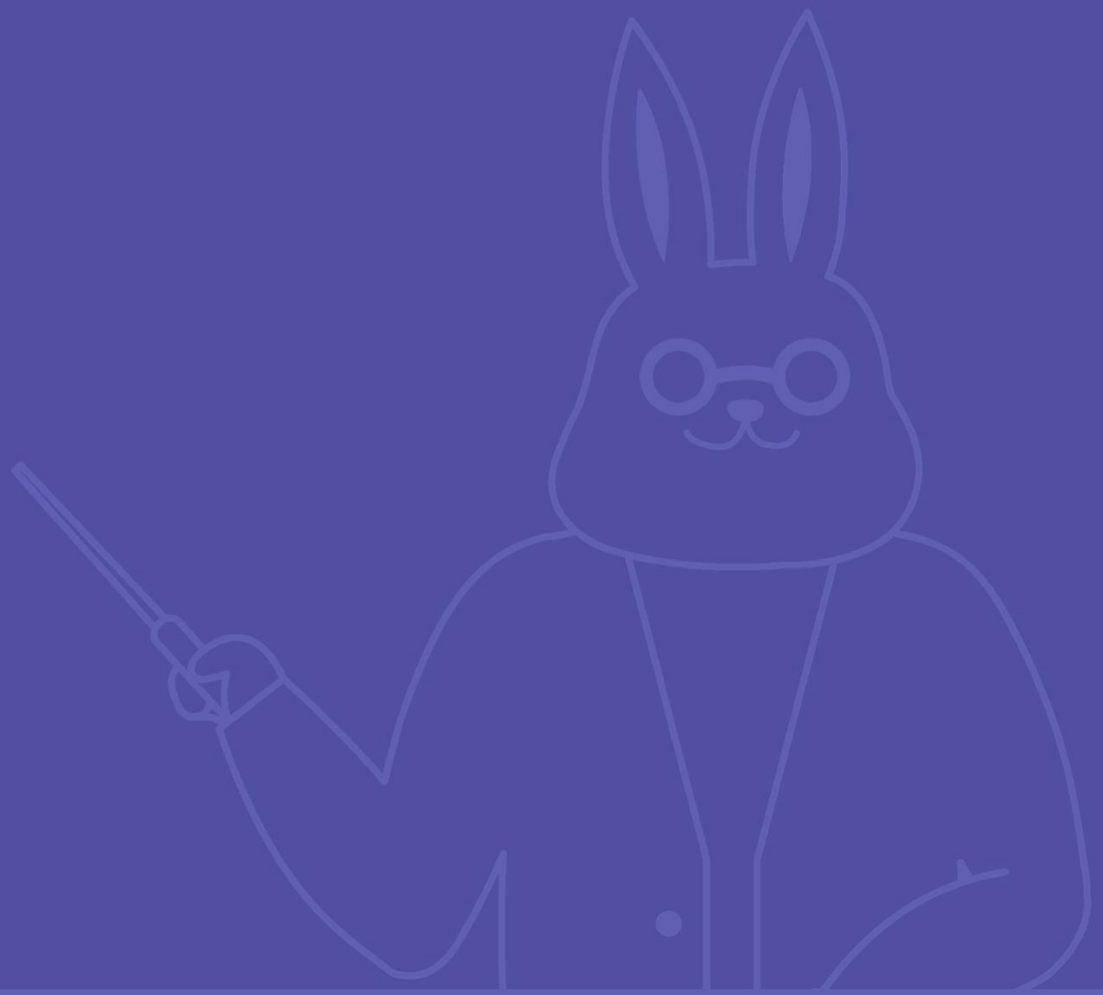
React 심화 I

02 SPA와 라우팅

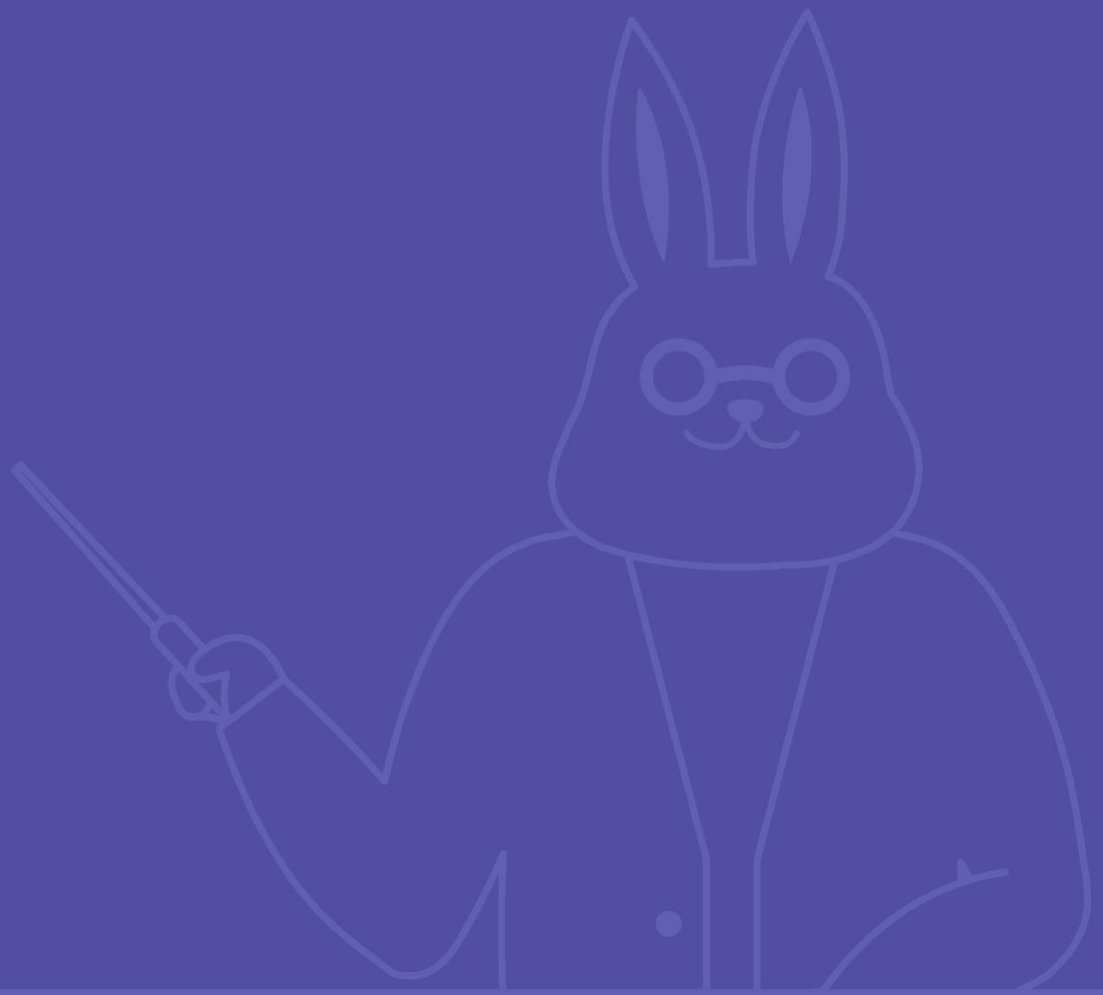


/* elice */

목차

01

SPA와 라우팅



Single Page Application

- SPA(Single Page Application)은 하나의 페이지 요청으로 전체 웹앱을 사용하는 방식.
- 유저는 웹페이지를 사용하며 모바일 앱 같은 경험을 느낌.

Multi Page Application

- MPA(Multi Page Application)은 서버에 미리 여러 페이지를 두고,
  유저가 네비게이션 시 요청에 적합한 페이지를 전달.
- 미리 서버에서 전체 페이지를 빌드해 브라우저로 전송됨.
- 서버에 라우팅을 처리하는 기능이 있고, 서버에서 여러 페이지를 관리함.
- 페이지 요청마다 모든 리소스를 다시 받아오므로, 페이지 간 데이터를 재활용하기 힘듦.

SPA의 특징

- Client-side routing 기술을 활용, 페이지 진입 시 리로드없이 라우팅함.
- AJAX 기술을 활용, 페이지 이동 시 서버에 데이터만 요청하여 자바스크립트로 페이지를 만듦.
- MPA와 다르게, 여러 페이지를 하나의 앱의 구성요소로 보고 여러 페이지 간의 스타일, 컴포넌트를 재활용하는 방향으로 구현.
- 자바스크립트만을 활용해 전체 페이지를 만들기에, 첫 요청 시 빈 페이지를 받게 됨.

