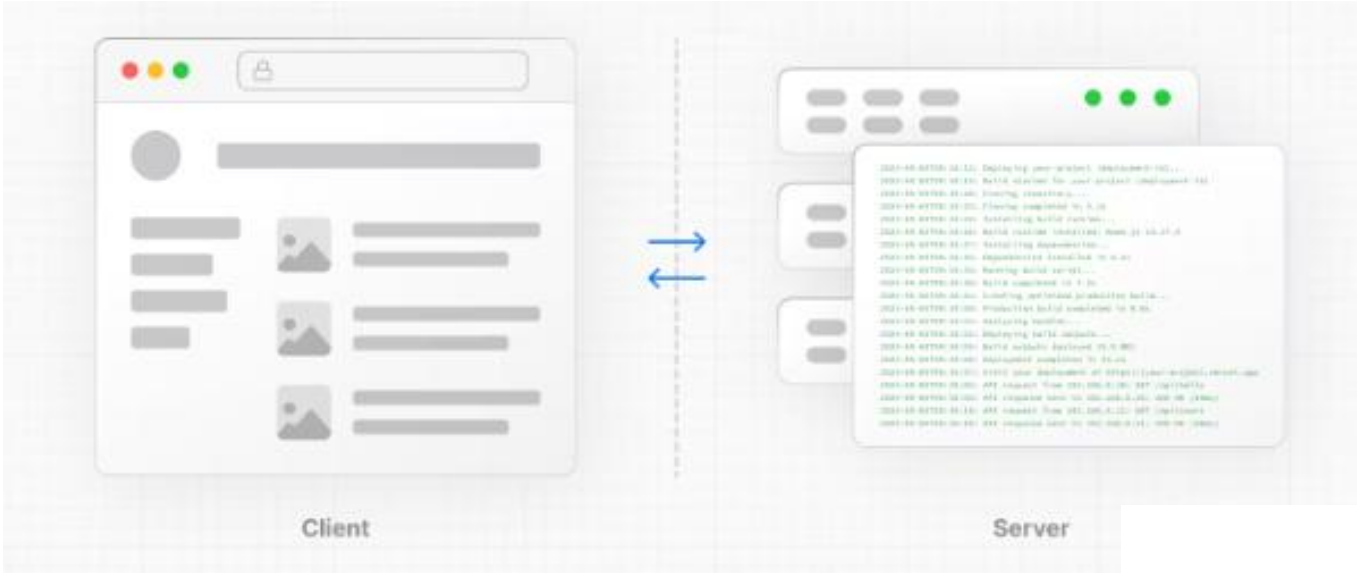
Next.js

김경민

Next.js

- Vercel이 개발한 React 프레임워크
  - React 기반 프레임워크로, 파일 기반 라우팅, 서버 사이드 렌더링(SSR), 정적 사이트 생성(SSG), API 라우트, 이미지 최적화, 자동 코드 분할 등 다양한 기능을 통해 React 앱을 더 쉽고 효율적으로 개발하고 배포할 수 있도록 도와 줌
- 목표
  - 프로덕션 수준의 웹 애플리케이션을 빠르고 효율적으로 개발할 수 있게 함
  - SEO(Search Engine Optimization) 및 퍼포먼스 최적화를 쉽게 처리할 수 있도록 설계됨
  - 프론트엔드와 백엔드를 하나의 프로젝트에서 함께 개발 가능

렌더링 방식(Rendering Strategy)



