

웹서버보안프로그래밍

3. Next.js 소개 및 예제 프로젝트

중부대학교 정보보호학과

이병천 교수

sultan@joongbu.ac.kr

전체 목차

1. 강의 개요
2. 웹개발환경 구축
- 3. Next.js 소개**
4. MongoDB CRUD
5. 쇼핑몰 예제 프로젝트
6. PBL 팀프로젝트 발표

3. Next.js 프레임워크 소개

1. React와 Next.js
2. Next.js의 특징
3. 예제 프로젝트 - clerk 인증
4. 예제 프로젝트 - github API 활용
5. 예제 프로젝트 - 백엔드 API 활용
6. Service Deploy



1. React

- React 도입
 - 2013년 Facebook이 개발 공개
 - <https://react.dev/>
 - 프론트엔드 유저인터페이스 개발을 위한 컴포넌트 기반 라이브러리
 - 새로운 프로젝트 시작 명령어
 - > npx create-next-app@latest
 - > npx create-remix
 - > npx create-gatsby
 - > npx create-expo-app



Video.js

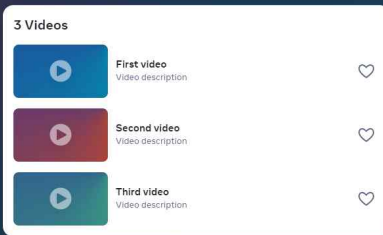
```
function Video({ video }) {  
  return (  
    <div>  
      <Thumbnail video={video} />  
      <a href={video.url}>  
        <h3>{video.title}</h3>  
        <p>{video.description}</p>  
      </a>  
      <LikeButton video={video} />  
    </div>  
  );  
}
```



컴포넌트 기반
유저인터페이스 생성

VideoList.js

```
function VideoList({ videos, emptyHeading }) {  
  const count = videos.length;  
  let heading = emptyHeading;  
  if (count > 0) {  
    const noun = count > 1 ? 'Videos' : 'Video';  
    heading = count + ' ' + noun;  
  }  
  return (  
    <section>  
      <h2>{heading}</h2>  
      {videos.map(video =>  
        <Video key={video.id} video={video} />  
      )}  
    </section>  
  );  
}
```



컴포넌트의 재사용

모듈화 프로그래밍

자바스크립트 프로그램과
UI의 상호작용

Full-stack with a Framework

- Production-grade React frameworks
 - <https://react.dev/learn/start-a-new-react-project>

Go full-stack with a framework

React is a library. It lets you put components together, but it doesn't prescribe how to do routing and data fetching. To build an entire app with React, we recommend a full-stack React framework like [Next.js](#) or [Remix](#).

<https://nextjs.org/>

```
> npx create-next-app
```

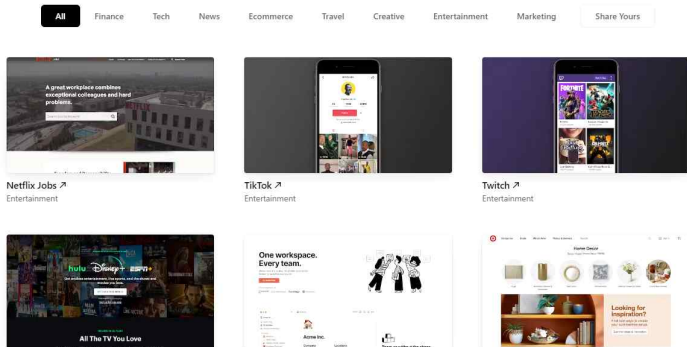
<https://remix.run/>

```
> npx create-remix
```

Full-stack React Framework로 Next.js를 첫번째로 추천함

Next.js를 적용한 웹서비스들

Meet thousands of beautiful
websites built with Next.js by Vercel



Netflix
TikTok
Twitch
Hulu
Notion
Target
Nike
HBO Max
AT&T
Realtor.com
Trip.com

2. Next.js의 특징

- 1. Rendering
- 2. Routing
- 3. Fullstack
- 4. Automatic code
- 5. React

Built-in Optimizations

Automatic Image, Font, and Script Optimizations for improved UX and Core Web Vitals.

Dynamic HTML Streaming

Instantly stream UI from the server, integrated with the App Router and React Suspense.

React Server Components

Add components without sending additional client-side JavaScript. Built on the latest React features.

Data Fetching

Make your React component async and await your data. Next.js supports both server and client data fetching.

CSS Support

Style your application with your favorite tools, including support for CSS Modules, Tailwind CSS, and popular community libraries.

Client and Server Rendering

Flexible rendering and caching options, including Incremental Static Regeneration (ISR), on a per-page level.

Server Actions

Run server code by calling a function. Skip the API. Then, easily revalidate cached data and update your UI in one network roundtrip.

Route Handlers

Build API endpoints to securely connect with third-party services for handling auth or listening for webhooks.

Advanced Routing & Nested Layouts

Create routes using the file system, including support for more advanced routing patterns and UI layouts.

Middleware

Take control of the incoming request. Use code to define routing and access rules for authentication, experimentation, and internationalization.

Next.js 14

The power of full-stack to the frontend. [Read the release notes.](#)



<https://www.perplexity.ai/> 에 질문

질문) next.js 프레임워크의 주요 기능들을 소개해줘.

답변)
Next.js는 React 기반의 강력한 웹 개발 프레임워크로, 다음과 같은 주요 기능들을 제공합니다:

렌더링 옵션

- 서버 사이드 렌더링 (SSR): 서버에서 페이지를 렌더링하여 초기 로딩 속도를 개선하고 SEO를 향상시킵니다.
- 정적 사이트 생성 (SSG): 빌드 시 정적 HTML 파일을 생성하여 매우 빠른 페이지 로딩을 제공합니다.
- 클라이언트 사이드 렌더링 (CSR): 필요에 따라 클라이언트에서 동적으로 콘텐츠를 렌더링할 수 있습니다.

개발 최적화

- 자동 코드 분할: 각 페이지에 필요한 코드만 로드하여 성능을 개선합니다.
- 파일 시스템 기반 라우팅: 직관적이고 간편한 라우팅 시스템을 제공합니다.
- API 라우트: 서버리스 함수를 쉽게 생성할 수 있는 API 엔드포인트를 제공합니다.

성능 최적화

- 이미지 최적화: 자동 이미지 최적화 기능으로 웹 성능을 향상시킵니다.
- 폰트 최적화: 웹 폰트 로딩을 최적화하여 사용자 경험을 개선합니다.

개발자 경험

- 핫 모듈 교체 (HMR): 코드 변경 사항을 실시간으로 반영하여 개발 생산성을 높입니다.
- TypeScript 지원: 기본적으로 TypeScript를 지원하여 타입 안정성을 제공합니다.
- 환경 변수 지원: 다양한 환경에서의 설정을 쉽게 관리할 수 있습니다.

확장성

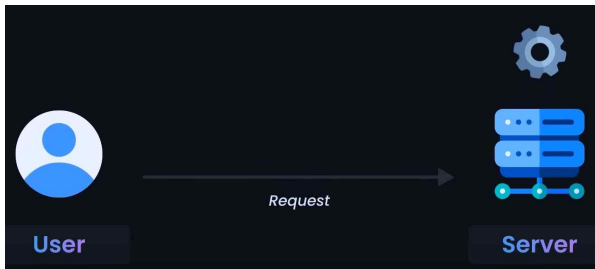
- 풀스택 지원: 백엔드 개발도 가능한 풀스택 프레임워크로 사용할 수 있습니다.
- 다양한 플러그인과 통합: 활발한 커뮤니티를 통해 다양한 플러그인과 통합 기능을 제공합니다.