SPA의 기술적 장점

- 서버에서 페이지를 만들 필요가 없으므로 CDN에 캐싱이 가능.
- 매번 페이지 요청을 할 필요가 없어 네트워크 요청이 줄어듦.
  마찬가지로 데이터 요청 등을 캐싱하여 재사용하는 등 제약 조건이 줄어듦.
- 웹사이트를 개별 페이지보다는 하나의 앱으로 보는 설계로 고도의 소프트웨어 설계와 패턴을 적용할 수 있음.

SPA의 기술적 난관들

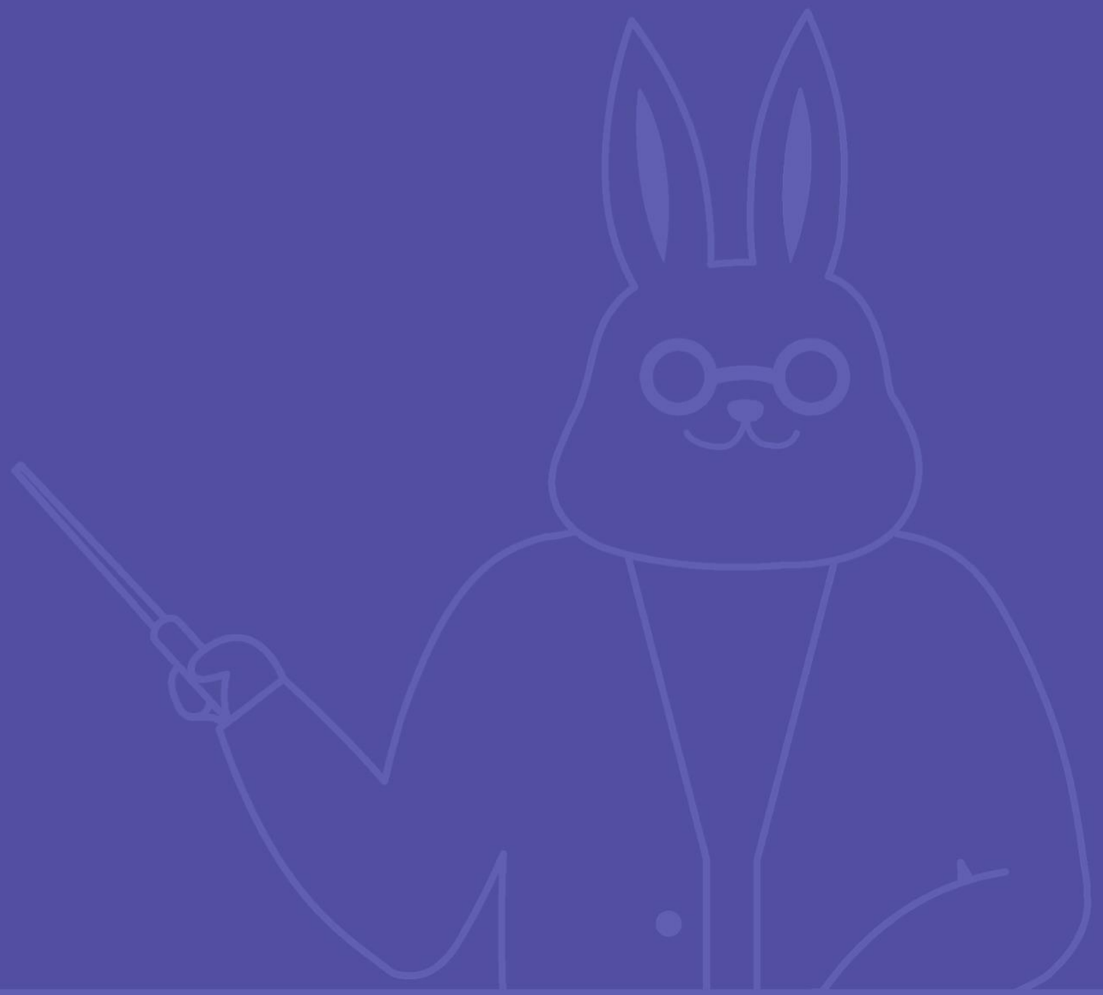
- MPA방식 보다는 Search Engine Optimization에 불리함.
- 하나의 자바스크립트 앱이 지속하므로, 메모리 관리와 성능, 데이터 활용 등이 중요.
- 여러 페이지를 전송받는 것 보다, 하나의 거대한 자바스크립트 앱을 전송받아야 하므로 코드가 많아질수록 로드 속도가 느려짐.