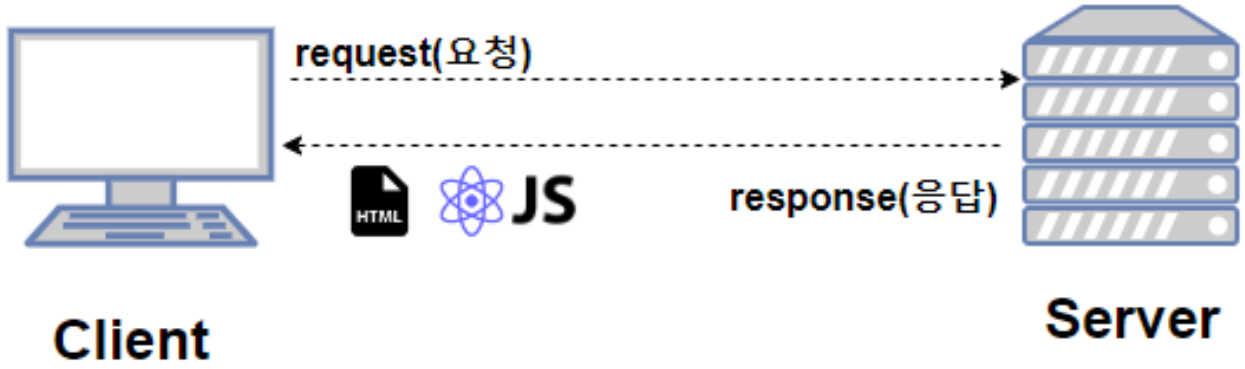
CSR
(**C**lient **S**ide **R**andering)

SSG
(**S**tatic **S**ite **G**eneration)

ISR
(**I**ncreamental **S**taic **R**egeneration)

SSR
(**S**erver **S**ide **R**endering)

CSR (Client Side Randering)



장점

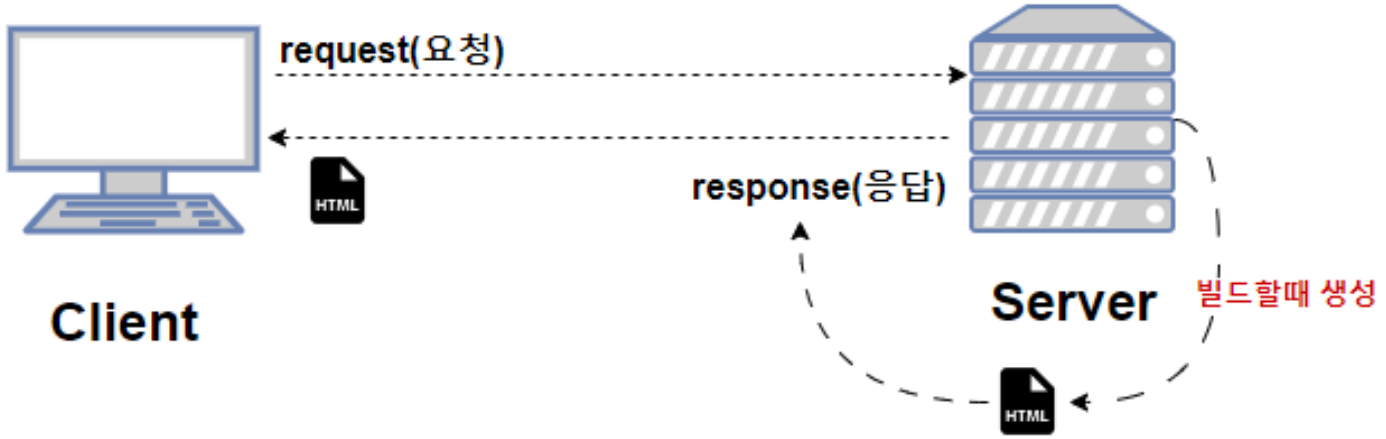
- 한번 로딩되면 빠른 UX 제공
- 서버의 부하가 작음

단점

- 페이지 로딩시간(TimeToView)이 길다
- 자바스크립트 활성화 필수 임
- SEO 최적화가 힘듬
- 보안에 취약함
- CDN에 캐시가 안됨

React JS

SSG (Static Site Generation)



