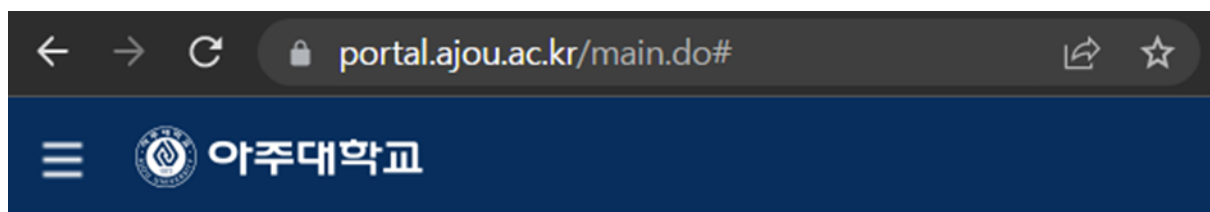


# REACT Router 사용법

아주대학교 사이버보안학과 202020689 임현아

## 1. React Router v6 소개

보통 웹사이트에서는 주소를 입력하거나 링크를 클릭하면 새로운 페이지를 보여준다.



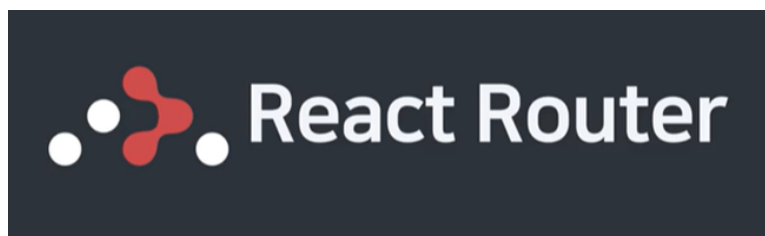
- 라우팅이란?

사용자가 요청한 URL에 따라 해당 URL에 맞는 페이지를 보여주는 것이다.

리액트에서는 라우팅 관련 라이브러리가 많이 있는데, 이중 리액트 라우터가 가장 많이 사용된다.

리액트로 이런 사이트를 만들기 위해서는 따로 추가적인 개발이 필요하다.

React Router 라는 것을 사용하면 이런 기능을 편하게 구현할 수 있다.



React Router는 리액트 컴포넌트로 페이지를 나누는 라이브러리이다.

Router의 핵심 컴포넌트에는 다음과 같이 4가지가 있다.

- Router

```
import {BrowserRouter} from 'react-router-dom';

function Main(){
  return <BrowserRouter>
  </BrowserRouter>
}

export default Main;
```

: React Router에서 사용하는 데이터들을 모두 가지고 있다.

(ex : 현재 주소, 페이지 데이터 등등...)

이게 없으면, React Router를 사용할 수 없다.

Header는 모든 URL에 공통 적용할 Component로 최상단에 위치 할 예정이다.

- Routes, Route

```
<Routes>
  <Route path="/" element={<App feedId={feedId} />}>
    <Route path="/" element={<HomePage />} />
    <Route path="send" element={<sendPage />} />
    <Route path="more" element={<morePage />} />
    <Route path="find" element={<findPage />} />
    <Route path="like" element={<likePage />} />
    <Route path="profile" element={<profilePage />} />
  </Route>
</Routes>
```

: 이 두 컴포넌트는 보통 같이 사용한다.

**Routes** 컴포넌트 안에서 **Route** 컴포넌트로 페이지의 경로와 보여줄 컴포넌트를 지정한다.

**Routes** 컴포넌트는 여러 Route를 감싸서 그 중 규칙이 일치하는 라우트 단 하나만을 렌더링 시켜주는 역할을 한다.

**Route** 컴포넌트의 path속성에 경로, element속성에는 컴포넌트를 넣어 준다.

여러 라우팅을 매칭하고 싶은 경우 URL 뒤에 \*을 사용하면 된다.

- Link

```
<Link to="/">홈페이지</Link>
<Link to="/a">페이지 a</Link>
<Link to="/b">페이지 b</Link>
```

웹 페이지에서 링크를 보여줄 때 사용하는 `<a></a>` 태그 대신 사용한다.