SPA에서의 라우팅

- 주로 History API 혹은 URL Hash를 이용해 페이지 리로드 없는 페이지 전환을 구현.
- history, location 등 HTML5 API를 활용.
- visibilitychange, popstate, beforeunload 등 window event를 활용하여 페이지 전환 등의 이벤트 시 핸들러를 등록.
- react-router, reach-router 등의 라이브러리를 활용하면, 라우팅 관련 기능을 쉽게 사용할 수 있음.

02

react-router 소개



react-router

- Declarative routing for React.
- React의 JSX를 이용하거나, History API를 사용하여 라우팅을 구현.
- 웹에서는 react-router-dom을 사용.
- 적용 시, 서버의 모든 path에서 같은 앱을 서빙하도록 해야 함.

react-router의 기능

- React 컴포넌트를 특정 path와 연결하면, 해당하는 path로 진입 시 컴포넌트를 렌더링하게 함.
- query, path variable 등 URL parameter를 얻어 활용함.
- 조건에 맞지 않을 경우 redirect 함.
- 페이지 이동 시, 이벤트 핸들러를 등록함.
- /posts/my-post-1 등의 nested route를 구현함.

react-router의 사용

- <BrowserRouter> 로 감싸 Router Context를 제공해야 함.
- Route로 path를 정의하고, 그 안에 렌더링하고자 하는 컴포넌트를 넣음.
- Link로 특정 페이지로 이동 시, 리로드 없이 페이지가 이동함.
- Switch로, 매칭되는 라우트 하나를 렌더링하게 함.

react-router의 사용

App.jsx

```
import { BrowserRouter, Route, Switch } from 'react-router-dom'

export function App() {
  return (
    <BrowserRouter>
      <Switch>
        <Route path="/about"><AboutPage /></Route>
        <Route path="/contact"><ContactPage /></Route>
        <Route path="/"><HomePage /></Route>
      </Switch>
    </BrowserRouter>
  )
}
```

react-router의 사용

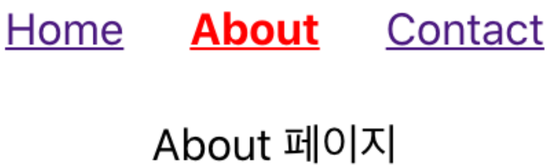
HomePage.jsx

```
import { NavLink } from 'react-router-dom'

function HomePage() {
  return (
    <div>
      <nav>
        <NavLink to="/">Home</NavLink>
        <NavLink to="/about">About</NavLink>
        <NavLink to="/contact">Contact</NavLink>
      </nav>
      <div>Home 페이지</div>
    </div>
  )
}
```