이러한 기능들을 통해 Next.js는 개발자들이 빠르고 효율적으로 고성능 웹 애플리케이션을 구축할 수 있도록 지원합니다. 9

Rendering

- 1. Rendering: CSR vs SSR

- Rendering: Javascript 프로그램에서 HTML, CSS로 이루어진 화면을 생성하는 것
- CSR: Client-side rendering (화면 생성을 클라이언트 브라우저가 실행)
- SSR: Server-side rendering (서버가 화면을 생성하여 클라이언트에게 제공)



Rendering

- React
 - CSR만 제공
 - 서버는 클라이언트에게 SPA 전체 자바스크립트 프로그램을 제공하고 UI 구성을 클라이언트가 처리
 - 브라우저에서 웹페이지 초기 로딩에 시간이 걸림
 - 검색엔진에 노출되지 않음, Search Engine Optimization(SEO)에 취약
- Next.js
 - CSR과 SSR을 모두 제공하며 개발자가 선택하여 제공 가능
 - 기본적으로 SSR 방식으로 서버가 UI를 생성하여 브라우저에게 제공, SEO 기능 제공
 - 브라우저에서 직접 실행되어야 하는 부분만 CSR 방식으로 적용

Search Engine Optimization

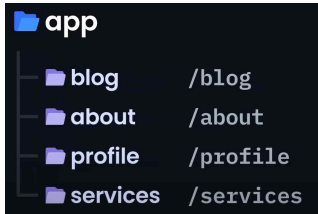
- 검색엔진이 웹페이지를 분석할 수 있어야 검색엔진에 키워드 등록 가능
 - Website visibility and ranking
 - SSR이 SEO에 유리함
 - Next.js는 기본적으로 서버가 pre-rendering된 콘텐츠를 클라이언트에게 제공



Source : Saabsoft

Routing

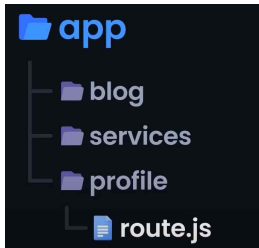
- React
 - React router를 이용하여 명시적 라우팅 설정 필요
- Next.js
 - 파일기반 라우팅 시스템
 - 라우팅 설정 불필요, 자동 제공



Fullstack

- Serverless API Routes

- API request를 처리하는 endpoint를 손쉽게 생성 가능
- 별도의 서버를 운영하지 않고 route.js 파일에 backend 기능을 설정할 수 있음
- 서버 운영의 효율성 향상

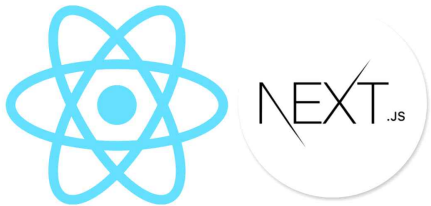


Automatic Code Splitting

- Code splitting
 - 대규모 자바스크립트 코드를 소규모 코드로 나누어 필요할때 로드할 수 있도록 운영
 - 초기 로드 시간을 줄이고 사용자 경험을 향상
- Automatic code splitting
 - Next.js에서는 code splitting을 자동 제공
 - 초기화면, 세부메뉴 등 로드시 효율성 향상
 - 대용량 이미지, 페이지 처리시 속도 향상

React

- Next.js는 React 기반 프레임워크
 - React 기술을 모두 사용 가능
 - React의 확장



Next.js 의 장점

- 웹사이트 개발시 필요한 많은 반복작업, 설정들을 프레임워크가 최적화된 방식으로 자동으로 처리. 시간 절약.
 - Routing
 - Code splitting
 - Search engine optimization
 - Rendering
- 개발자에게 React 프로젝트 자체의 개발에 집중할 수 있게 도와줌.

기본 템플릿 next.js app 생성

- 프로젝트 폴더 만들기
 - 예) test
- VSCode로 프로젝트 폴더 열기
- 터미널 열기
- 기본 템플릿 app 생성
 - > npx create-next-app@latest .
 - 현재 열려있는 프로젝트 폴더에 생성 (.은 현재 폴더를 나타냄)
 - 폴더를 지정하지 않으면 폴더이름을 요구함

VSCode에서 프로젝트 폴더 열기가 중요함

프로젝트 루트 폴더에 node_modules 폴더가
생성되고 패키지가 설치되어야 함

에러가 발생하는 경우

```
D:\WebDev\2024-2\nextjs\test>npx create-next-app@latest .
```

```
npm error code ENOENT
```

```
npm error syscall lstat
```

```
npm error path C:\Users\User\AppData\Roaming\npm
```

```
npm error errno -4058
```

```
npm error enoent ENOENT: no such file or directory, lstat 'C:\Users\User\AppData\Roaming\npm'
```

```
npm error enoent This is related to npm not being able to find a file.
```

```
npm error enoent
```

```
npm notice
```

```
npm notice New patch version of npm available! 10.8.2 -> 10.8.3
```

```
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.8.3
```

```
npm notice To update run: npm install -g npm@10.8.3
```

```
npm notice
```

```
npm error A complete log of this run can be found in: C:\Users\User\AppData\Local\npm-cache\_logs\2024-09-03T08_12_05_344Z-debug-0.log
```

npx 명령어가 설치되지 않아서 에러 발생

```
D:\WebDev\2024-2\nextjs\test>npm i -g npx
```

```
npm warn deprecated npx@10.2.2: This package is now part of the npm CLI.
```

```
added 1 package in 4s
```

```
1 package is looking for funding
```

```
run `npm fund` for details
```

npx 명령어를 시스템에 설치

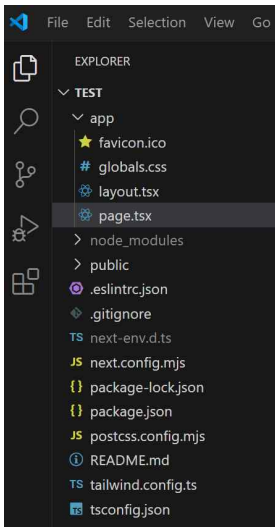
선택사항 입력

```
D:\WebDev\2024-2\nextjs\test>npx create-next-app@latest .  
Need to install the following packages:  
create-next-app@14.2.7  
Ok to proceed? (y)  
  
✓ 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 to use `src/` directory? ... No / Yes  
✓ Would you like to use App Router? (recommended) ... No / Yes  
✓ Would you like to customize the default import alias (@/*)? ... No / Yes  
Creating a new Next.js app in D:\WebDev\2024-2\nextjs\test.  
  
Using npm.  
  
Initializing project with template: app-tw  
  
Installing dependencies:  
- react  
- react-dom  
- next
```

Package.json

```
{
  "name": "test",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint"
  },
  "dependencies": {
    "react": "^18",
    "react-dom": "^18",
    "next": "14.2.7"
  },
  "devDependencies": {
    "typescript": "^5",
    "@types/node": "^20",
    "@types/react": "^18",
    "@types/react-dom": "^18",
    "postcss": "^8",
    "tailwindcss": "^3.4.1",
    "eslint": "^8",
    "eslint-config-next": "14.2.7"
  }
}
```

- 프로젝트의 뼈대, 설정사항을 관리하는 파일
- 스크립트 명령
 - > npm run dev : 개발 서버 실행
 - > npm run build : 컴파일, 빌드하여 SSR 상태의 서버 생성
 - > npm run start : 빌드된 서버를 실행
 - > npm run lint : 에러 여부 검사
- Dependencies
 - 이 프로젝트에 설치된 외부 패키지들과 버전 등록
 - 이 패키지들은 npm install 명령으로 언제든지 다시 설치할 수 있음
- devDependencies
 - 개발시에만 사용하는 패키지들의 목록
 - 빌드하여 컴파일하면 더 이상 사용되지 않음



프로젝트 폴더 분석

/app/globals.css : 프로젝트 전체에 적용되는 CSS 파일. Tailwind 적용.