`<a></a>` 는 클릭시 페이지를 새로 불러오기때문에 사용하지 않는다.

생김새는 `<a></a>` 를 사용하지만, History API를 통해 브라우저 주소의 경로만 바꾸는 기능이 내장되어 있다.

- SPA와 리액트 라우터

리액트는 SPA (Single Page Application) 방식으로, 기존 웹 페이지처럼(MPA 방식) 여러 개의 페이지를 사용, 새로운 페이지를 로드하는 방식이 아니다.

새로운 페이지를 로드하지 않고 하나의 페이지 안에서 필요한 데이터만 가져오는 형태를 가진다.

리액트 라우터는 신규 페이지를 불러오지 않는 상황에서 각각의 url에 따라 선택된 데이터를 하나의 페이지에서 렌더링 해주는 라이브러리라고 볼 수 있다.

## 2. React Router 설치하기

리액트 라우터는 리액트에서 제공하는 기본 기능이 아니기 때문에 따로 설치해서 사용해야 한다.

먼저 npm 패키지에 대해서 가볍게 알아보고, react-router-dom 이라는 패키지를 설치해보자.

- 패키지란?

Node.js를 설치하면 npm이라는 프로그램도 함께 설치된다.

여기서 패키지라는 건 자바스크립트 모듈을 모아 놓은 묶음 같은 것이다.

npm은 node package manager의 약자로, 말 그대로 패키지를 설치하거나 삭제하는 것처럼 패키지를 관리하는 프로그램이다.

패키지를 설치하면 미리 작성된 자바스크립트 모듈을 가져다 쓸 수 있다.

리액트를 설치해 사용해보았다면, 아마 이를 react 와 react-dom 이라는 패키지로 사용하고 있었을 것이다.

npmjs.com 에 접속하면 오픈 소스로 공개된 패키지들을 볼 수 있는데, 리액트와 마찬가지로 리액트 라우터도 npm에서 다운로드 하면 된다.

npm 사이트에서 `react-router-dom` 이라고 검색하면 사용할 패키지를 확인할 수 있다.

여기서 잠깐, 만약 리액트를 설치하지 않은 상태라면 먼저 리액트 설치를 진행해야 한다.

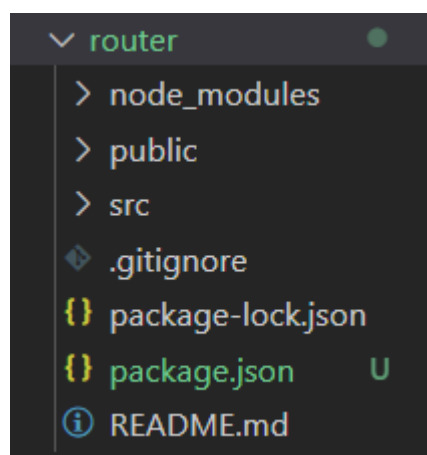
## • 리액트 설치하기

터미널을 열어 `npx create-react-app <폴더 이름>` 또는 `npm init react-app .` 명령어를 입력해 리액트 프로젝트를 생성해준다.

```
Creating a new React app in C:\Users\Myona\Desktop\react\REACT\router.  
  
Installing packages. This might take a couple of minutes.  
Installing react, react-dom, and react-scripts with cra-template...  
  
added 1384 packages in 2m  
  
193 packages are looking for funding  
  run `npm fund` for details  
  
Installing template dependencies using npm...
```

잘 설치됐는지 확인해보면,

router라는 폴더가 생겼고, 그 폴더 안에 여러 파일들이 생성된 것을 볼 수 있다.