장점

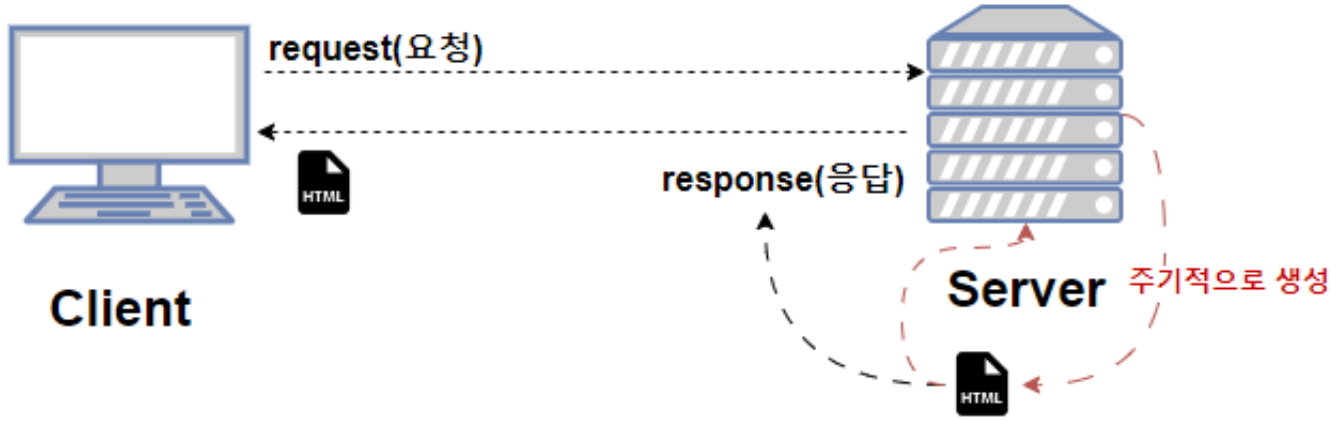
- 페이지 로딩시간(TimeToView)이 빠름
- 자바스크립트 필요 없음
- SEO 최적화가 좋음
- 보안이 뛰어남
- CDN 캐시가 됨

단점

- 데이터가 정적임
- 사용자별 정보 제공이 어려움

React JS

ISR (Increamental Staic Regeneration)



장점

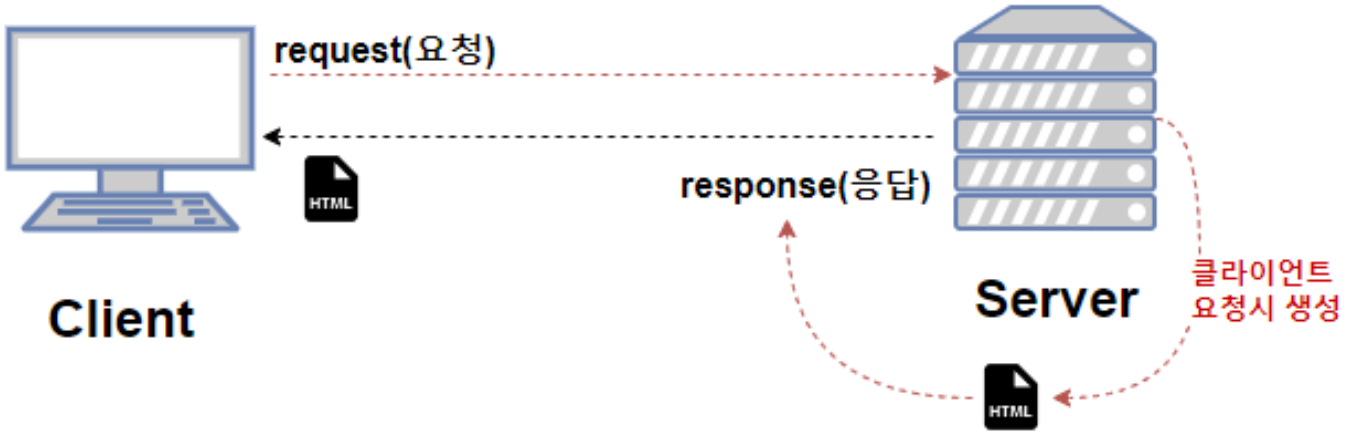
- 페이지 로딩시간(TimeToView)이 빠름
- 자바스크립트 필요없음
- SEO 최적화가 좋음
- 보안이 뛰어남
- CDN 캐시가 됨
- 데이터가 주기적으로 업데이트

단점

- 데이터가 정적임
- 사용자별 정보 제공이 어려움

React JS

SSR (Server Side Rendering)



장점
- 페이지 로딩시간(TimeToView)이 빠름
- 자바스크립트 필요없음
- SEO 최적화가 좋음
- 보안이 뛰어남
- 실시간 데이터를 사용
- 사용자별 필요한 데이터 사용