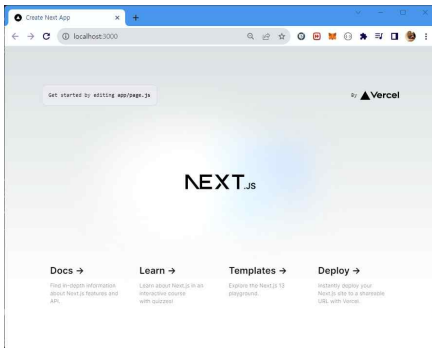
/app/layout.tsx : 프로젝트의 레이아웃을 정의 (정해진 파일명)

/app/page.tsx : 루트 페이지의 컴포넌트 (정해진 파일명)

/public : 프로젝트에서 사용되는 정적 자료들을 관리하기 위한 폴더

기본 템플릿 앱 열기

- 개발 서버 열기
 - > npm run dev
 - 반응형 레이아웃 적용
 - <http://localhost:3000>



고정된 특수 파일명

- page.tsx
 - UI를 가지는 페이지 컴포넌트
- layout.tsx
 - 페이지 컴포넌트의 레이아웃을 정의
- loading.tsx
 - 페이지가 로딩되기 전에 표시해주는 페이지
- error.tsx
 - 에러 발생시 표시해주는 페이지
- route.ts
 - Backend API handler를 정의
- globals.css
 - 전체 프로젝트에 적용되는 CSS 설정

layout.tsx (고정된 파일명)

```
import type { Metadata } from "next";
import { Inter } from "next/font/google";
import "../globals.css";

const inter = Inter({ subsets: ["latin"] });

export const metadata: Metadata = {
  title: "Create Next App",
  description: "Generated by create next app",
};

export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <html lang="en">
      <body className={inter.className}>{children}</body>
    </html>
  );
}
```

globals.css를 전체 프로젝트에 적용

구글 폰트 Inter를 사용

메타데이터 관리

RootLayout 지정

모든 페이지들에 적용되는 레이아웃 설정

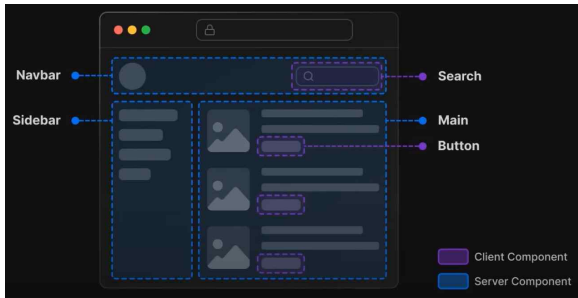
Navbar, Footer 등을 지정 가능

Children은 여러 세부 페이지들을 나타냄

page.tsx (고정된 파일명)

- 루트 페이지 컴포넌트

- page.tsx는 UI를 나타내는 페이지 컴포넌트의 정해진 파일명
- Server component (SSR) Next.js의 모든 페이지는 기본적으로 서버에서 렌더링하여 제공
- Client component (CSR)로 제공하려면 컴포넌트 첫부분에 'use client'를 명시



When to use Server and Client Components?

To simplify the decision between Server and Client Components, we recommend using Server Components (default in the `app` directory) until you have a use case for a Client Component.

This table summarizes the different use cases for Server and Client Components:

What do you need to do?	Server Component	Client Component
Fetch data.	✓	✗
Access backend resources (directly)	✓	✗
Keep sensitive information on the server (access tokens, API keys, etc)	✓	✗
Keep large dependencies on the server / Reduce client-side JavaScript	✓	✗
Add interactivity and event listeners (<code>onClick()</code> , <code>onChange()</code> , etc)	✗	✓
Use State and Lifecycle Effects (<code>useState()</code> , <code>useReducer()</code> , <code>useEffect()</code> , etc)	✗	✓
Use browser-only APIs	✗	✓
Use custom hooks that depend on state, effects, or browser-only APIs	✗	✓
Use React Class components ↗	✗	✓

Server component vs Client component

Server component (default)

- 정적인 콘텐츠를 제공하는 경우
- 외부 데이터를 가져오는 작업이 필요한 경우
- 백엔드 서버에 저장된 자료를 사용하는 경우
- 서버에 저장되는 보안정보(엑세스토큰, API키)를 사용하는 경우

Client component ('use client' 명시)

- 사용자의 이벤트에 따라 반응하는 동적 페이지
- `useState`, `useEffect` 등 브라우저에서 상태를 관리하는 react hook을 적용하는 경우

Routing

- File-based routing
 - `/posts/page.tsx`
 - 폴더와 파일을 생성하면 라우팅이 생성됨
- Nested routing
 - `/posts/new/page.tsx`
 - 중첩된 폴더구조의 라우팅 생성
- Dynamic routing
 - **`/posts/[postId]/page.tsx`**
 - 사용자의 선택, 응답에 따라 생성되는 라우팅
- Catch-all routing
 - **`/posts/[...slug]/page.tsx`**
 - 복잡한 폴더 구조의 라우팅을 자동 처리

Data Fetching

- 외부 데이터를 가져와서 사용하는 3가지 방법
- 1. Server Side Rendering (SSR)
 - 서버에서 데이터를 불러와서 html, css 페이지를 만들어서 클라이언트에게 제공
 - 매 요청시마다 새로운 데이터로 렌더링을 새로 함. 콘텐츠가 빠르게 바뀌는 환경에 적합
- 2. Static Site Generation (SSG)
 - 웹서비스 빌드시 서버가 외부 데이터를 불러와서 고정된 페이지를 만들고 이후로는 이 고정된 페이지로 서비스를 제공함
 - 콘텐츠 변경이 거의 없는 고정된 데이터인 경우에 적합
- 3. Incremental Static Generation (ISG)
 - 서버는 주어진 일정 시간 간격으로 데이터를 읽어오고 페이지를 새로 생성함
 - SSR과 SSG의 절충 형태

Backend API Handlers

- 백엔드 전용 폴더 생성 : /app/api
 - /app/api/users/route.ts
 - /app/api/posts/route.ts
 - 백엔드에서도 file-based routing 적용
 - API route를 정의하는 경우 route.ts 파일 내에 정의해야 함

```
1  import type { NextApiRequest, NextApiResponse } from 'next'
2
3  type ResponseData = {
4    message: string
5  }
6
7  export default function handler(
8    req: NextApiRequest,
9    res: NextApiResponse<ResponseData>
10 ) {
11   res.status(200).json({ message: 'Hello from Next.js!' })
12 }
```

3. 예제 프로젝트 – Clerk 인증

- 제공하는 기능
 - Next.js, Tailwind CSS, Typescript 프로젝트
 - Layout, Menu design
 - Clerk authentication
 - 외부 API 활용 - api.github.com
 - Backend API service
 - CRUD with MongoDB

Next.js 프로젝트 생성

- 새로운 프로젝트 생성
 - > npx create-next-app@latest
- 프로젝트 폴더 열기
- 개발 서버 실행
 - > npm run dev

확장자 이름 변경
page.js → page.jsx
layout.js → layout.jsx

```
D:\WebDev\2023-2\nextjs> npx create-next-app@latest
√ What is your project named? ... clerk-api
√ 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 to use `src/` directory? ... No / Yes
√ Would you like to use App Router? (recommended) ... No / Yes
√ Would you like to customize the default import alias? ... No / Yes
Creating a new Next.js app in D:\WebDev\2023-2\nextjs\clerk-api.
```


루트 페이지 수정

/app/page.tsx

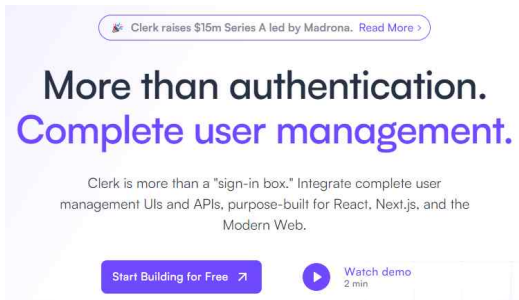
```
export default function Home() {  
  return (  
    <>  
    <h1>Welcome Home!</h1>  
    </>  
  )  
}
```

/app/globals.css

```
@tailwind base;  
@tailwind components;  
@tailwind utilities;
```

Clerk 회원가입 및 로그인

- Clerk.com
 - <https://clerk.com/>
 - 종합 인증 서비스
 - 각종 소셜로그인 서비스 제공
 - 사용자 관리를 clerk이 대신함
- 회원가입 및 로그인
 - 회원가입
 - 로그인
- 앱 생성하기
 - Start building > add application



Clerk with Next.js

<https://clerk.com/docs/quickstarts/nextjs>

- **Use Clerk with Next.js**

- 1. Install @clerk/nextjs
- 2. Set environment variables
- 3. Add Middleware to your application
- 4. Add <ClerkProvider> and components to your app
- 5. Protect routes using clerk Middleware
- 6. Create custom sign-in, sign-up pages
- 7. Read user and session data



1 Install `@clerk/nextjs`

The package to use with Clerk and NextJS.

`npm` `yarn` `pnpm`

terminal

```
npm install @clerk/nextjs
```

2 Set your environment variables

Add these keys to your `.env.local` or create the file if it doesn't exist. Retrieve these keys anytime from the API keys page.

