리액트 스터디 6강

캡스 39기 송윤석

강의 전체 로드맵

- 1강: 스터디 OT, 개발환경 구축, HTML/CSS ✅
- 2강: Javascript, Typescript ✅
- 3강: React의 세계관, Electron 개념, 프로젝트 생성 ✅
- 4강: 컴포넌트, 상태관리 ✅
- 5강: 훅, 전역상태관리 ✅
- 6강: 라우팅, 라이브러리, 외부 API, 배포 <- 오늘!!!!!!

목차 및 오늘의 목표

- 6.1 React Router 개요 및 설치
- 6.2 Outlet, Link, useNavigate 활용
- 6.3 동적 라우팅, 쿼리 파라미터
- 6.4 axios로 외부 API 연동
- 6.5 마크다운(GitHub raw) API 호출 및 파싱
- 6.6 마크다운 렌더링 라이브러리 소개 (ex: react-markdown)
- 6.7 Vercel을 통한 간단 배포

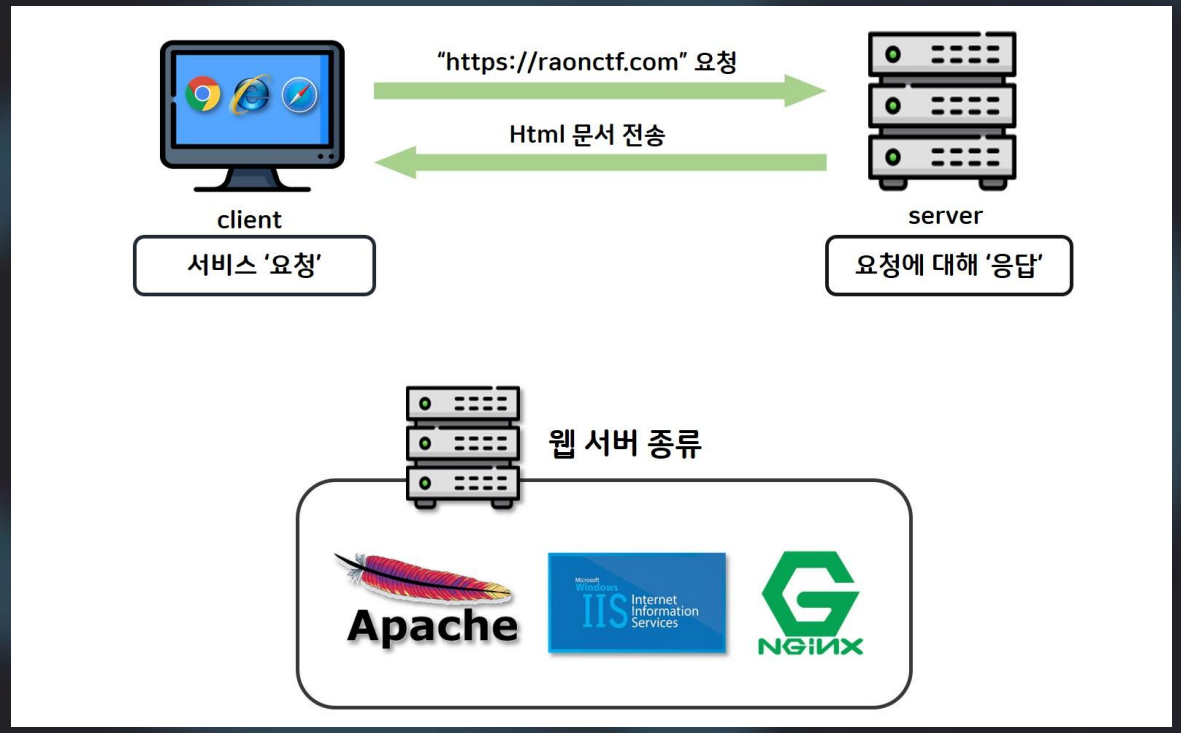
6.1 React Router 개요 및 설치

- 라우팅이 뭘까?
  - 네트워크에서 경로를 선택하는 과정

- 웹 개발에서는 이 개념을 웹 페이지의 경로를 정하는 데 사용
- 사용자가 웹사이트의 주소창(URL)에 특정 주소를 입력했을 때,
- 그 주소에 맞는 페이지(화면)를 보여주는 과정