AboutPage.jsx

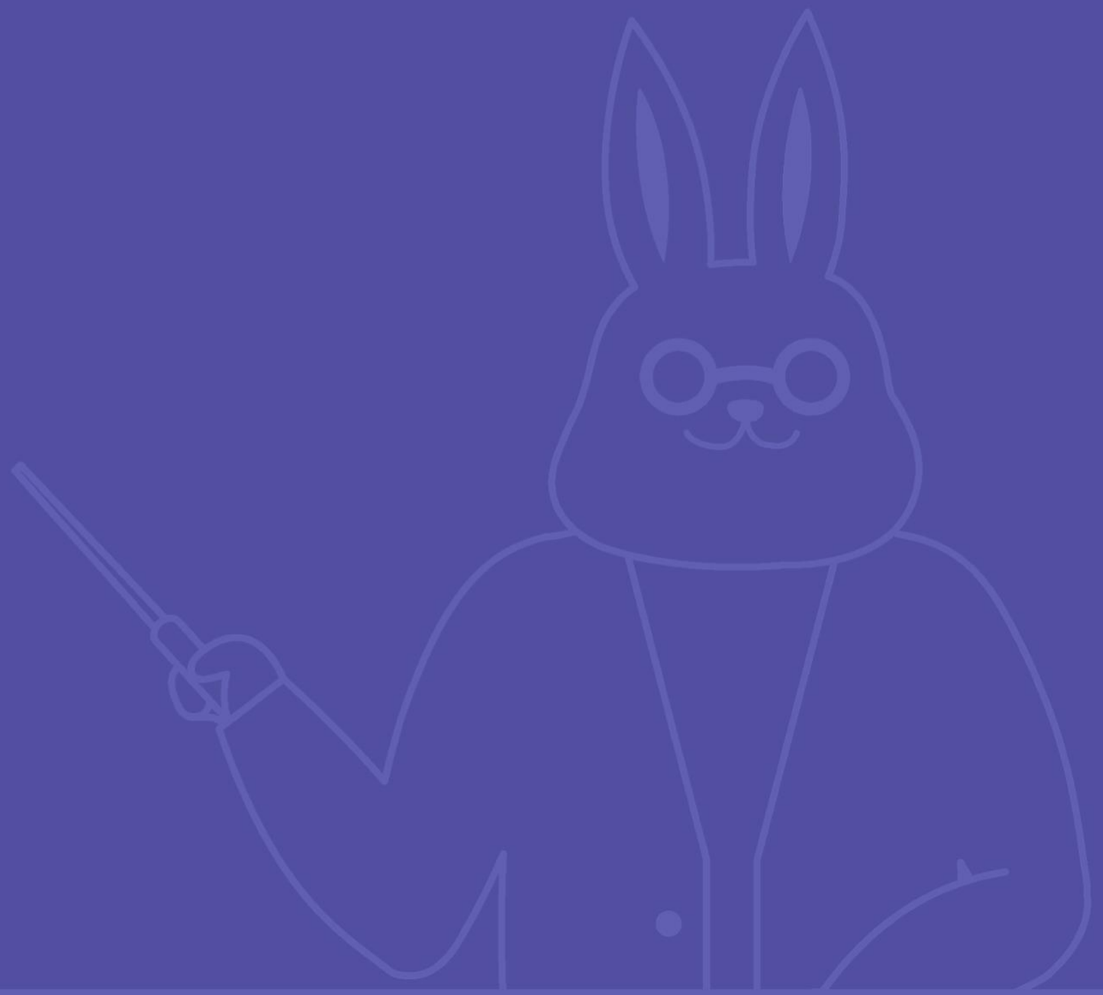
```
import { NavLink } from 'react-router-dom'

function AboutPage() {
  return (
    <div>
      <nav>
        <NavLink to="/">Home</NavLink>
        <NavLink to="/about">About</NavLink>
        <NavLink to="/contact">Contact</NavLink>
      </nav>
      <div>About 페이지</div>
    </div>
  )
}
```

react-router의 사용



03

react-router 컴포넌트



BrowserRouter

- HTML5의 History API를 사용하여, UI와 URL의 싱크를 맞추는 역할.
- 모든 URL에 대해 동작하게 하기 위해서는 서버 설정 필요.
- 모든 path 앞의 basename을 지정할 수 있음.
  ex) basename="/ko"
- forceRefresh로, 페이지 이동 시 리프레시할 것인지 지정할 수 있음.

Switch

- 여러 Route 중 매치되는 Route 위에서부터 하나 선택하여 렌더링함.
- 매칭되는 Route가 없으면 아무것도 보여주지 않음.
  fallback용으로 404 Not Found Page를 추가함.
- path="/"의 경우 모든 path에 매칭되므로 exact 키워드를 추가하거나
  가장 아래로 내림.

Route

- path와 컴포넌트를 매칭함.
- 매칭되는 컴포넌트는 children으로 넣어주거나
  component prop으로 넘김.
- exact 키워드로 정확하게 매칭하는 path를 설정함.
- Route로 렌더링 되는 최상위 컴포넌트는
  match, location, history를 prop으로 받음.
- render prop으로, 매칭되었을 때 실제 어떤 컴포넌트를 렌더링할지 통제함.