`.env.local`

```
NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY=pk_test_aW1t  
CLERK_SECRET_KEY=.....
```

3 Update `middleware.ts`

Update your middleware file or create one at the root

`middleware.ts`

(1) Clerk 패키지 설치

- (1) Clerk 패키지 설치
 - > npm i @clerk/nextjs

```
"dependencies": {  
  "@clerk/nextjs": "^5.4.0",  
  "next": "14.2.7",  
  "react": "^18",  
  "react-dom": "^18"  
},
```

(2) 환경변수 설정

- Add Application (앱 생성)
 - Application Name 입력
- User sign in 방법 선택
 - Email address (default)
 - Google (default)
 - Github (추가)
 - 기타 다른 인증수단 추가 가능
 - Create application
- API keys 복사하여 .env.local 파일에 저장

API Keys

Paste the keys in the code snippet below into your `.env.local` file

`.env.local`

`.env.local`

```
1 NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY=pk_test_a2Vibi0ZWfSLTU4LmNsZXJrLmFjY291bnRzLmRldiQ
2 CLERK_SECRET_KEY=sk_test_.....
```

[CONTINUE IN DOCS](#)

Let's build your <SignIn />

Application name

clerk-auth

How will your users sign in?

- Email address Popular ☒
- Phone number ☐
- Username ☐
- Google Popular ☒
- Facebook ☐
- Apple ☐
- GitHub ☒

[Hide 19 items](#)

[CANCEL](#)

[CREATE APPLICATION](#)

(3) Middleware 설정

- 프로젝트 루트폴더에 middleware.ts 파일 생성
 - 구체적인 인증 설정 가능

/middleware.ts

```
import { clerkMiddleware } from '@clerk/nextjs/server'

export default clerkMiddleware()

export const config = {
  matcher: [
    // Skip Next.js internals and all static files, unless found in search params
    '/((?!_next|[^?]*\\.?(?:html?|css|js(?:on)?|jpeg?|webp|png|gif|svg|ttf|woff2?|ico|csv|docx?|xlsx?|zip|webmanifest)).*)',
    // Always run for API routes
    '/(api|trpc)(.*)',
  ],
}
```

(4) ClerkProvider 추가

/app/layout.tsx

```
import type { Metadata } from 'next'
import { Inter } from 'next/font/google'
import './globals.css'
```

```
import { ClerkProvider } from '@clerk/nextjs'
```

← Clerk 인증을 제공하는 provider

```
const inter = Inter({ subsets: ['latin'] })
```

```
export const metadata: Metadata = {
  title: 'Create Next App',
  description: 'Generated by create next app',
}
```

```
export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode
}>) {
  return (
    <ClerkProvider>
      <html lang="en">
        <body className={inter.className}>{children}</body>
      </html>
    </ClerkProvider>
  )
}
```

/tailwind.config.ts

```
/** @type {import("tailwindcss").Config} */
module.exports = {
  content: [
    './pages/**/*.{js,ts,jsx,tsx,mdx}',
    './components/**/*.{js,ts,jsx,tsx,mdx}',
    './app/**/*.{js,ts,jsx,tsx,mdx}',
  ],
  theme: {
    extend: {
      backgroundImage: {
        'gradient-radial': 'radial-gradient(var(--tw-gradient-stops))',
        'gradient-conic':
          'conic-gradient(from 180deg at 50% 50%, var(--tw-gradient-stops))',
      },
    },
  },
  container: {
    center: true,
  },
  plugins: [],
}
```

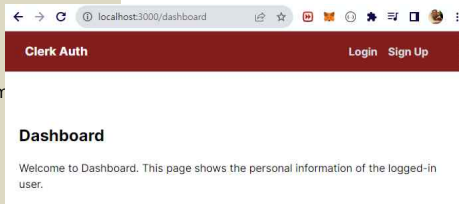
← Container 클래스는 가운데 정렬

(5) Middleware를 이용한 페이지 접근 제어

- 로그인된 사용자만 접근할 수 있는 페이지 설정하기
- 대시보드 페이지 만들기
 - /dashboard는 로그인된 사용자의 개인정보를 보여주는 페이지

/app/dashboard/page.tsx

```
const DashboardPage = () => {  
  return (  
    <div>  
      <h1 className="text-2xl font-bold mb-5">Dashboard</h1>  
      <p className="mb-5">  
        Welcome to Dashboard. This page shows the personal information of the  
        logged-in user.  
      </p>  
    </div>  
  )  
}  
export default DashboardPage
```



Middleware를 이용한 페이지 접근 제어

/middleware.ts

```
import { clerkMiddleware, createRouteMatcher } from '@clerk/nextjs/server'

const isProtectedRoute = createRouteMatcher(['/dashboard(.*)', '/forum(.*)'])
const isPublicRoute = createRouteMatcher(['/', '/sign-in(.*)', '/sign-up(.*)'])

export default clerkMiddleware((auth, req) => {
  if (isProtectedRoute(req)) auth().protect()
  if (!isPublicRoute(req)) auth().protect()
})

export const config = {
  matcher: [
    // Skip Next.js internals and all static files, unless found in search params
    '/(?!_next|[^?]*\\.(?:html?|css|js(?!on)|jpe?g|webp|png|gif|svg|ttf|woff2?|ico|csv|docx?|xlsx?|zip|webmanifest)).*',
    // Always run for API routes
    '/(api|trpc)(.*)',
  ],
}
```

보호해야 할 루트를 지정
공개할 루트를 지정

(6) Sign-up, Sign-in 페이지 만들기

/app/sign-in/[...sign-in]/page.tsx

```
import { SignIn } from '@clerk/nextjs'

export default function SignInPage() {
  return <SignIn />
}
```

/app/sign-up/[...sign-up]/page.tsx

```
import { SignUp } from '@clerk/nextjs'

export default function SignUpPage() {
  return <SignUp />
}
```

회원가입, 로그인 요청을 처리하는 catch-all routing

/.env.local

```
NEXT_PUBLIC_CLERK_SIGN_IN_URL=/sign-in
NEXT_PUBLIC_CLERK_SIGN_UP_URL=/sign-up
```

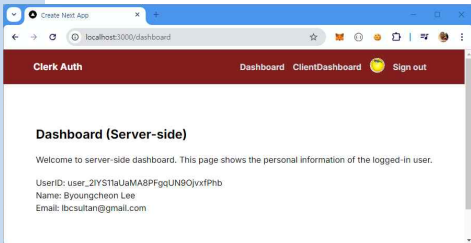
인증 관련 URL을 환경변수로 설정

(7) Session, User data 읽기

/app/dashboard/page.tsx – server-side

```
import { auth, currentUser } from '@clerk/nextjs/server'

export default async function Dashboard() {
  const { userId } = auth()
  const user = await currentUser()
  return (
    <div>
      <h1 className="text-2xl font-bold mb-5">Dashboard (Server-side)</h1>
      <p className="mb-5">
        Welcome to server-side dashboard. This page shows the personal
        information of the logged-in user.
      </p>
      {userId && (
        <div>
          <p>UserID: {userId}</p>
          <p>Name: {user?.fullName}</p>
          <p>Email: {user?.primaryEmailAddress?.emailAddress}</p>
        </div>
      )}
    </div>
  )
}
```



Session, User data 읽기

/app/client-dashboard/page.tsx – server-side

```
'use client'
import { useAuth, useUser } from '@clerk/nextjs'

export default function ClientDashboard() {
  const { isLoading: isLoadingAuth, userId, sessionId, getToken } = useAuth()
  const { isLoading: isLoadingUser, isSignedIn, user } = useUser()

  if (!isLoadingAuth || !userId) {
    return null
  }

  if (!isLoadingUser || !isSignedIn) {
    return null
  }

  return (
    <div>
      <h1 className="text-2xl font-bold mb-4">Dashboard (client-side)</h1>
      <p>
        Hello, {user.fullName} <br />
        Your user ID: {userId} <br />
        Your current active session: {sessionId}
      </p>
    </div>
  )
}
```

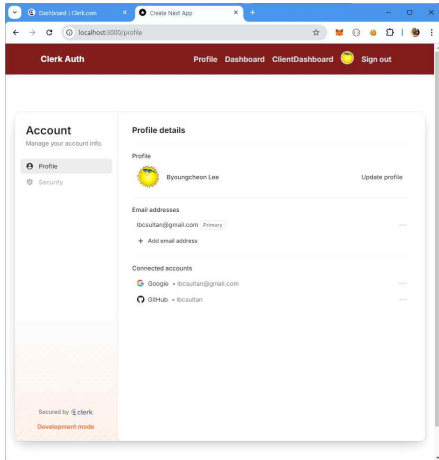
Profile 페이지 만들기

/app/profile/[...profile]/page.tsx

```
import { UserProfile } from '@clerk/nextjs'

export default function ProfilePage() {
  return (
    <div>
      <UserProfile />
    </div>
  )
}
```

