

- Next.js 설치하기

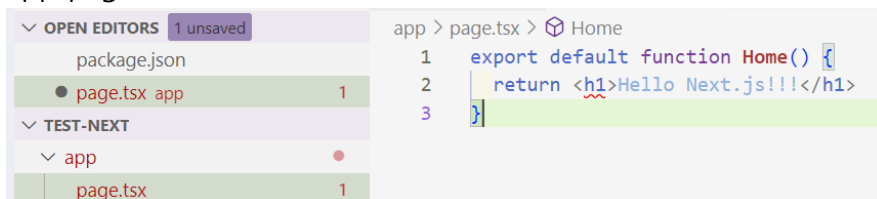
- 수동으로 설치하기

- ◆ 프로젝트 이름은 소문자로만, 두 단어 이상일 때 케밥 케이스로 만든다.

1. test-next 폴더 만들고 해당 경로에서 vscode을 실행하고 터미널 열기
2. npm init -y
3. npm i react [next](#) react-dom
4. package.json을 열어 test 지우고 dev 스크립트 추가하기

```
"scripts": {  
  "dev": "next dev"  
},
```

5. app/page.tsx 만들기



6. 실행하기

npm run dev

```
We detected TypeScr  
✓ Ready in 14.2s
```

실행하면 typescript 에 대한 설명과 함께 typescript 라이브러리 설치된다.

```
It looks like you're trying to use TypeScript but do not have t  
Installing dependencies  
{  
  "devDependencies": {  
    "@types/node": "22.13.14",  
    "@types/react": "19.0.12",  
    "typescript": "5.8.2"  
  }  
}
```

page.tsx 파일을 확인하면 경고가 사라졌다.

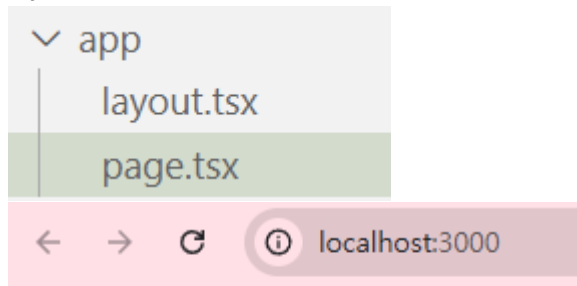
```
export default function Home() {  
  return <h1>Hello Next.js!!!</h1>  
}
```

7. 브라우저에서 확인하기 : Ctrl + 링크 클릭

```
next dev  
  
▲ Next.js 14.1.0  
- Local:      http://localhost:3000
```

```
○ Compiling / ...  
⚠ Your page app/page.tsx did not have a root layout. We created one for you at app/layout.tsx.  
✖ ./  
Module not found: Can't resolve 'D:\UPus\UPlusFrontEnd\lecture\app\layout.tsx'
```

layout.tsx 없어서 새로 생성해 준다. > 삭제 해도 바로 생성해 준다



Hello Next!!

■ 자동으로 설치하기

형식] 가장 최근 버전으로 생성

`npx create-next-app`

`npx create-next-app --use-npm` //npm으로 변경하기

형식] 버전 지정해서 생성하기

`npx create-next-app@버전 --use-npm`

ex) `npx create-next-app@14.1.0`

```
PS C:\kdg\06_react> npx create-next-app --use-npm  
✓ What is your project named? ... my-app  
✓ Would you like to use TypeScript? ... No / Yes  
✓ Would you like to use ESLint? ... No / Yes  
✓ Would you like to use Tailwind CSS? ... No / Yes  
✓ Would you like your code inside a `src/` directory? ... No / Yes  
✓ Would you like to use App Router? (recommended) ... No / Yes  
✓ Would you like to use Turbopack for `next dev`? ... No / Yes  
? Would you like to customize the import alias (`@/*` by default)? » No / Yes
```

■ Would you like to customize the default import alias(@/*) 옵션

Next.js에서는 기본적으로 @를 src/ 폴더에 매핑한다. 기본을 사용하기 위해서는 No
ex)

`src/components/Button.tsx` 컴포넌트를 import하기

```
import Button from '@components/Button'
```

사용자가 원하는 대로 import alias를 설정할 경우 Yes

ex) @custom 에 원하는 경로를 설정 한 경우

```
import Button from '@custom/components/Button'
```

tsconfig.json에서 변경 가능하다.

```
    "paths": {  
      "@/*": [".src/*"]  
    }
```