6.1 React Router 개요 및 설치

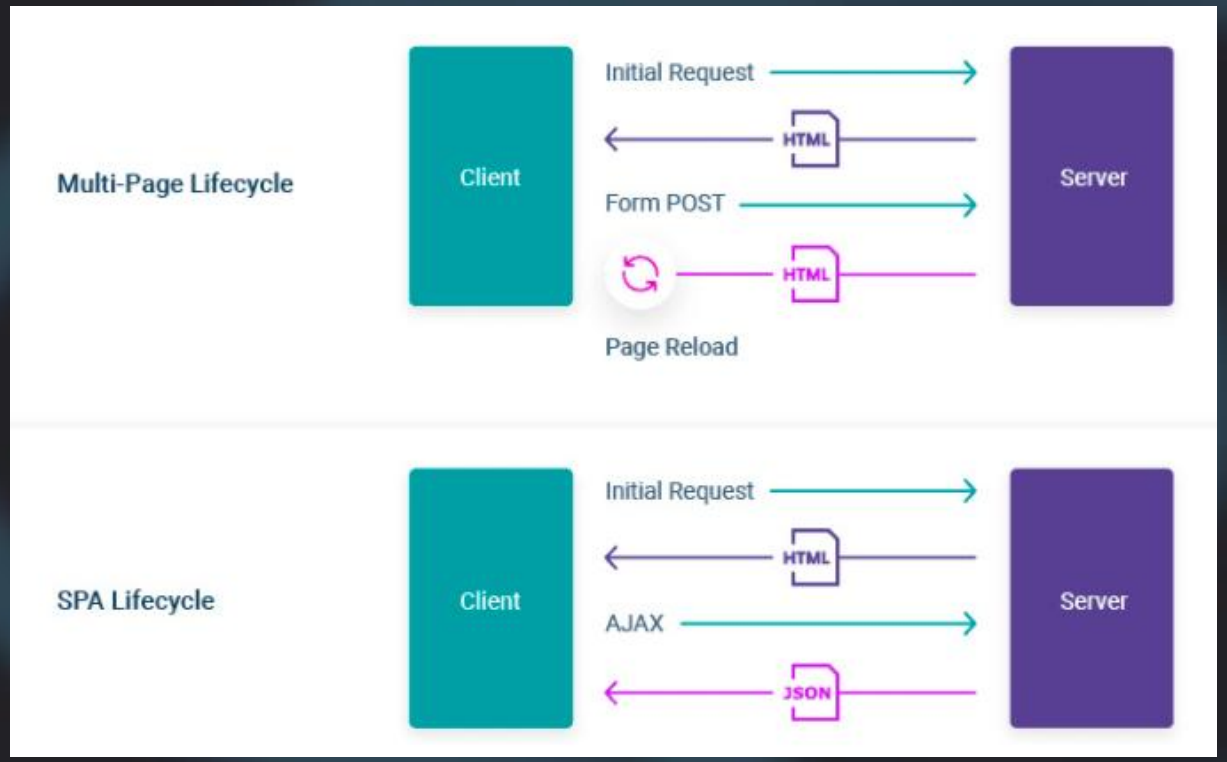
- 사용자가 다른 페이지로 이동할 때마다 서버에 새로운 페이지(HTML)를 요청
- 서버는 요청에 맞는 새로운 HTML 파일을 보내줌
- 이 과정에서 페이지 전체가 새로고침되기 때문에, 화면이 깜빡이거나 느리게 느껴질 수 있음