`cd <디렉토리 이름>` 명령어로 리액트 프로젝트를 생성한 위치로 이동한다.

```
C:\Users\Myona\Desktop\react\REACT>cd router
C:\Users\Myona\Desktop\react\REACT\router>
```

`npm run start` 또는 `npm start` 명령어를 통해 서버를 실행한다.

```
Compiled successfully!

You can now view router in the browser.

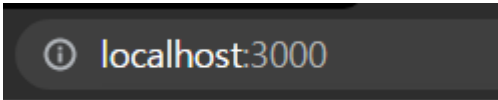
Local:            http://localhost:3000
On Your Network:  http://192.168.130.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully

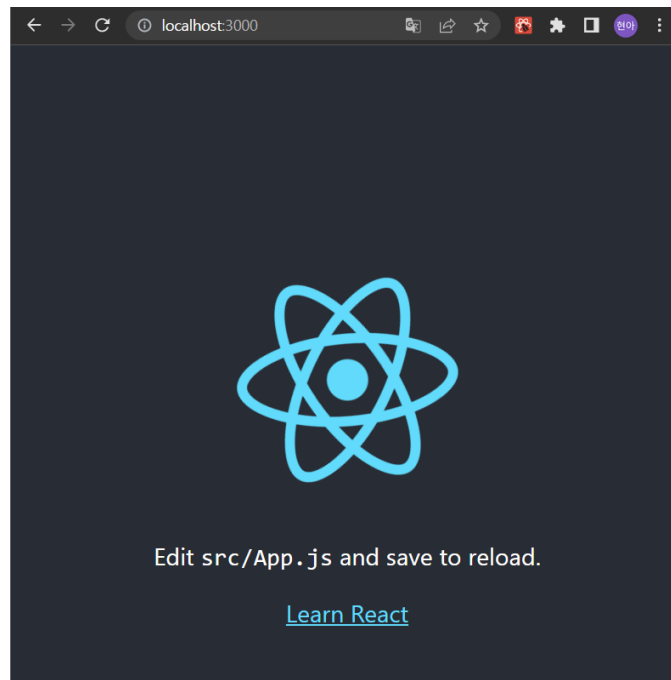
```

혹은 `localhost:3000` 와 같은 url을 통한 접근도 가능하다.



두 방법 모두 결과는 같다! 개발 모드로 프로그램을 시작하게 된다.

프로그램을 시작한 모습은 다음과 같다.



그리고 개발을 모두 마쳤다면, `ctrl + c` 명령어를 통해 서버 닫고 개발 모드 종료한다.

```
webpack compiled successfully
일괄 작업을 끝내시겠습니까 (Y/N)? y
C:\Users\Myona\Desktop\react\REACT\router>
```

### • 라우터 패키지 설치하기

패키지를 설치하려면 package.json이 있는 폴더에서 터미널을 열고, `npm install <패키지 이름>` 이라는 명령어를 실행하면 된다.

VS Code에서 프로젝트를 열어서 터미널을 열고 `npm install react-router-dom@6` 이라고 입력하자.

```
C:\Users\Myona\Desktop\react\REACT\router>npm install react-router-dom@6
npm notice Beginning October 4, 2021, all connections to the npm registry - including for
package installation - must use TLS 1.2 or higher. You are currently using plaintext ht
p to connect. Please visit the GitHub blog for more information: https://github.blog/202
-08-23-npm-registry-deprecating-tls-1-0-tls-1-1/
npm notice Beginning October 4, 2021, all connections to the npm registry - including for
package installation - must use TLS 1.2 or higher. You are currently using plaintext ht
p to connect. Please visit the GitHub blog for more information: https://github.blog/202
-08-23-npm-registry-deprecating-tls-1-0-tls-1-1/

added 9 packages in 4s

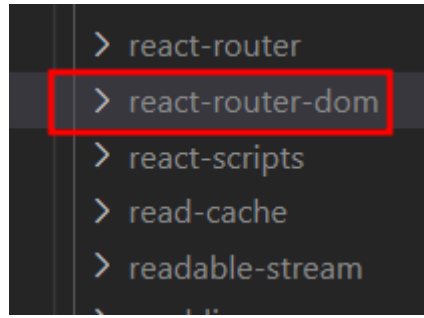
67 packages are looking for funding
run `npm fund` for details
```

여기서 `react-router-dom` 은 패키지 이름이고,  
뒤에 있는 `@6` 은 패키지 버전을 6점대로 설치하겠다는 의미이다.