Redirect

- Link와 비슷하나, 렌더링되면 to prop으로 지정한 path로 이동함.
- Switch 안에서 쓰일 경우, from, to를 받아 이동하게 만듦.
  ex) from="/" to="/login"

Link, NavLink

- to prop을 특정 URL로 받아, 클릭 시 네비게이션 함.
- anchor tag를 래핑함.
- NavLink의 경우, 매칭 시 어떤 스타일을 가질지 등의 추가 기능이 있음.
- to에 location object나 함수를 받을 수 있음.

useHistory, useLocation, useParams, useRouteMatch

- 최상위 컴포넌트가 아니더라도, hook으로 react-router 관련 객체에 접근할 수 있음.
- history, location, params, match 객체에 접근함.

04

react-router로 페이지 구성하기

/* elice */

공통 페이지 레이아웃

PageLayout.jsx

```jsx
export default function PageLayout({ header,
children }) {
  return (
    <Layout>
      <Navigation />

      <header>
        <h2>{header}</h2>
      </header>

      <main>{children}</main>
    </Layout>
  );
}
```

PageLayout.jsx

```jsx
const Layout = styled.div`
  height: 100vh;

  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
`;
```

공통 페이지 네비게이션

Navigation.jsx

```
function Navigation() {
  return (
    <Nav>
      <NavLink to="/">Home</NavLink>
      <NavLink to="/about">About</NavLink>
      <NavLink
to="/contact">Contact</NavLink>
    </Nav>
  );
}
```

Navigation.jsx

```
const Nav = styled.div`
  padding: 24px;

  & > a:not(:first-of-type) {
    margin-left: 24px;
  }
`;
```

개별 페이지 컴포넌트

HomePage.jsx

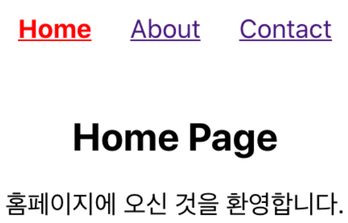
```
function HomePage() {
  return (
    <PageLayout header="Home Page">홈페이지에 오신 것을 환영합니다.</PageLayout>
  );
}
```

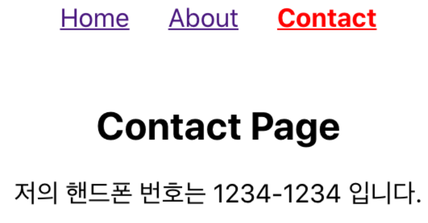
라우터 연결

App.jsx

```
function App() {
  return (
    <BrowserRouter>
      <Switch>
        <Route exact path="/">
          <HomePage />
        </Route>
        <Route path="/about">
          <AboutPage />
        </Route>
        <Route path="/contact">
          <ContactPage />
        </Route>
      </Switch>
    </BrowserRouter>
  );
}
```