6.1 React Router 개요 및 설치

- 리액트는 SPA(Single Page App)이다!
- 즉, 하나의 HTML문서 안에서, JS를 통해 화면을 동적으로 변경하는 방식

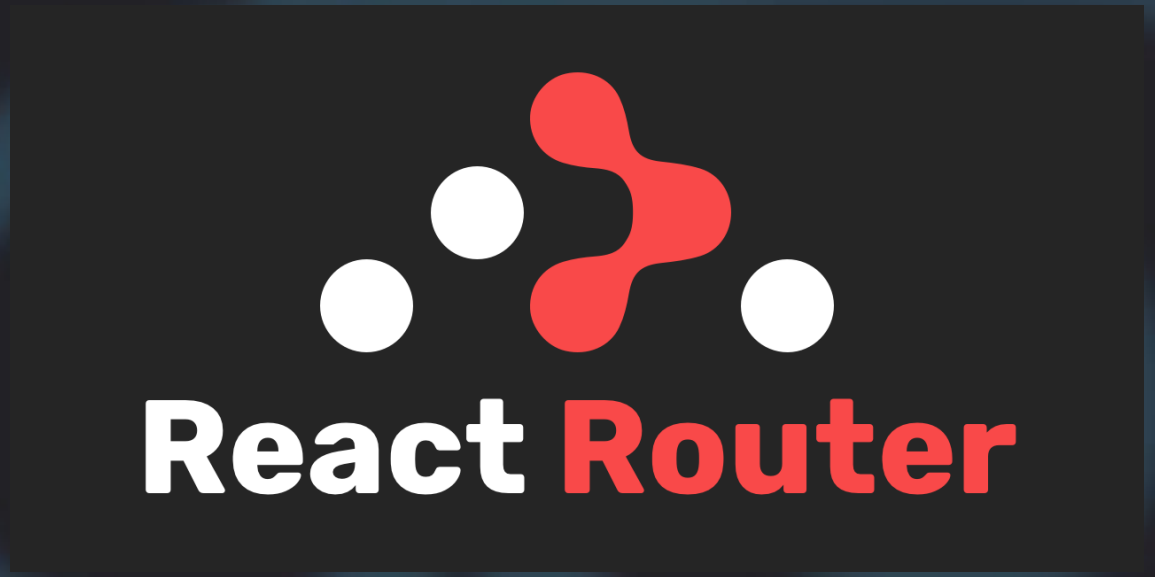
- 그렇기에 /post /admin 등이 새 HTML을 불러오지 않음!
  애초에 경로가 없음



6.1 React Router 개요 및 설치

- https://reactrouter.com/home
- Declarative: 선언적 모드
- Data: 데이터 흐름 모드
- Framework: 구 Remix, 서버사이드렌더링 등 기능이 많은 모드

- 이번에 사용할 건 Data 모드



6.1 React Router 개요 및 설치

- 새 프로젝트 생성 후
- npm i react-router

- routes/Router.tsx
- pages/Root.tsx
- pages/Post.tsx

```tsx
import {
  createBrowserRouter,
  RouterProvider
} from "react-router";

import RootPage from "./pages/Root";
import PostPage from "./pages/Post";

function App() {
  const router = createBrowserRouter([
    { path: "/", Component: RootPage },
    { path: "/posts", Component: PostPage}
  ])
  return <RouterProvider router={router} />
}

export default App
```

6.2 Outlet, Link, useNavigate 활용

```jsx
import { Link, useNavigate } from "react-router"

export default function Header() {
  const navigate = useNavigate();
  return (
    <header>
      <h2>리액트 블로그</h2>
      <Link to="/">홈</Link>
      <button onClick={() => navigate("/posts")}>
        글 목록
      </button>
    </header>
  )
}
```

- Link 컴포넌트와 to props를 통해 새로고침 없는 페이지 이동이 가능함
- 컴포넌트를 통한 선언적!

- 프로그래밍적으로 이동시키고 싶으면 useNavigate를 사용
- 주로 복잡한 로직 후 페이지 전환을 할 때

6.2 Outlet, Link, useNavigate 활용

- Outlet을 활용하여 렌더링 요소를 내부에 집어넣을 수 있음
- 이를 이용해 공통 레이아웃 구성이 가능함

```jsx
import { Outlet } from "react-router";
import Header from "../components/Header";

export default function RootLayout() {
  return (
    <div>
      <Header />
      <Outlet />
    </div>
  )
}
```

```jsx
import { createBrowserRouter, RouterProvider } from "react-router";
import RootPage from "./pages/Root";
import PostPage from "./pages/Post";
import RootLayout from "./layouts/RootLayout";

function App() {
  const router = createBrowserRouter([
    {
      path: "/",
      element: <RootLayout />,
      children: [
        { path: "/", element: <RootPage />},
        { path: "/posts", element: <PostPage />}
      ]
    },
  ])
  return <RouterProvider router={router} />
}

export default App
```