헤더 메뉴에 Profile 등록



```
import {
  SignedIn,
  SignedOut,
  SignInButton,
  SignOutButton,
  SignUpButton,
  UserButton,
} from '@clerk/nextjs'
import Link from 'next/link'
import React from 'react'

export default function Header() {
  return (
    <div>
      <nav className="bg-red-900 py-4 px-8">
        <div className="flex items-center justify-between container">
          <div className="flex items-center font-bold">
            <Link href="/">
              <div className="text-lg text-white"> Clerk Auth </div>
            </Link>
          </div>
          <div className="flex items-center font-bold">
            <SignedOut>
              <div className="text-gray-300 hover:text-white mr-4">
                <SignInButton />
              </div>
              <div className="text-gray-300 hover:text-white mr-4">
                <SignUpButton />
              </div>
            </SignedOut>
          </div>
        </div>
      </nav>
    </div>
  )
}
```

로그인이 안된 상태: sign-in, sign-up 메뉴를 보여줌

...계속...

```
<SignedIn>
  <Link
    href="/profile"
    className="text-gray-300 hover:text-white mr-4"
  >
    Profile
  </Link>
  <Link
    href="/dashboard"
    className="text-gray-300 hover:text-white mr-4"
  >
    Dashboard-Server
  </Link>
  <Link
    href="/client-dashboard"
    className="text-gray-300 hover:text-white mr-4"
  >
    Dashboard-Client
  </Link>
  <div className="text-gray-300 hover:text-white mr-4">
    <UserButton />
  </div>
  <div className="text-gray-300 hover:text-white mr-4">
    <SignOutButton />
  </div>
</SignedIn>
</div>
</nav>
</div>
}
```

로그인된 상태: dashboard, 사용자버튼, sign-out 메뉴를 보여줌

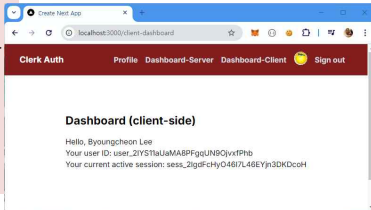
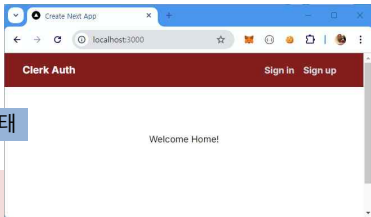
메뉴 만들기

- Layout에 Header 등록

/app/layout.tsx

```
<ClerkProvider>
  <html lang="en">
    <body className={inter.className}>
      <Header />
      <main className="container">
        <div className="flex items-start justify-center min-h-screen">
          <div className="mt-20">{children}</div>
        </div>
      </main>
    </body>
  </html>
</ClerkProvider>
```

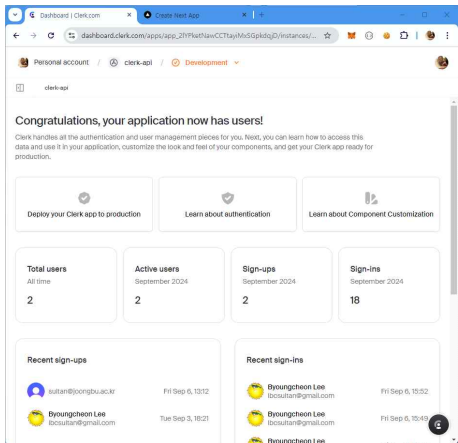
로그인이 안된 상태



로그인된 상태

Clerk dashboard

- <https://dashboard.clerk.com/>
- Clerk 대시보드에서 인증 현황 표시
 - Clerk.com에서 사용자 관리를 대신함
 - Email 회원가입, social signup



4. 예제 프로젝트 – github API 활용

- Github.com에서 제공하는 API 서비스
 - <https://api.github.com/>
 - <https://api.github.com/users/lbcsultan>
 - <https://api.github.com/users/lbcsultan/repos>
- 이들 API를 활용하기 위한 페이지를 개발
 - /app/repos/page.tsx 생성
 - 메뉴에 등록
- React-icons 패키지 설치
 - > npm install react-icons
 - FontAwsom 아이콘 활용 예정
 - <https://react-icons.github.io/react-icons/>

메인 페이지

/app/repos/page.tsx

```
import Link from 'next/link'
import { FaStar, FaCodeBranch, FaEye } from 'react-icons/fa'
```

```
const username = 'bradtraversy'
```

```
export default async function ReposPage() {
  const response = await fetch('https://api.github.com/users/${username}/repos')
  await new Promise((resolve) => setTimeout(resolve, 1000))
  const repos = await response.json()
```

```
  return (
    <div>
      <h2 className="text-2xl font-bold mb-4">
        Github Repositories of {username}
      </h2>
      <ul>
```

... 정보 가져오기 ...

```
      </ul>
    </div>
  )
}
```

Fetch를 이용한 외부 API 서비스
정보 가져오기

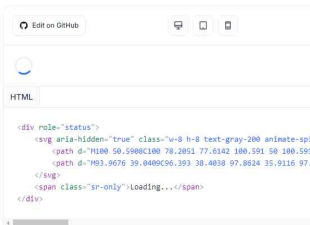
로딩페이지 실습

```
{repos.map((repo: any) => (
  <li key={repo.id} className="bg-gray-100 m-4 p-4 rounded-md">
    <Link href={`/${repo.name}`}>
      <h3 className="text-xl font-bold">{repo.name}</h3>
      <p>{repo.description}</p>
      <div className="flex justify-between items-center">
        <span className="flex items-center gap-1">
          <FaStar /> {repo.stargazers_count}
        </span>
        <span className="flex items-center gap-1">
          <FaCodeBranch /> {repo.forks_count}
        </span>
        <span className="flex items-center gap-1">
          <FaEye /> {repo.stargazers_count}
        </span>
      </div>
    </Link>
  </li>
)
)
})
```

Loading 페이지 작성

/app/loading.tsx

- Tailwind loading spinner 검색
 - <https://flowbite.com/docs/components>



- /app/loading.tsx 생성

```
const LoadingPage = () => {
  return (
    <div className="flex min-h-screen justify-center items-center">
      <svg
        aria-hidden="true"
        class="w-8 h-8 mr-2 text-gray-200 animate-spin dark:text-gray-600 fill-blue-600"
        viewBox="0 0 100 101"
        fill="none"
        xmlns="http://www.w3.org/2000/svg"
      >
        <path
          d="M100 50.5908C100 78.2051 77.6142 100.591 50 100.591C22.3858 100.591 0 78.2051 0 50.5908C22.9766 22.3858 50.5908 0 50.5908C77.6142 0 100.5908 22.9766 100 50.5908ZM9.08144 50.5908C9.08144 73.1895 27.4013 91.5094 50.5908 91.5094C72.9587 91.5094 90.9186 73.1895 90.9186 50.5908C90.9186 27.9921 72.9587 9.67226 50.5908 9.67226C27.4013 9.67226 9.08144 27.9921 9.08144 50.5908Z"
          fill="currentColor"
        />
        <path
          d="M93.9676 39.0409C96.393 38.4038 97.8624 35.9116 97.0079 33.5539C95.2932 28.8227 92.871 24.3692 89.8167 20.348C85.8452 15.1192 80.8826 10.7238 75.2124 7.41289C69.5422 4.10194 63.2754 1.94025 56.7698 1.05124C51.7666 0.367541 46.6976 0.446843 41.7345 1.27873C39.2613 1.69328 37.813 4.19778 38.4501 6.62326C39.0873 9.04874 41.5694 10.4717 44.0505 10.1071C47.8511 9.54855 51.7191 9.52689 55.5402 10.0491C60.8642 10.7766 65.9928 12.5457 70.6331 15.2552C75.2735 17.9648 79.3347 21.5619 82.5849 25.841C84.9175 28.9121 86.7997 32.2913 88.1811 35.8758C89.083 38.2158 91.5421 39.6781 93.9676 39.0409Z"
          fill="currentFill"
        />
      </svg>
      <span className="sr-only">Loading...</span>
    </div>
  )
}

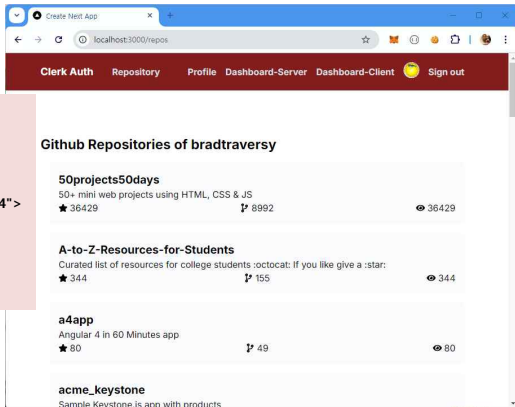
export default LoadingPage
```