단점
- 서버의 과부화가 생겨 비교적 느릴 수 있음
- CDN에 캐시가 안됨

React JS

Next.js Hybrid Web App

혼합 특정 목적을 달성하기 위해
CSR, SSG, ISR, SSR 중 두개 이상의 기능이나 요소를 결합

Hydration
서버에서 받은 정적 HTML에 React
컴포넌트를 동적으로 연결하는 과정

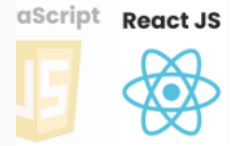
각 페이지마다 최적 렌더링 방식 선택 가능 →
성능 + SEO + 유연성 확보

Next.js 프로젝트 생성

```
D:\frontend2025> npx create-next-app@latest mynext
√ 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
√ What import alias would you like co
Creating a new Next.js app in D:\fron
```

src/ 디렉토리를 나타내는 경로 별칭(alias)

```
Using npm.

Initializing project with template: app-tw


Installing dependencies:
- react
- react-dom
- next

Installing devDependencies:
- typescript
- @types/node
- @types/react
- @types/react-dom
- @tailwindcss/postcss
- tailwindcss
- eslint
- eslint-config-next
- @eslint/eslintrc


added 316 packages, and audited 317 packages in 43s

130 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
Initialized a git repository.

Success! Created mynext at D:\frontend2025\mynext
```

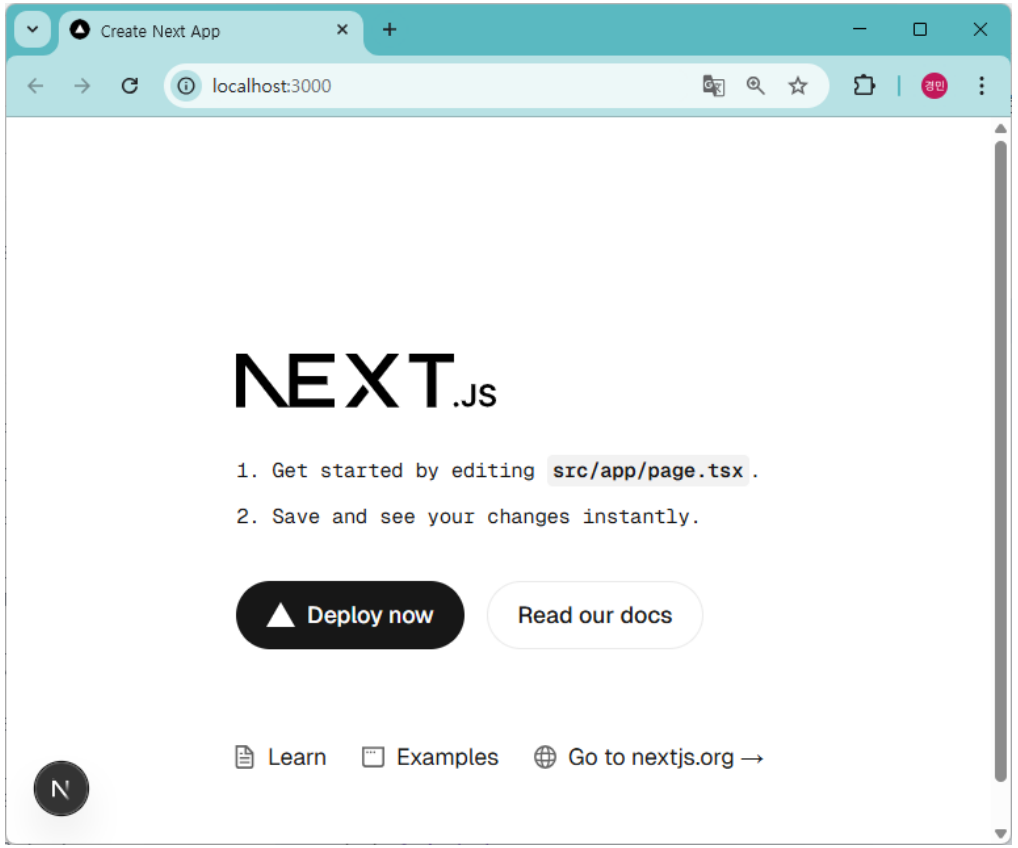


라이브러리 추가

- 리액트 아이콘 설치
  - npm install react-icons
- jotai 설치
  - npm install jotai
- axios 설치
  - npm install axios
- 환경변수
  - 파일명 : .env.local
  - Prefix : NEXT_PUBLIC_
  - 사용시 : process.env.NEXT_PUBLIC_
- Json 서버 실행
  - npx json-server --watch db.json --port 3005