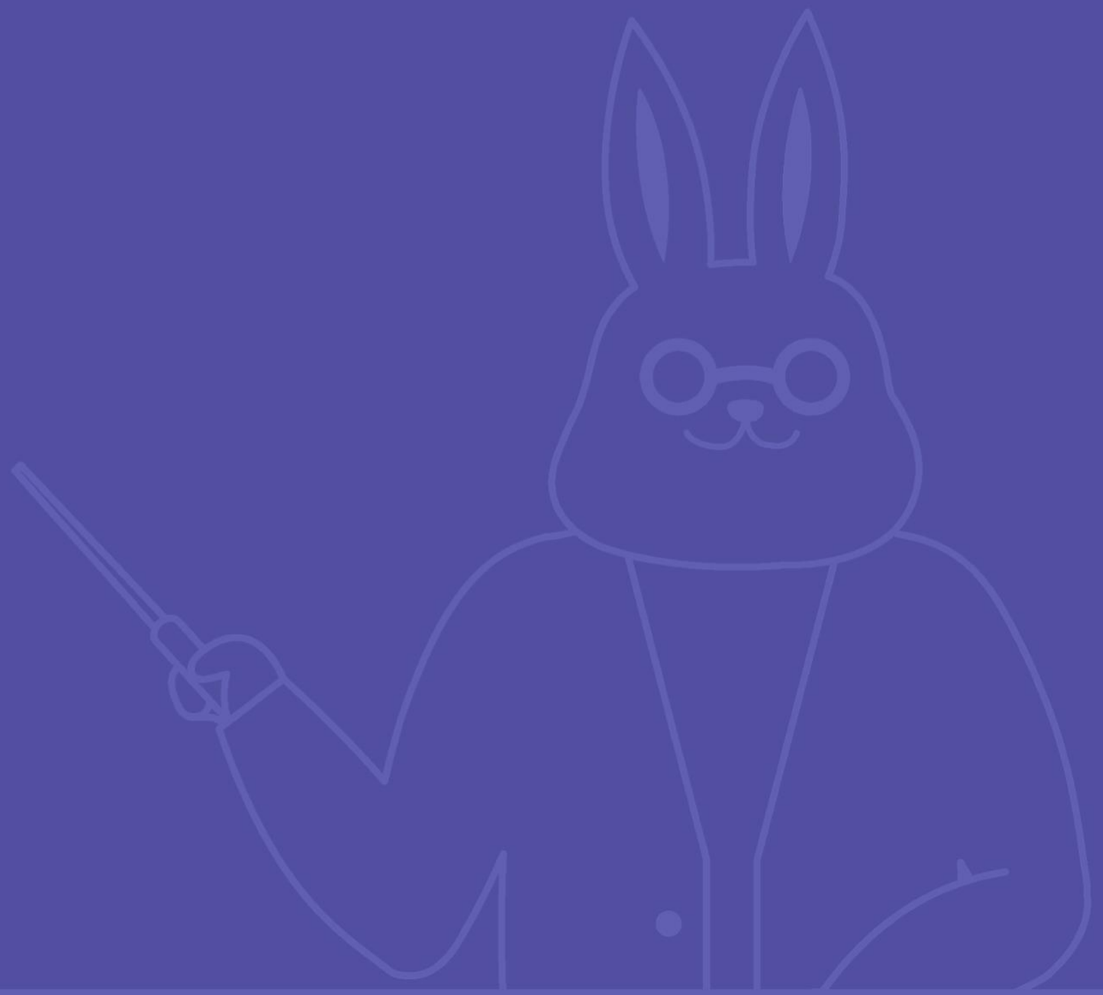
결과 페이지 모습


Home About Contact
Home Page
홈페이지에 오신 것을 환영합니다.


Home About Contact
Contact Page
저의 핸드폰 번호는 1234-1234 입니다.

05

react-router 응용



Private Route 만들기

- 특정 조건이 충족되지 않았을 때 다른 페이지로 Redirect 하도록 하는 기능.
- 유저의 상세 페이지, 개인정보 변경 페이지 등을 만들 때 사용됨.

Private Route 만들기 - declarative

Code

```
function PrivateRoute({ component: Component, ...props }) {
  return <Route {...props} render={props => {
    const isLoggedIn = !!getUserInfo()
    if (!isLoggedIn) {
      return <Redirect to="/login" />
    }
    return <Component {...props} />
  }}
}
```

Private Route 만들기 - imperative

Code

```
function usePrivateRoute(validateFunc) {
  const history = useHistory()

  useEffect(() => {
    if (!validateFunc()) {
      history.push("/login")
    }
  }, [])
}
```

query string 활용하기

- URL의 query string 정보를 활용해 앱을 구성할 수 있음.
- URLSearchParams API를 활용함.

query string 활용하기

Code

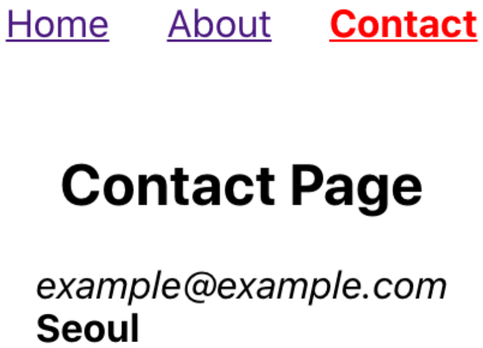
```
function ContactPage() {
  const location = useLocation();
  const searchParams = new URLSearchParams(location.search);

  const email = searchParams.get("email");
  const address = searchParams.get("address");

  return (
    <PageLayout header="Contact Page">
      <em>{email}</em>
      <br />
      <strong>{address}</strong>
    </PageLayout>
  );
}
```

query string 활용하기

Code

```
function App() {

  // ...

  return (
    <Link to="/contact?email=example@example.com&address=Seoul">
      Contact
    </Link>
  );
}
```

query string 활용하기



크레딧

/* elice */

코스 매니저
이재성

콘텐츠 제작자
김일식

강사
김일식

감수자
-

디자이너
강혜정

연락처

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io