6.3 동적 라우팅, 쿼리 파라미터

- 동적 라우팅
- /posts/{post_id} 같이 라우팅이 미리 정해져있지 않은 경우
- 리액트 라우터에선 :postId로 표시

- useParams로 라우팅 파라미터를 가져올 수 있음

```
  const router = createBrowserRouter([
    {
      path: "/",
      element: <RootLayout />,
      children: [
        { path: "/", element: <RootPage /> },
        { path: "/posts", element: <PostPage /> },
        { path: "/posts/:postId", element: <SinglePostPage /> }
      ]
    },
  ])
  return <RouterProvider router={router} />
}
```

6.3 동적 라우팅, 쿼리 파라미터

- 동적 라우팅
- /posts/{post_id} 같이 라우팅이 미리 정해져있지 않은 경우
- 리액트 라우터에선 :postId로 표시

- useParams로 라우팅 파라미터를 가져올 수 있음

```
import { useParams } from 'react-router';

export default function SinglePostPage( ) {
  const { postId } = useParams();

  return (
    <div>
      <p>포스트: {postId}</p>
    </div>
  )
}
```

6.3 동적 라우팅, 쿼리 파라미터

- **쿼리 파라미터**
- /hello?to=world&i=happy
- **Url 뒤에 물음표를 붙여서 key-value 형태로 데이터를 전달하는 방식**
- **여러 데이터 전달시 &으로 구분**

- 주로 웹사이트에서 **데이터를 필터링, 정렬하거나 페이지네이션을 구현**할 때 사용
- **리다이렉트 URL 저장**과 같은 기능에도 유용하게 사용

```jsx
import { useSearchParams } from "react-router"

export default function PostPage() {
  const [searchParams, setSearchParams] = useSearchParams();
  const author = searchParams.get('author');

  const handleClick = () => {
    setSearchParams({ author: 'newuser' });
  };

  return (
    <div>
      <p>현재 작성자: {author || '모두'}</p>
      <button onClick={handleClick}>작성자 변경</button>
    </div>
  );
}
```

6.4 axios로 외부 API 연동

- **Axios**는 브라우저와 Node.js 환경에서 **HTTP 통신을 쉽게 할 수 있도록 도와주는** JavaScript 라이브러리

- **fetch 보다 직관적**
- **응답 자동 JSON 변환**
- **인터셉터** - 요청을 보내기 전이나 응답을 받은 후, 데이터를 가로채서 처리하는 기능
- **인스턴스** - 여러 요청에 공통으로 적용될 기본 설정을 만들어 재사용

6.4 axios로 외부 API 연동

- axios.create()로 공통 인스턴스 만들기
- 긴 요청 URL을 매번 작성 안해도 됨
- 에러시 재시도, 로그인 정보 첨부 등의 부가 기능도 추가 가능

```
import axios from 'axios';

const api = axios.create({
  baseURL: 'https://jsonplaceholder.typicode.com',
  headers: {
    'Content-Type': 'application/json',
  },
  timeout: 5000,
});

export default api;
```

6.4 axios로 외부 API 연동

- Axios를 사용하면 HTTP 요청을 보낼 때 .메서드() 형태로 아주 직관적으로 작성

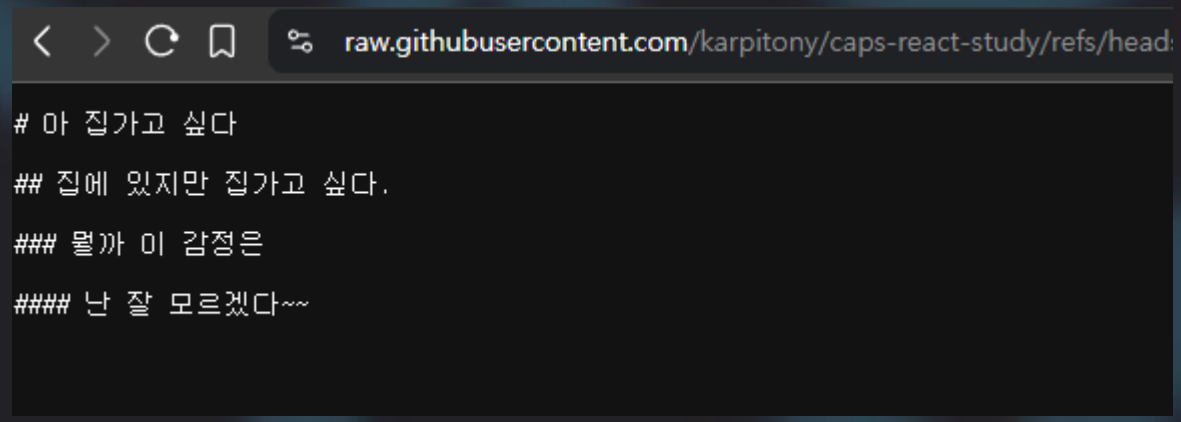
```
const response =
  await api.get<PostData>(`/posts/${postId}`);
```