메인 페이지

- 메뉴에 추가

/components/Header.tsx

```
<div className="flex items-center font-bold">
  <Link href="/">
    <div className="text-lg text-white"> Clerk Auth </div>
  </Link>
</div>
<div className="flex items-center font-bold">
  <Link href="/repos" className="text-gray-300 hover:text-white mr-4">
    Repository
  </Link>
</div>
<div className="flex items-center font-bold">
  ... 중략 ...
</div>
```



Repo 상세보기

/app/repos/[name]/page.tsx

```
import Repo from '@components/Repo'
import RepoDirs from '@components/RepoDirs'
import Link from 'next/link'
import React, { Suspense } from 'react'
```

```
export default function RepoPage({ params }: { params: { name: string } }) {
  return (
    <div className="flex flex-col justify-start items-start max-w-lg">
      <Link
        href="/repos"
        className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded mb-4"
      >
        Back to Repositories
      </Link>
      <Suspense fallback=<div>Loading repo...</div>>
        <Repo name={params.name} />
      </Suspense>
      <Suspense fallback=<div>Loading directories...</div>>
        <RepoDirs name={params.name} />
      </Suspense>
    </div>
  )
}
```

Dynamic routing 기능 활용
/app/repos/[name]/page.jsx 생성

2개의 컴포넌트를 활용

<Repo /> Repo 정보 보기
<RepoDirs /> 하부 디렉토리 정보 보기

```
import Link from 'next/link'
import { FaStar, FaCodeBranch, FaEye } from 'react-icons/fa'
```

```
interface RepoProps {
  name: string
}
```

```
const Repo: React.FC<RepoProps> = async ({ name }) => {
  const username = 'bradtraversy'
  await new Promise((resolve) => setTimeout(resolve, 1000))
  const response = await fetch(
    `https://api.github.com/repos/${username}/${name}`
  )
  const repo = await response.json()
  // console.log(repo)
```

/components/Repo.tsx

```
return (
  <div>
    <h3 className="text-xl font-bold">
      <Link href={`https://github.com/${username}/${name}`}>{repo.name}</Link>
    </h3>
    <p>{repo.description}</p>
    <div className="flex justify-between items-center mb-4">
      <span className="flex items-center gap-1">
        <FaStar /> {repo.stargazers_count}
      </span>
      <span className="flex items-center gap-1">
        <FaCodeBranch /> {repo.forks_count}
      </span>
      <span className="flex items-center gap-1">
        <FaEye /> {repo.stargazers_count}
      </span>
    </div>
  </div>
)
}
export default Repo
```

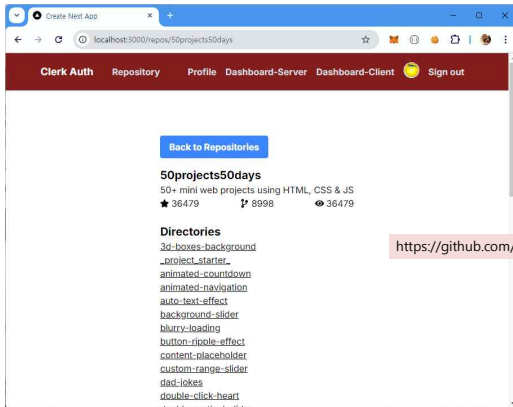
```
import Link from 'next/link'
```

```
interface RepoProps {
  name: string
}
```

```
const RepoDirs: React.FC<RepoProps> = async ({ name }) => {
  const username = 'bradtraversy'
  await new Promise((resolve) => setTimeout(resolve, 2000))
  const response = await fetch(
    `https://api.github.com/repos/${username}/${name}/contents`
  )
  const contents = await response.json()
  const dirs = contents.filter((content: any) => content.type === 'dir')
  // console.log(dirs)
```

```
return (
  <div className="mt-2">
    <h3 className="text-xl font-bold">Directories</h3>
    <ul>
      {dirs.map((dir: any) => (
        <li key={dir.path}>
          <Link
            className="underline"
            href={`https://github.com/${username}/${name}/tree/master/${dir.path}`}
          >
            {dir.path}
          </Link>
        </li>
      ))}
    </ul>
  </div>
)
}
export default RepoDirs
```

테스트



<http://localhost:3000/repos/50projects50days>

<https://github.com/bradtraversy/50projects50days>

<https://github.com/bradtraversy/50projects50days/tree/master/3d-boxes-background>

SSG, SSR, ISG

- 3가지 렌더링 방식

- 1. SSG : Static Site Generation

- 웹서비스 빌드시에 정적 페이지를 생성하고 이것으로 서비스 제공
 - 장점: 서버의 부담이 적음
 - 단점: 자료가 변경되는 경우 반영할 수 없음

- 2. SSR : Server-Side Rendering

- 클라이언트의 요청시마다 페이지를 새로 생성하여 서비스
 - 장점: 자료의 변경을 실시간으로 반영할 수 있음
 - 단점: 클라이언트의 요청이 많은 경우 서버의 부담이 큼

- 3. ISR : Incremental Static Regeneration

- 일정 시간마다 자료 업데이트를 반영하여 정적 페이지를 갱신하고 이것으로 서비스 제공
 - 위 두가지 방식의 절충 형태

/app/repos/page.tsx

```
import Link from 'next/link'
import { FaStar, FaCodeBranch, FaEye } from 'react-icons/fa'

const username = 'bradtraversy'

export default async function ReposPage() {
  // 1. SSG: Static Site Generation
  // const response = await fetch(`https://api.github.com/users/${username}/repos`)

  // 2. SSR: Server-Side Rendering
  // const response = await fetch(
  //   `https://api.github.com/users/${username}/repos`,
  //   { cache: 'no-store' }
  // )

  // 3. ISR: Incremental Static Generation
  const response = await fetch(
    `https://api.github.com/users/${username}/repos`,
    { next: { revalidate: 60 } }
  )

  await new Promise((resolve) => setTimeout(resolve, 1000))
  const repos = await response.json()
  // console.log(repos)
```

... 중략

1. SSG
서버 빌드시 정적 페이지 생성
2. SSR
캐시를 저장하지 않는 옵션
요청시마다 생성하여 제공
3. ISR
일정 시간 간격으로 페이지를 갱신

요구사항에 따라 간단하게 선택하여 사용 가능
서버의 performance를 관리 가능

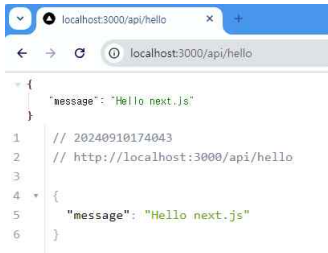
5. 예제 프로젝트 - 백엔드 API 활용

- /app/api 폴더 생성
 - Next.js에서 백엔드 서비스 제공 가능
 - 관습적으로 폴더명을 api로 씀
 - 백엔드 API의 파일명은 route.ts로 정해진 파일명을 사용
- /app/api/hello/route.ts