■ 디렉토리 구조

> .next	Next.js가 빌드한 결과물이 저장되는 디렉토리
> public	정적 파일(이미지, 아이콘, 폰트)을 위한 디렉토리
▼ src	프로젝트의 주요 소스 코드가 포함된 디렉토리
> app	Root segment(Home) 및 app routing
> components	재사용 가능한 UI 컴포넌트들을 위한 디렉토리
> services	Ajax를 위한 API
> store	전역 상태 관리 라이브러리를 위한 디렉토리
> styles	전역 CSS, CSS 모듈 파일 등을 위한 디렉토리
> types	TypeScript 타입 정의 파일을 위한 디렉토리
> utils	자주 사용하는 유틸리티를 위한 디렉토리

■ 실행하기 (package.json 참고)

```
npm run dev
```

■ 브라우저 자동으로 띄워기

pacakage.json 에 설정하기

```
    "scripts": {  
      "dev": "npm run open-browser && next dev",  
      "open-browser": "start http://localhost:3000",  
      "build": "next build",  
      "start": "next start",  
      "lint": "next lint"  
    },
```

```
"dev": "npm run open-browser && next dev",
```

```
"open-browser": "start http://localhost:3000",
```

- Next Router

- 라우팅

12/13 ver

pages/경로/index.tsx or index.js or index.jsx



```
// pages/index.js
export default function Page() {
  return <h1>Hello, Next.js!</h1>;
}
```

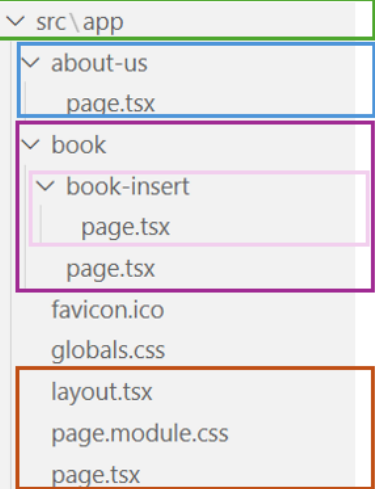
The image shows a file tree for Next.js 12/13 version. The 'pages' directory contains 'index.js' (example URL: http://localhost:3000/), 'about.js' (example URL: http://localhost:3000/about), 'products' directory (containing 'index.js' (http://localhost:3000/products), 'list.js' (http://localhost:3000/products/list), and '[id].js' (http://localhost:3000/products/1, http://localhost:3000/products/abc)), and 'blog' directory (containing '[...slug]' (http://localhost:3000/blog/study, http://localhost:3000/blog/2023/07/10)). Below the tree is a code snippet for 'pages/index.js' showing a default export function 'Page' that returns an HTML element with 'Hello, Next.js!'.

13/14/15 ver

pages/경로/index.tsx or index.js or index.jsx

app/경로/page.tsx or page.js or page.jsx

- ◆ app router



root/about-us 를 위한 page
중첩 라우팅 root/book/book-insert 를 위한 page
root/book 을 위한 page
root (http://localhost:3000/) 를 위한 page

The image shows a file tree for Next.js 13/14/15 version using the app router. The 'src/app' directory contains 'about-us/page.tsx' (for root/about-us), 'book' directory (containing 'book-insert/page.tsx' for nested routing and 'page.tsx' for root/book), 'favicon.ico', 'globals.css', and 'layout.tsx', 'page.module.css', 'page.tsx' (for root). Colored boxes and text labels on the right explain the purpose of each directory and file.

```
// app/layout.js
export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body>{children}</body>
    </html>
  );
}

// app/page.js
export default function Page() {
  return <h1>Hello, Next.js!</h1>;
}
```

