plugins: [
    appsInToss({
      brand: {
        displayName: "%%appName%%", // 화면에 노출될 앱의 한글 이름으로 바꿔주세요.
        primaryColor: "#3182F6", // 화면에 노출될 앱의 기본 색상으로 바꿔주세요.
        icon: null, // 화면에 노출될 앱의 아이콘 이미지 주소로 바꿔주세요.
        bridgeColorMode: "basic",
      },
      permissions: [],
    }),
  ],
});
```

- `<app-name>` : 앱인토스에서 만든 앱 이름이에요.
- `brand` : 앱 브랜드와 관련된 구성이에요.
  - `displayName` : 브릿지 뷰에 표시할 앱 이름이에요.
  - `icon` : 앱 아이콘 이미지 주소예요. 사용자에게 앱 브랜드를 전달해요.
  - `primaryColor` : Toss 디자인 시스템(TDS) 컴포넌트에서 사용할 대표 색상이에요. RGB HEX 형식(eg. `#3182F6`)으로 지정해요.

  - `bridgeColorMode` : 브릿지 뷰의 배경 색상 유형이에요. 흰 배경인 `basic` 또는 검은 배경인 `inverted` 중 하나를 선택할 수 있어요.
- `permissions` : 권한이 필요한 함수 앱 설정하기 문서를 참고해서 설정하세요.

TDS React Native 패키지 설치하기

`@apps-in-toss/framework` 를 사용하려면 TDS React Native 패키지를 추가로 설치해야 해요. 자세한 내용은 TDS 시작하기를 참고해 주세요.

번들 파일 생성하기

번들 파일은 `.ait` 확장자를 가진 파일로, 빌드된 프로젝트를 패키징한 결과물이에요. 이를 생성하려면 아래 명령어를 실행하세요.

npm pnpm yarn

sh

```
npm run build
```

위 명령어를 실행하면 프로젝트 루트 디렉터리에 `<서비스명>.ait` 파일이 생성돼요. 해당 파일은 앱을 출시할 때 사용해요.

앱 출시하기

앱을 출시하는 방법은 앱 출시하기문서를 참고하세요.

Previous page
Unity 포팅하기

Next page
UI 표현하기