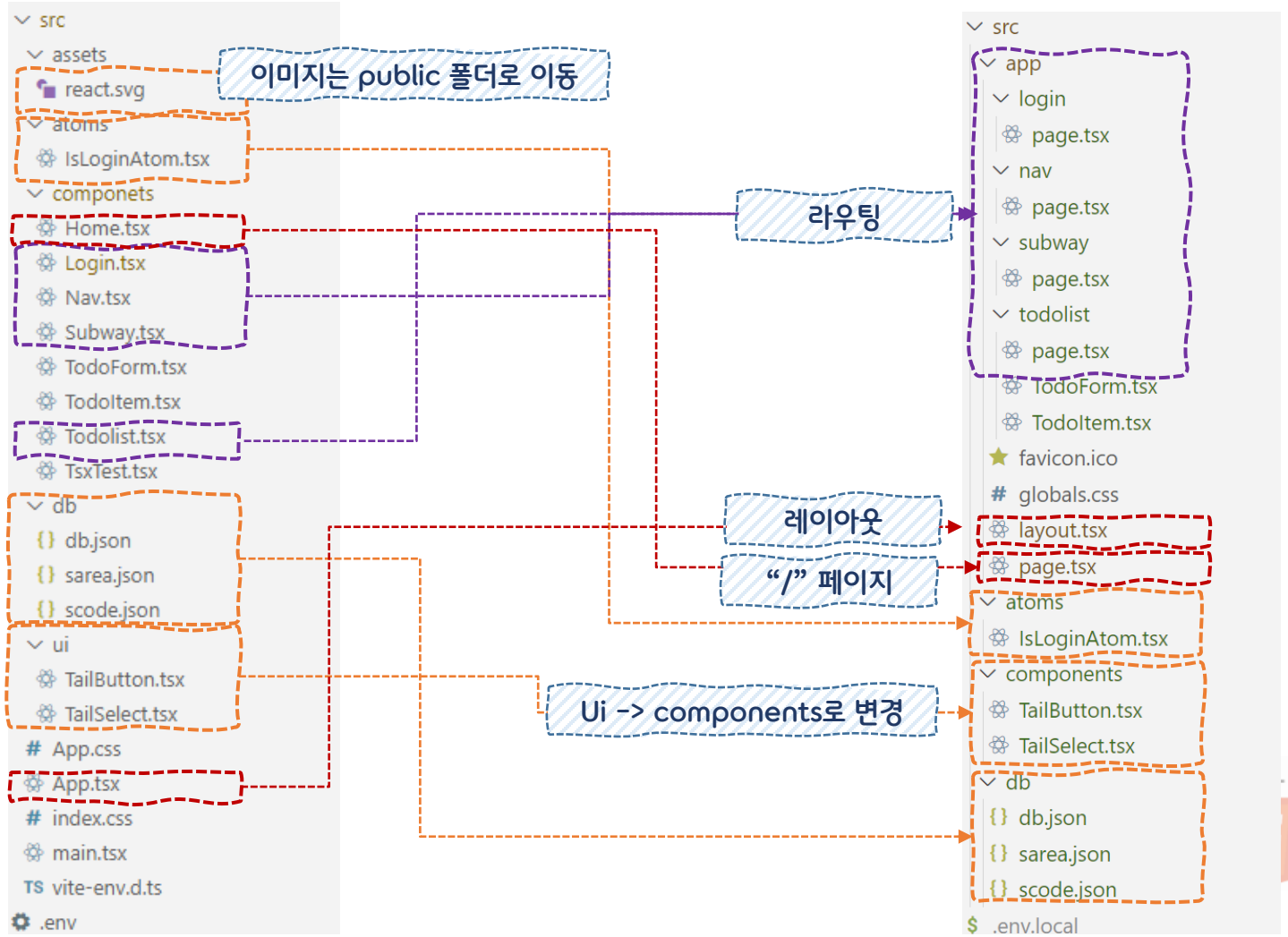
Next.js 프로젝트 실행

```
○ PS D:\frontend2025\mynext> npm run dev

  > mynext@0.1.0 dev
  > next dev --turbopack

     ▲ Next.js 15.3.0 (Turbopack)
     - Local:        http://localhost:3000
     - Network:      http://172.30.1.75:3000

   ✓ Starting...
   ✓ Ready in 10.1s
   ○ Compiling / ...
   ✓ Compiled / in 6s
   GET / 200 in 8600ms
   ✓ Compiled /favicon.ico in 386ms
   GET /favicon.ico?favicon.45db1c09.ico 200 in 1288ms
```