■ 페이지 링크

a tag 대신 Link tag 사용

형식 <Link href="url"> </Link>

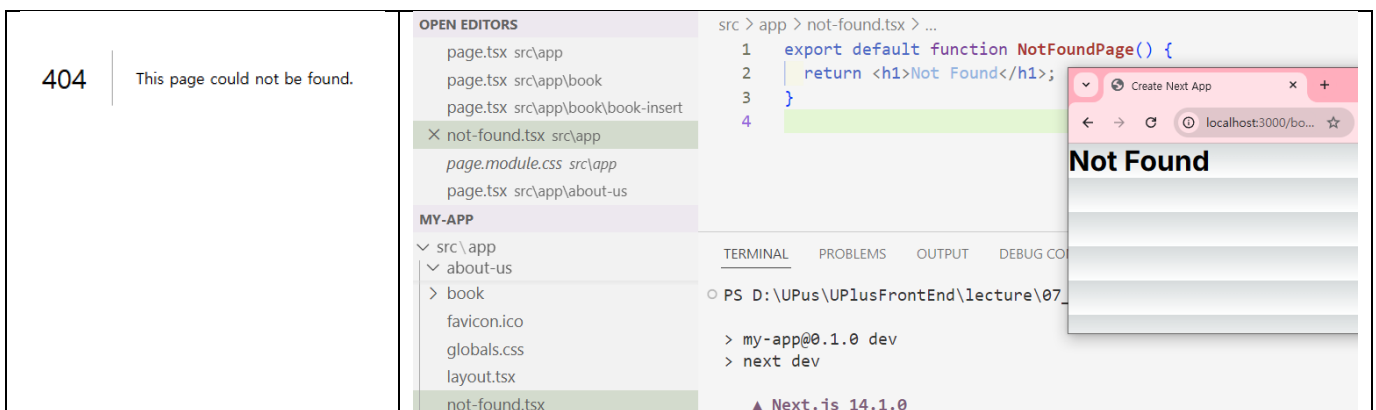
■ App routing Directory에 배치하는 파일

파일명	설명
page.tsx	해당 경로에 대한 페이지 컴포넌트
layout.tsx	해당 경로 이하에 적용할 부분 레이아웃
loading.tsx	로딩 상태 시 보여질 컴포넌트
error.tsx	에러 발생 시 보여질 컴포넌트
not-found.tsx	404 페이지를 커스터마이징할 때 사용

■ Not found 페이지 설정

app>not-found.tsx 배치 : 전역 설정

각 routing 디렉토리에 배치: local 설정



■ Metadata

- html로 rendering될 때 head의 title과 meta를 생성하는 기능
- page or layout에 metadata를 생성할 수 있다
- 전역으로 app에 설정할 수 있고 local로 각 routing 마다 설정할 수 있다.
- Metadata 종류

항목	설명	역할 / 사용 목적	예시 설정
robots	검색 엔진 크롤링 허용 여부 설정	페이지를 검색 엔진에 노출할지 제어함	<code>{ index: true, follow: true }</code>
canonical	중복 콘텐츠의 원본 주소 지정	SEO 중복 페이지 문제 방지	<code>canonical: 'https://example.com/about'</code>
twitter	트위터 카드 메타 정보	트위터에서 링크 공유 시 썸네일, 설명, 제목 등 표시	<code>{ card: 'summary_large_image', title: '제목' }</code>
openGraph	페이스북, 슬랙, 카톡 등에 서의 링크 미리보기 정보	링크 공유 시 썸네일, 제목, 설명 보여주기	<code>{ title: 'OG 제목', images: [url] }</code>
viewport	모바일 반응형 뷰 설정	모바일 기기에서의 화면 크기/스케일 제어	<code>{ width: 'device-width', initialScale: 1 }</code>

■ Private directory

라우팅과 관련이 없다고 명시하기 위해 Directory 명 앞에 `_` prefix 붙이기
로직 분리 및 시각화에 유용하다.

형식] `_directory-name`

■ Route group

URL Path에 포함시키지 않으면서 그룹화할 때 사용하는 기법
형식] `(directory-name)`

■ Dynamic Routes

Path Variable를 통해 해당 데이터를 위한 페이지를 작성할 때 사용
Path Variable의 이름을 `directory`이름으로 설정하고 Data는 `props.params.pathVariable`로 접근해서 사용한다.

형식] `[directory-name]`

Ex) `book>[isbn]>page.tsx`

● SSR vs CSR

■ CSR (Client-Side Rendering)

초기 HTML 파일이 로드될 때 내용이 거의 비어 있고, 이후 **JavaScript가 실행되면서** React가 컴포넌트를 렌더링하여 화면에 표시하는 방식

◆ CSR의 장점

1. 브라우저에서 View 렌더링하기 때문에 서버 트래픽을 감소시키고, 사용자에게 더 빠른 인터랙션을 제공
2. 새로고침이 발생하지 않아 사용자에게 네이티브 앱과 비슷한 경험을 제공

◆ CSR의 문제점

1. 첫 페이지 로딩 속도가 SSR보다 느리다
첫 요청 시 전체 페이지에 대한 모든 파일을 다운 받은 후에 렌더링하기 때문.
2. JS를 무시하면 화면이 안보인다.
3. SEO(Search Engine Optimization)가 웹 사이트에 대한 데이터를 수집하지 못하기 때문에 CSR로 작성된 페이지는 검색결과에서 제외될 수 있으므로 별도의 보완 작업이 필요하다.(ex sitemap 문서)
단, 구글은 검색엔진에 자바스크립트 엔진이 내장되어 있다.

■ SSR(Server Side Renderin)

클라이언트에서 요청이 들어올 때마다 매번 서버에서 새로운 화면(View)을 만들어 제공하는 방식

◆ SSR의 장점

1. 페이지 로딩 속도가 클라이언트 사이드 렌더링에 비해 더 빠르다.
2. 검색엔진최적화(SEO)가 가능

◆ SSR의 단점

초기 로딩 이후 페이지 이동 시 속도가 다소 느리다.