잘 설치됐는지 확인해보면,

```
router > {} package.json > {} browserslist > [ ] development > 1
1  {
2    "name": "router",
3    "version": "0.1.0",
4    "private": true,
5    "dependencies": {
6      "@testing-library/jest-dom": "^5.16.4",
7      "@testing-library/react": "^13.3.0",
8      "@testing-library/user-event": "^13.5.0",
9      "react": "^18.2.0",
10     "react-dom": "^18.2.0",
11     "react-router-dom": "^6.3.0",
12     "react-scripts": "5.0.1",
13     "web-vitals": "^2.1.4"
14   },
```

package.json 파일에 `dependencies` 아래에 `react-router-dom` 이란 게 추가되고



`node_modules/` 폴더에 `react-router-dom` 이라는 폴더가 생긴 것이 보인다.

- **react-router-dom 패키지를 사용할 때 주의할 점!**

VSCode에서 자동완성 기능을 사용하다 보면, 가끔 `react-router-dom` 이 아닌 `react-router` 라는 패키지에서 임포트할 때가 있다.

여기서 `react-router` 라는 패키지는 리액트 라우터를 만드는 개발자들이 내부적으로 사용하는 것이다.

`react-router-dom` 을 설치하면 함께 설치되는 패키지이기 때문에 자동완성에도 종종 뜨는 것이다.

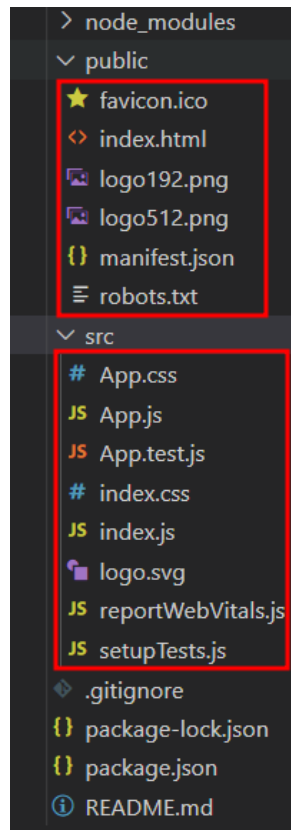
웹 사이트를 만들 때는 반드시 `react-router-dom` 패키지를 통해서 임포트해야 하니까, 혹시 잘못 임포트하지 않도록 주의하자.

### 3. 리액트 라우터 사용해보기

- **간단한 리액트 웹 페이지 구성하기**

먼저, 리액트로 프로젝트를 생성하며 자동으로 생성된 파일들을 다 지워준다.





그리고 다음 파일들을 생성해주자.

src (디렉토리) > index.js

```
import ReactDOM from "react-dom";
import Main from "../main";

ReactDOM.render(<Main />, document.getElementById("root"));
```

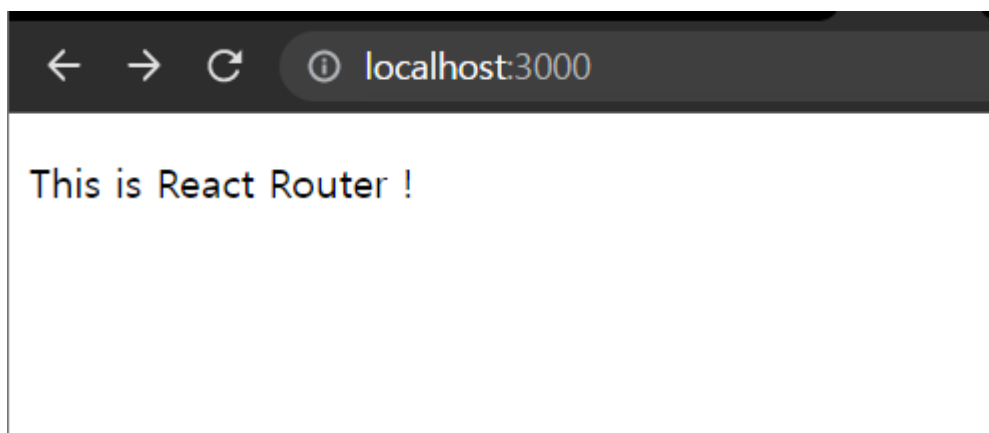
public > index.html

```
<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Router practice</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

src > main.js