/app/api/hello/route.ts

```
import { NextResponse } from 'next/server'

export async function GET() {
  const data = {
    message: 'Hello next.js',
  }
  return NextResponse.json(data)
}
```



```
[
  {
    "id": "1",
    "title": "React Front To Back",
    "description": "Learn Modern React, Including Hooks, Context API, Full Stack MERN & Redux By Building Real Life Projects.",
    "link": "https://www.traversymedia.com/Modern-React-Front-To-Back-Course",
    "level": "Beginner"
  },
  {
    "id": "2",
    "title": "Node.js API Masterclass",
    "description": "Build an extensive RESTful API using Node.js, Express, and MongoDB",
    "link": "https://www.traversymedia.com/node-js-api-masterclass",
    "level": "Intermediate"
  },
  {
    "id": "3",
    "title": "Modern JavaScript From The Beginning",
    "description": "37 hour course that teaches you all of the fundamentals of modern JavaScript.",
    "link": "https://www.traversymedia.com/modern-javascript-2-0",
    "level": "All Levels"
  },
  {
    "id": "4",
    "title": "Next.js Dev To Deployment",
    "description": "Learn Next.js by building a music event website and a web dev blog as a static website",
    "link": "https://www.traversymedia.com/next-js-dev-to-deployment",
    "level": "Intermediate"
  },
  {
    "id": "5",
    "title": "50 Projects in 50 Days",
    "description": "Sharpen your skills by building 50 quick, unique & fun mini projects.",
    "link": "https://www.traversymedia.com/50-Projects-In-50-Days",
    "level": "Beginner"
  }
]
```

Courses API 서비스

DB에 저장된 교육코스를
안내하는 페이지 작성

편의상 DB를 활용하지 않고
data.json 파일의 내용을
서비스하도록 구성

/app/api/courses/route.ts

```
import { NextResponse } from 'next/server'
import { v4 as uuidv4 } from 'uuid'
import courses from './data.json'

export async function GET() {
  return NextResponse.json(courses)
}

export async function POST(request: any) {
  const { title, description, level, link } = await request.json()

  const newCourse = {
    id: uuidv4(),
    title,
    description,
    level,
    link,
  }

  courses.push(newCourse)

  return NextResponse.json(courses)
}
```

Courses API 서비스

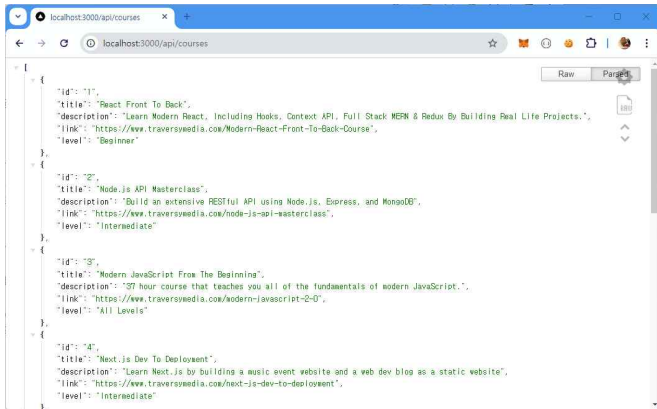
API 서비스를 제공

GET 요청에 대해 data.json 파일의 내용을 서비스

POST 요청에 대해 클라이언트의 정보를 바탕으로
newCourse를 생성하고 이것을 courses 배열에 추가함

id를 생성하는데 uuid 패키지를 이용함. 설치 필요
> npm install uuid
> npm install -D @types/uuid

API 서비스 제공 확인



```
[{"id": "1", "title": "React Front To Back", "description": "Learn Modern React, Including Hooks, Context API, Full Stack MERN & Redux By Building Real Life Projects.", "link": "https://www.traversymedia.com/Modern-React-Front-To-Back-Course", "level": "Beginner"}, {"id": "2", "title": "Node.js API Masterclass", "description": "Build an extensive RESTful API using Node.js, Express, and MongoDB.", "link": "https://www.traversymedia.com/node-js-api-masterclass", "level": "Intermediate"}, {"id": "3", "title": "Modern JavaScript From The Beginning", "description": "37 hour course that teaches you all of the fundamentals of modern JavaScript.", "link": "https://www.traversymedia.com/modern-javascript-2-0", "level": "All Levels"}, {"id": "4", "title": "Next.js Dev To Deployment", "description": "Learn Next.js by building a music event website and a web dev blog as a static website", "link": "https://www.traversymedia.com/next-js-dev-to-deployment", "level": "Intermediate"}]
```

Frontend 페이지 작성

- 1. /app/courses/page.tsx
- 2. /components/Courses.tsx
- 3. 헤더 메뉴에 등록

/app/courses/page.tsx

```
'use client'
import Courses from '@components/Courses'
import { useEffect, useState } from 'react'
import LoadingPage from '../loading'

export default function CoursesPage() {
  const [courses, setCourses] = useState([])
  const [loading, setLoading] = useState(true)

  useEffect(() => {
    const fetchCourses = async () => {
      const res = await fetch('/api/courses')
      const data = await res.json()
      setCourses(data)
      setLoading(false)
    }

    fetchCourses()
  }, [])

  if (loading) {
    return <LoadingPage />
  }

  return (
    <div>
      <h1 className="text-2xl font-bold mb-4">Courses </h1>
      <Courses courses={courses} />
    </div>
  )
}
```

Frontend 페이지 작성

- 페이지에서 보내준 courses 자료를 화면에 표시하는 기능

/components/Courses.tsx

```
import Link from 'next/link'

interface Course {
  id: string
  title: string
  description: string
  link: string
  level: string
}

interface CoursesProps {
  courses: Course[]
}

const Courses: React.FC<CoursesProps> = async ({ courses }) => {
  return (
    <div className="grid-1">
      {courses.map((course: Course) => (
        <div key={course.id} className="bg-blue-200 p-4 m-4 rounded-lg">
          <h2 className="text-lg">{course.title}</h2>
          <small>Level: {course.level}</small>
          <p className="mb-4">{course.description}</p>
          <Link
            href={course.link}
            target="_blank"
            className="py-2 px-4 bg-blue-700 hover:bg-blue-800 text-white rounded-lg mb-4"
          >
            Go To Course
          </Link>
        </div>
      ))}
    </div>
  )
}

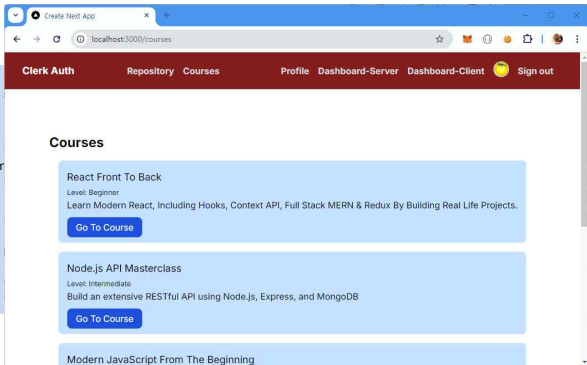
export default Courses
```


Frontend 페이지 작성

- 메뉴 추가

/components/Header.tsx

```
<div className="flex items-center font-bold">
  <Link href="/">
    <div className="text-lg text-white">Clerk Auth</div>
  </Link>
</div>
<div className="flex items-center font-bold">
  <Link href="/repos" className="text-gray-300 hover:text-white mr-4">
    Repository
  </Link>
  <Link
    href="/courses"
    className="text-gray-300 hover:text-white mr-4"
  >
    Courses
  </Link>
</div>
```



```
import Link from 'next/link'
```

```
/app/page.tsx
```

```
export default function Home() {
```

```
  return (
```

```
    <>
```

```
    <h1 className="text-2xl mb-4 font-bold">Welcome Home</h1>
```

```
    <div className="mb-4 bg-gray-100 p-4 m-4 rounded-lg">
```

```
      <Link
```

```
        href="https://clerk.com/"
```

```
        className="text-xl text-blue-700 font-bold"
```

```
      >
```

```
        clerk.com
```

```
    </Link>
```

```
    <p className="mb-2">
```

```
      This is a demo of Clerk authentication. Clerk is more than a "sign-in  
      box." Integrate complete user management UIs and APIs, purpose-built  
      for React, Next.js, and the Modern Web.
```

```
    </p>
```

```
  </div>
```

```
  <div className="mb-4 bg-gray-100 p-4 m-4 rounded-lg">
```

```
    <Link
```

```
      href="https://api.github.com/"
```

```
      className="text-xl text-blue-700 font-bold"
```

```
    >
```

```
      api.github.com
```

```
    </Link>
```

```
    <p className="mb-2">Explore github repositories using Fetch API</p>
```

```
  </div>
```

```
  <div className="mb-4 bg-gray-100 p-4 m-4 rounded-lg">
```

```
    <Link
```

```
      href="/api/courses"
```

```
      className="text-xl text-blue-700 font-bold"
```

```
    >
```

```
      Backend API
```

```
    </Link>
```

```
    <p className="mb-2">
```

```
      Provide Backend API service that provide extensive courses information  
      in web development
```