■ Next.js

기본적으로 SSR을 사용한다. 모든 Component들이 Server Side에서 먼저 render된다.

◆ CSR Component

“use client” 지시어를 컴포넌트 최 상단에 선언한 경우 CSR Component가 된다

일반적인 html은 SSR이지만 React 라이브러리를 이용한 hydration은 CSR로 처리된다.

➤ hydration

서버에서 렌더링된 페이지에 스크립트 코드를 통해 웹페이지를 동적으로 처리하는 기능

● Axios

<https://axios-http.com/kr/docs/intro>

Axios는 node.js와 브라우저를 위한 Promise 기반 HTTP 클라이언트

서버 사이드에서는 네이티브 node.js의 http 모듈을 사용하고,

클라이언트(브라우저)에서는 XMLHttpRequest를 사용한다.

설치

npm 사용하기:

```
$ npm install axios
```

yarn 사용하기:

```
$ yarn add axios
```

unpkg CDN 사용하기:

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

Axios API 레퍼런스

`axios` 에 해당 config을 전송하면 요청이 가능합니다.

axios(config)

```
// POST 요청 전송
axios({
  method: 'post',
  url: '/user/12345',
  data: {
    firstName: 'Fred',
    lastName: 'Flintstone'
  }
});
```

```
// node.js에서 GET 요청으로 원격 이미지 가져오기
axios({
  method: 'get',
  url: 'http://bit.ly/2mTM3nY',
  responseType: 'stream'
})
.then(function (response) {
  response.data.pipe(fs.createWriteStream('ada_lovelace.jpg'))
});
```


요청 메소드 명령어

편의를 위해 지원하는 모든 요청 메소드의 명령어를 제공합니다.

`axios.request(config)`

`axios.get(url[, config])`

`axios.delete(url[, config])`

`axios.head(url[, config])`

`axios.options(url[, config])`

`axios.post(url[, data[, config]])`

`axios.put(url[, data[, config]])`

`axios.patch(url[, data[, config]])`

- TanStack Query

웹 애플리케이션에서 서버 상태를 가져오고, 캐싱하고, 동기화하고, 업데이트하는 작업을 아주 쉽게 해주는 API

- useQuery

- 서버 상태 가져오기

```
const result = useQuery({ queryKey: ['todos'], queryFn: fetchTodoList })
```

쿼리를 수행한 결과 외에 아래 상태도 제공한다.

- `isPending` 또는 `status === 'pending'` - 쿼리에 아직 데이터가 없습니다.
- `isError` 또는 `status === 'error'` - 쿼리에서 오류가 발생했습니다.
- `isSuccess` 또는 `status === 'success'` - 쿼리가 성공했으며 데이터를 사용할 수 있습니다.

```
function Todos() {
  const { isPending, isError, data, error } = useQuery({
    queryKey: ['todos'],
    queryFn: fetchTodoList,
  })

  if (isPending) {
    return <span>Loading...</span>
  }

  if (isError) {
    return <span>Error: {error.message}</span>
  }

  // We can assume by this point that `isSuccess === true`
  return (
    <ul>
      {data.map((todo) => (
        <li key={todo.id}>{todo.title}</li>
      ))}
    </ul>
  )
}
```

****queryKey**

첫번째 요소 : 요청 자원명(리소스 식별자)

두번째 요소부터 : useQuery를 다시 수행시킬 기준 (해당 데이터가 변하면 다시 수행)

옵션	설명
queryKey	고유한 쿼리 식별자. 배열 형태로 지정합니다. (ex: ['user', userId])
queryFn	데이터를 fetch하는 함수 (비동기 함수). ex: () => axios.get(...)
enabled	쿼리를 자동 실행할지 여부. false 면 수동으로 실행 필요
select	데이터를 가공해서 반환할 수 있는 함수
staleTime	데이터가 오래됐다고 간주하기 전까지의 시간 (ms)
cacheTime	쿼리 결과가 메모리에 남아있는 시간 (ms)
refetchOnWindowFocus	윈도우 포커스될 때 다시 fetch할지 여부 (true , false , "always")
retry	실패 시 재시도 횟수 또는 재시도 여부
onSuccess	성공했을 때 콜백
onError	실패했을 때 콜백

■ useMutation

데이터를 생성/업데이트/삭제하거나 서버 사이드 이펙트를 수행하는 데 사용

```
const mutation = useMutation({
  mutationFn: (newTodo) => {
    return axios.post('/todos', newTodo)
  },
})
```

옵션	설명
mutationFn	실제로 서버에 요청할 비동기 함수. (필수)
onSuccess	요청 성공 시 실행할 콜백 함수
onError	요청 실패 시 실행할 콜백 함수
onSettled	성공이든 실패든 요청이 끝나면 실행