- .get(): 서버로부터 데이터를 가져올 때 (READ).
- .post(): 서버에 새로운 데이터를 생성할 때 (CREATE).
- .put(): 서버의 기존 데이터를 수정할 때 (UPDATE, 전체 수정).
- .delete(): 서버의 데이터를 삭제할 때 (DELETE).

6.5 마크다운(GitHub raw) API 호출

- 깃허브에 블로그 글까지 업로드
- Github raw로 서버처럼 사용

- https://raw.githubusercontent.com/karpitony/caps-react-study/refs/heads/main/react-markdown-blog/src/posts/250819.md



6.5 마크다운(GitHub raw) API 호출

- 깃허브에 블로그 글까지 업로드
- Github raw로 서버처럼 사용

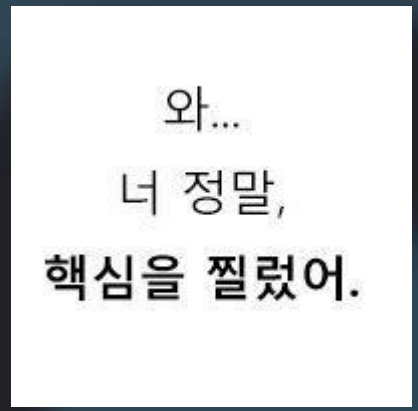
```
import axios from 'axios';

const api = axios.create({
  baseURL: 'https://raw.githubusercontent.com/karpitony/caps-react-study/refs/heads/main/react-markdown-blog/src/posts',
  headers: {
    'Content-Type': 'application/json',
  },
  timeout: 5000,
});

export default api;
```

6.6 마크다운 렌더링 라이브러리 소개

- 만약 마크다운이 뭔지, 어떻게 쓰는건지 모른다면...
- 이미 알고 있을수도! Velog, notion도 마크다운 문법 일부 사용
- chatGPT의 '너 정말 **핵심을 찔렀어**'도 마크다운



- 깃허브의 README 등 개발자 문서에 자주 사용되는 매우매우 쉬운 문법

- # 을 통해 제목을 지정(큰글씨 됨)
- **강조**로 볼드 처리 *기울임* ***강조 기울임***
- - 또는 * 하나로 항목 나열
- `(백틱)을 통해 코드 하이라이팅도 가능!
- > 으로 인용 표시

https://blog.coinsect.io/markdown-editor

6.6 마크다운 렌더링 라이브러리 소개

- npm i react-markdown
- npm i github-markdown-css

- https://github.com/karpitony/9oormthonUniv-React-Study/blob/78be7a3edc9391d72a44198ff51b4506e1841151/src/pages/Article.tsx#L55
- 위처럼 하나하나 조정도 가능함

```jsx
import ReactMarkdown from 'react-markdown';
import 'github-markdown-css/github-markdown-light.css';

export default function SinglePostPage( ) {
  return (
    <div>
      <p>포스트: {postId}</p>
      <div className='markdown-body'>
        <ReactMarkdown>{data}</ReactMarkdown>
      </div>
    </div>
  )
}
```