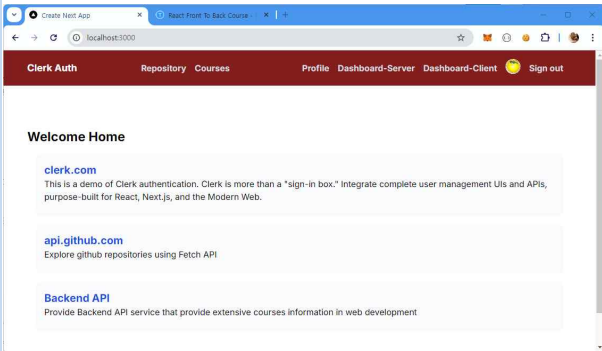
```
    </p>
```

```
  </div>
```

```
  )
```

```
}
```

루트페이지 내용 업데이트



6. Service Deploy

- 1. 빌드 테스트

- > npm run build (에러 발생 여부 확인)
- > npm run start (빌드된 정적 파일들로 서비스 시작, 서비스가 동일하게 동작하는지 확인)

Creating an optimized production build ...

- ✓ Compiled successfully
- ✓ Linting and checking validity of types
- ✓ Collecting page data
- ✓ Generating static pages (11/11)
- ✓ Collecting build traces
- ✓ Finalizing page optimization

Route (app)	Size	First Load JS
└─ f /	178 B	94.1 kB
└─ f /_not-found	871 B	88 kB
└─ f /api/courses	0 B	0 B
└─ ○ /api/hello	0 B	0 B
└─ f /client-dashboard	454 B	108 kB
└─ f /courses	1.33 kB	95.3 kB
└─ f /dashboard	141 B	87.3 kB
└─ f /profile/[...profile]	580 B	114 kB
└─ f /repos	178 B	94.1 kB
└─ f /repos/[name]	178 B	94.1 kB
└─ f /sign-in/[...sign-in]	580 B	114 kB
└─ f /sign-up/[...sign-up]	580 B	114 kB
+ First Load JS shared by all	87.1 kB	
└─ chunks/23-1ee1877d66a585a6.js		31.6 kB
└─ chunks/fd9d1056-bcd08f0f16acde53.js		53.6 kB
└─ other shared chunks (total)		1.95 kB

f Middleware 59 kB

○ (Static) prerendered as static content
f (Dynamic) server-rendered on demand

Service Deploy

- 2. Github.com에 repository 생성 및 프로젝트 업로드
 - <https://github.com/lbcsultan/clerk-api>

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner *	Repository name *
 lbcsultan	/ clerk-api
	 clerk-api is available.

Great repository names are short and memorable. Need inspiration? How about [miniature-train](#)?

Description (optional)

- ☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**
You choose who can see and commit to this repository.

Quick setup — if you've done this kind of thing before

 Set up in Desktop or  HTTPS  SSH <https://github.com/lbcsultan/clerk-api.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include

...or create a new repository on the command line

```
echo "# clerk-api" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/lbcsultan/clerk-api.git
git push -u origin main
```

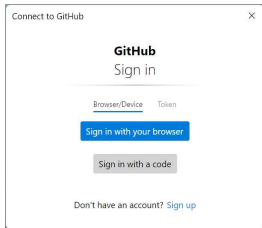
...or push an existing repository from the command line

```
git remote add origin https://github.com/lbcsultan/clerk-api.git
git branch -M main
git push -u origin main
```

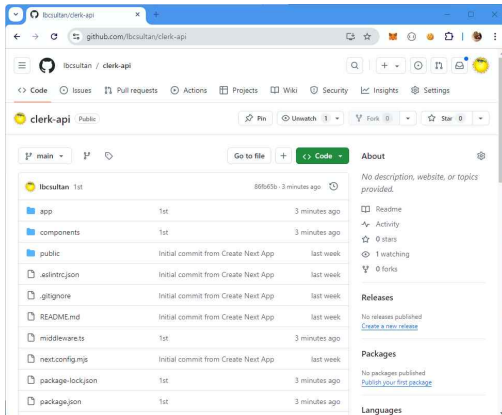
Service Deploy

- **...or create a new repository on the command line**
 - echo "# clerk-api" >> README.md
 - git init
 - git add README.md
 - git commit -m "first commit"
 - git branch -M main
 - git remote add origin https://github.com/lbcsultan/clerk-api.git
 - git push -u origin main
- * 처음 실행시 로그인 필요

Vscode에서 github 계정정보 설정
> git config --global user.email "이메일주소"
> git config --global user.name "계정이름"
한번 계정정보를 설정하고 로그인하면
윈도우의 자격 증명 관리자에
인증정보가 저장되며 인증이 유지됨
계정정보 설정을 다시 입력할 필요 없음



Github에 업로드 성공



Service Deploy

- 3. Vercel.com에 deploy
 - <https://vercel.com/> 에 회원가입 및 로그인 (github으로 회원가입 및 로그인 가능)

Create Your Vercel Account


Plan Type


Hobby
I'm working on personal projects


Pro
I'm working on commercial projects

Continue

Let's connect your Git provider

 Continue with GitHub

 Continue with GitLab

 Continue with Bitbucket

[Continue with Email →](#)


Service Deploy

- Add New... project
 - Import git repository
 - .env.local의 설정 정보를 Environment variable에 입력해야 함

Import Git Repository

🔍 ibcsultan

🔍 Search...

 clerk-api · 7m ago

Import

Environment Variables

Key	Value	
EXAMPLE_NAME	i9JU23NF394R6HH	Add

TIP: Paste a .env above to populate the form

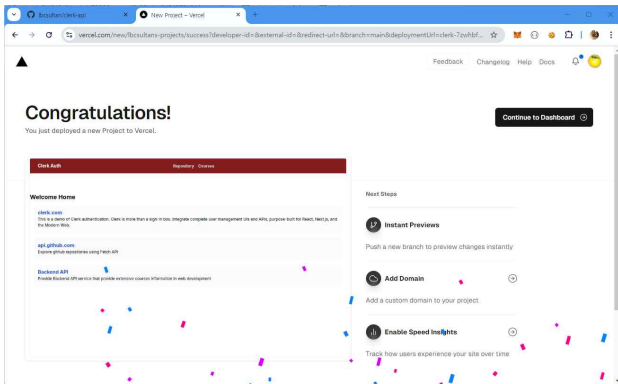
[Learn more about Environment Variables](#)

Key	Value	
NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY	pk_test_aW1tZW5zZS1qYXliaXJkLTUyLmNsZXJrLm	⊗
CLERK_SECRET_KEY	sk_test_cfucvclwpz9qMZFP1abF7ADsXftOr8zTi8d	⊗
NEXT_PUBLIC_CLERK_SIGN_IN_URL	/sign-in	⊗
NEXT_PUBLIC_CLERK_SIGN_UP_URL	/sign-up	⊗

Deploy

Vercel deploy

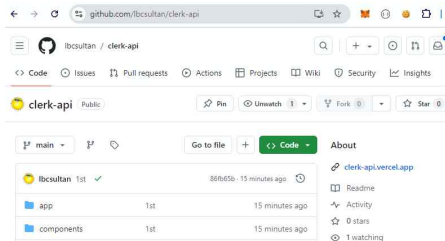
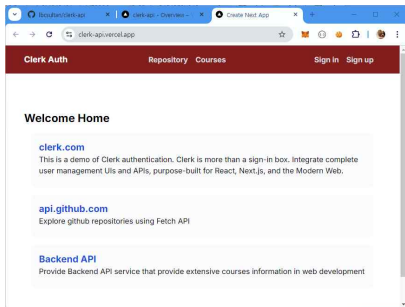
- 성공 화면



Domain 클릭하여 설정
Github repository에 도메인이 자동 등록됨

Vercel deploy

- Continue to dashboard 에서 간단한 이름의 도메인 클릭
- Deploy 완료
 - <https://clerk-api.vercel.app/>
- Github repository에 이 도메인 이름이 등록됨



Deploy 완료

- 로그인 테스트 성공