폴더 구조

React 프로젝트 폴더



Next.js 프로젝트 폴더

RootLayout 컴포넌트

```tsx
import type { Metadata } from "next";
import { Geist, Geist_Mono } from "next/font/google";
import "./globals.css";
import Nav from "@/app/nav/page";
```

- Next.js에서 SEO 및 문서 메타 정보를 정의할 수 있는 타입
- Google Fonts API를 통한 폰트 로딩 (Next.js 내장)
- CSS 파일 import (Tailwind 등 포함 가능)
- 상단 네비게이션 컴포넌트

```tsx
const geistSans = Geist({
  variable: "--font-geist-sans",
  subsets: ["latin"],
});

const geistMono = Geist_Mono({
  variable: "--font-geist-mono",
  subsets: ["latin"],
});
```

font 설정
- Next.js의 next/font API를 사용해 Google Fonts를 최적화된 방식으로 불러옴
- 각각 CSS variable로 폰트 변수를 할당

```tsx
export const metadata: Metadata = {
  title: "Login Test",
  description: "Generated by create next app",
};
```

metadata 설정
- metadata 객체는 Next.js에서 <head> 영역에 자동 삽입
- title, description 등 SEO에 사용되며, 페이지 타이틀에 반영

RootLayout 컴포넌트

```tsx
export default function RootLayout({children,}: Readonly<{children: React.ReactNode;}>)
  return (
    <html lang="ko">
      <body
        className={`${geistSans.variable} ${geistMono.variable} antialiased`}
      >
      <div className="w-full xl:w-4/5 h-screen mx-auto
                      flex flex-col justify-start items-center">

          <Nav />
          <main className="w-full flex-grow flex flex-col items-center
                           overflow-y-auto">
            {children}
          </main>

          <footer className="w-full h-24 min-h-24 flex justify-center items-center
                             ■bg-gray-950 □text-white text-lg">
            [부산대학교 K-Digital] AI 데이터 분석 풀스택 웹 개발자 양성과정
          </footer>
        </div>
      </body>
    </html>
  );
}
```

상단 고정 내비게이션

라우팅되는 페이지 콘텐츠가 들어갈 자리

고정 푸터 내용

Route의 엔트리 포인트 : page.tsx

```tsx
'use client'

import { useAtom } from "jotai" ;
import { isLogin } from "@/atoms/IsLoginAtom";
import Login from "@/app/login/page";
import { useEffect, useState } from "react";

export default function Home() {
  const [login, setLogin] = useAtom(isLogin) ;

  /**...
  const [email, setEmail] = useState<string | null>(null) ;
  useEffect(() =>{
    const locEmail = localStorage.getItem('email') ;
    if (locEmail) {
      setEmail(locEmail) ;
      setLogin(true);
    }
  }, []);

  return (
    <div className="w-full h-full">
      {login ? <div className="flex min-h-full
                               text-2xl font-bold items-center
                               justify-center px-6 py-12 lg:px-8">
                <span className="text-blue-700">{email}</span>
                님이 로그인 되었습니다.
                </div>
               : <Login />}
    </div>
  )
}
```