6.6 문법 하이라이트 추가하기

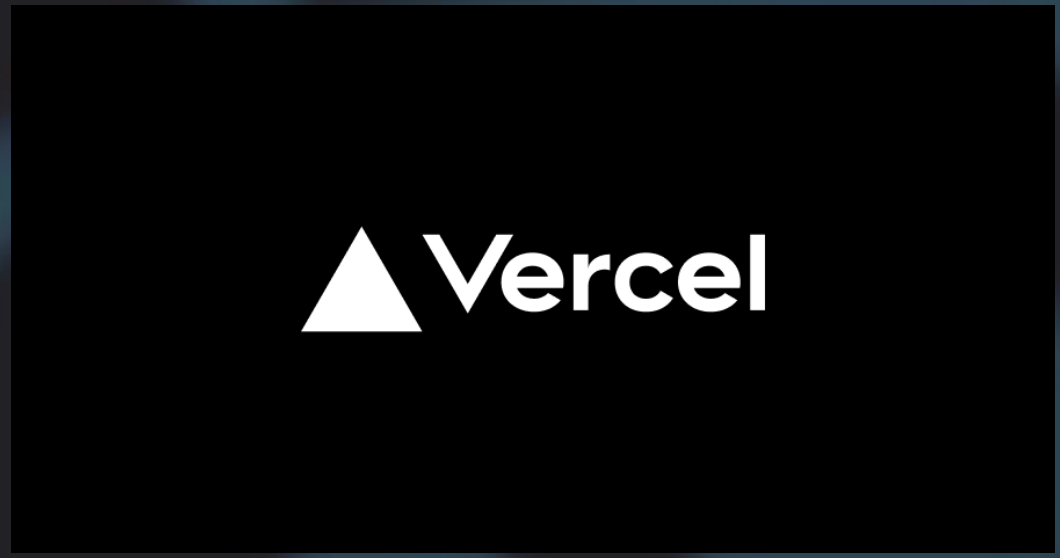
- npm i react-syntax-highlighter
- npm i @types/react-syntax-highlighter

- https://github.com/karpitony/caps-react-study/blob/main/react-markdown-blog/src/pages/SinglePost.tsx
- (코드가 너무 많아 깃허브 링크로 대체)

6.7 Vercel을 통한 간단 배포

- Vercel: 프론트를 위한 무료 AWS
- 일정 사용량이 넘어가면 그때부터 돈을 달라고 한 뒤 서비스
- Github와 연동되어 매우 쉽게 프론트엔드를 배포할 수 있음

- https://vercel.com



6.7 Vercel을 통한 간단 배포

- 리액트는 SPA이기 때문에 버셀과 라우팅을 맞춰줘야 함

- 버셀은 /posts 등이 오면 그 파일을 찾으려 하지만, 우리는 /index.html 하나밖에 없기 때문

```
{
    "rewrites": [
        {
            "source": "/(.*)",
            "destination": "/index.html"
        }
    ]
}
```

6.7 Vercel을 통한 간단 배포

- **버셀 분석기 붙이기**
- npm i @vercel/analytics

```
import { Outlet } from "react-router";
import Header from "../components/Header";
import { Analytics } from '@vercel/analytics/react';

export default function RootLayout() {
  return (
    <div>
      <Header />
      <Outlet />
      <Analytics />
    </div>
  )
}
```

프론트(React) 스터디는 여기까지

- 오늘 배운 것
  - react-router로 리액트에서 라우팅 구현하기
  - Axios로 외부 API 쉽게 연동
  - React-router + Axios 실습 – 마크다운 블로그 만들기

- 실습 코드
  - 깃허브에 업로드 했으니 직접 결과 확인을 원하면 편하게 사용하세요
  - https://github.com/karpitony/caps-react-study

프론트(React) 스터디는 여기까지

- 앞으로 해보면 좋은것들
  - 만든 마크다운 블로그는 부실합니다! CSS 꾸미기, 추가적인 기능 넣기!
  - 배운 내용들로 개발 블로그를 채워봅시다!
  - 자기소개 페이지도 한번 넣어보세요!

  - 요새는 리액트의 프레임워크인 Next.js도 많이 사용합니다.
  - 리액트 프로젝트에 익숙해지면 이 친구도 배워보세요! 혼자서 백엔드 일부 기능을 쉽게 만들 수 있습니다.
  - https://nomadcoders.co/nextjs-for-beginners

- 6주동안 수고 많으셨습니다!