- localStorage는 브라우저에서만 존재하는 객체이므로, SSR 단계에서는 접근할 수 없음으로 useEffect를 통해 클라이언트 사이드에서만 실행되도록 처리

상단 고정 내비게이션 : app>nav>page.tsx

```tsx
'use client'
import Image from 'next/image';

// import { Link } from 'react-router-dom' ;
import Link from 'next/link';

import { useAtom } from 'jotai' ;
import { isLogin } from "@/atoms/IsLoginAtom" ;
// import { useNavigate } from 'react-router-dom';
import { useRouter } from 'next/navigation'

export default function Nav() {
  const [login, setLogin] = useAtom(isLogin) ;
  // const navigate = useNavigate();
  const router = useRouter();

  const handleLogout = () => {
    localStorage.setItem("email", '') ;
    setLogin(false);
    // navigate("/") ;
    router.push("/") ;
  }
```

- Next.js App Router에서는 기본적으로 서버 컴포넌트(Server Component)로 동작함으로 use client를 선언하면 해당 파일을 클라이언트 컴포넌트(Client Component)로 강제 지정
- useState, useEffect, jotai 같은 React 클라이언트 훅을 사용하기 위한 필수 선언

<Image> 컴포넌트

- next/image에서 제공되며, 성능과 사용자 경험을 모두 고려한 최적화된 이미지 렌더링 방식
- width와 height 필수
  이미지 비율 및 레이아웃 계산에 필요함 (레이아웃 shift 방지)

- Next.js에서 페이지 간 이동을 위한 컴포넌트
- 기본 HTML <a> 태그를 대체하면서도, 페이지 전체 새로고침 없이 부드럽게 이동
- <Link href="/">홈</Link>

- Next.js의 App Router에서 제공하는 라우터 훅
- 클라이언트 컴포넌트에서만 사용할 수 있으며, 페이지 이동, 쿼리 조작 등 다양한 내비게이션 기능을 제공
- Next.js 13 이상에서 app/ 디렉토리 사용 시 사용
- 현재 페이지에서 지정한 경로("/")로 이동하는 클라이언트 사이드 라우팅을 수행
- 전체 페이지를 새로고침 하지 않고 React 방식으로 라우팅(SPA 특성 유지)

/login : app>login >page.tsx

```tsx
'use client'

import { useRef } from "react" ;
// import { useNavigate } from "react-router-dom";
import { useRouter } from "next/navigation";
import { useAtom } from "jotai";
import { isLogin } from "@/atoms/IsLoginAtom";

export default function Login() {
  const emailRef = useRef<HTMLInputElement>(null) ;
  const pwdRef = useRef<HTMLInputElement>(null) ;

  const [, setLogin] = useAtom(isLogin) ;
  // const navigate = useNavigate() ;
  const router = useRouter() ;

  const handleOk = () => {

        ...

  localStorage.setItem("email", emailRef.current.value) ;
  setLogin(true) ;
  // navigate("/") ;
  router.push("/");
}
```

- Next.js의 App Router에서 제공하는 라우터 훅
- 클라이언트 컴포넌트에서만 사용할 수 있으며, 페이지 이동, 쿼리 조작 등 다양한 내비게이션 기능을 제공
- Next.js 13 이상에서 app/ 디렉토리 사용 시 사용

- 현재 페이지에서 지정한 경로("/")로 이동하는 클라이언트 사이드 라우팅을 수행
- 전체 페이지를 새로고침 하지 않고 React 방식으로 라우팅(SPA 특성 유지)

/subway : app>subway>page.tsx

```tsx
'use client'

// import sarea from "../db/sarea.json" ;
import sarea from "@/db/sarea.json" ;

// import scode from "../db/scode.json" ;
import scode from "@/db/scode.json" ;

//import TailSelect from "../ui/TailSelect";
import TailSelect from "@/components/TailSelect";

const getFetchData = async (idx : string): Promise<void>  => {
  // const apikey = import.meta.env.VITE_APP_API_KEY ;
  const apikey = process.env.NEXT_PUBLIC_API_KEY;
  let url = `https://apis.data.go.kr/6260000/IndoorAirQuality/getIndoorAirQualityByStation?`;
  url = `${url}serviceKey=${apikey}&pageNo=1&numOfRows=5&resultType=json`;
  url = `${url}&controlnumber=${new Date().toISOString().slice( 0, 10 ).replaceAll('-','') }06`;
  url = `${url}&areaIndex=${idx}` ;

  console.log(url)
  const resp = await fetch(url) ;
  const data = await resp.json() ;

  setTdata(data.response.body.items.item[0])
}
```

- @를 사용
  src/ 디렉토리를 나타내는 경로 별칭(alias)

- 환경변수 값을 불러오기

SSR 방식 랜더링 : app>todolist>page.tsx

```tsx
import TodoItem from "./TodoItem";
import TodoForm from "./TodoForm";

const baseUrl = "http://localhost:3005/todos"

// SSR 방식으로 todo 불러오기
async function getTodos() {
  const res = await fetch(baseUrl, {
    // SSR을 위해 캐시 없이
    cache: "no-store",
  });
  if (!res.ok) throw new Error("Failed to fetch todos");
  return res.json();
}

export default async function Todolist() {
  const todos = await getTodos();

  return (
    <>
      <div className="w-full h-40 min-h-40 bg-amber-50 mt-5
                      flex flex-col justify-center items-center">
        <h1 className='w-full text-2xl font-bold mb-5 text-center'>Todo List</h1>
        <TodoForm />
      </div>
      <ul className="w-10/12 flex flex-col p-5">
        {
          todos.map((item:any) => <TodoItem key={item.id}
                                    todo = {item}
                                  /> )
        }
      </ul>
    </>
  )
}
```

- 'use client'와 클라이언트 훅인 useEffect, useState를 삭제
- json-server가 제공하는 REST API 주소에서 데이터 가져오기
- SSR에서 매 요청마다 최신 데이터를 가져오도록 설정
- 데이터가 정상적으로 오지 않으면 에러를 throw
- 입력 함수는 TodoForm 컴포넌트에서 구현
- 수정과 삭제 함수는 TodoItem 컴포넌트에서 구현

TodoFrom.jsx

```
'use client'
import axios from "axios";
import TailButton from "@/components/TailButton";
import { useRef } from "react";

const baseUrl = "http://localhost:3005/todos"
export default function TodoForm() {
  const refInput = useRef<HTMLInputElement>(null);
  const refSel = useRef<HTMLSelectElement>(null);

  const addTodo = async (text : string, completed : "O" | "X")  => {
    await axios.post(baseUrl, {
      text : text ,
      completed : completed
    })

  }

  const handleClick = (e : React.MouseEvent<HTMLButtonElement>) => {
    e.preventDefault();
    if (refInput.current?.value == '') {
      alert('할일을 입력하세요.');
      refInput.current.focus();
    }

    if (refInput.current && refSel.current) {
      addTodo(refInput.current?.value, refSel.current?.value as "O" | "X");
    }

    window.location.reload(); // SSR이므로 새로고침 필요
  }
```

- Props로 전달되는 내용이 없음
- 입력 함수 구현
- 현재 페이지를 새로고침(refresh) 하는 명령어
- 서버에서 데이터가 변경된 후 다시 반영하고 싶을 때

TodoItem

```tsx
'use client'
import axios from "axios";
import TailButton from "@/components/TailButton"

interface Todo {
  id : string ;
  text : string ;
  completed : "O" | "X" ;
}

interface TodoItemProps {
  todo : Todo;
}
const baseUrl = "http://localhost:3005/todos"
export default function TodoItem({ todo } : TodoItemProps) {
  const handleDelete = async(id:string) => {
    await axios.delete(baseUrl+`/${id}`) ;
    window.location.reload(); // SSR이므로 새로고침 필요
  }

  const handleToggle = async(id:string) => {
    const resp = await axios.get<Todo>(baseUrl+`/${id}`) ;
    const todo = resp.data ;

    const done = todo.completed == "O" ? "X" : "O";
    await axios.patch(baseUrl+`/${id}`, {
      completed : done
    }) ;
    window.location.reload(); // SSR이므로 새로고침 필요
  }
```

- Props로 전달되는 수정, 삭제함수 없음
- 삭제 함수 구현
- 수정함수 구현