```
function Main() {  
  return <p>This is React Router !</p>;  
}  
  
export default Main;
```

이렇게 코드를 작성해주면,

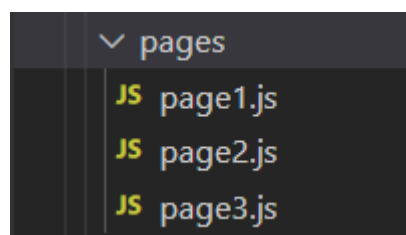


이렇게 간단한 웹 페이지가 하나 만들어진다.

이제 Router를 본격적으로 적용해서 웹 페이지를 구성해보자.

### • 페이지 생성하기

페이지를 3개 정도 만들어준다.



페이지 코드 각각의 코드는 간단하게 작성해준다.

src/images > page1.js

```
function Page1() {  
  return <p>This is Page 1.</p>;  
}  
  
export default Page1;
```

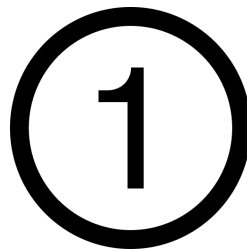
(page2.js page3.js도 위와 같이 작성해주면 된다.)

### • 내비게이션 바 만들기

내비게이션 바를 하나 만들어줄 것이다.

내비게이션 바의 아이콘을 클릭하면 page1, page2, page3로 각각 이동하도록 만들고 싶다.

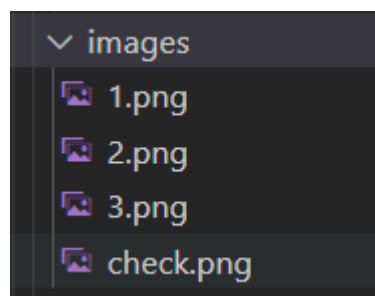
먼저 아이콘에 사용할 이미지 파일을 다운 받아 주자.



이런 번호 이미지를 사용해주었다.

(자유롭게 자신이 원하는 이미지를 사용해보자! )

src/images 폴더에 저장해주자.



1.png, 2.png, 3.png 는 각각 page1, 2, 3에 해당하는 아이콘이고, check.png는 메인 페이지에 해당하는 아이콘이다.

이렇게 다운 받은 이미지 파일을 적용해서 내비게이션 바를 본격적으로 만들어보자.

`Link` 를 사용할 것인데, 이것 사용하려면 먼저

```
import { Link } from "react-router-dom";
```

이 코드를 통해 `Link` 컴포넌트를 불러와야 한다.

`Link`는 `to` 라는 프로퍼티를 가진다.

이 프로퍼티 뒤에 이동할 경로를 지정해주면 된다.

```
<Link to="/">
```

이런 식으로 말이다.

이때, 경로의 맨 앞에 `/` 를 붙이는 것은, 이 경로가 **절대경로**라는 것을 의미한다.

이렇게 하면, 코드가 실행되며 localhost:3000이라는 도메인의 `/<이동할 경로>` 를 붙여준 주소로 이동하게 된다.

`<a></a>` 와 동일하게 동작한다고 보면 된다.

src > nav.js

```
import { Link } from "react-router-dom";
import Container from "../container";
import CheckImg from "../images/check.png";
import Img1 from "../images/1.png";
import Img2 from "../images/2.png";
import Img3 from "../images/3.png";

import styles from "../css/nav.module.css"; //nav바에 대한 css

function Nav() {
  return (
    <div className={styles.nav}>
```

```

    <Container className={styles.container}>
      <Link to="/">
        <img src={CheckImg} alt="Check Logo" className={styles.imgSize} />
      </Link>
      <li>
        <Link to="/page1">
          <img src={Img1} alt="Page1 Logo" className={styles.imgSize} />
        </Link>
      </li>
      <li>
        <Link to="/page2">
          <img src={Img2} alt="Page2 Logo" className={styles.imgSize} />
        </Link>
      </li>
      <li>
        <Link to="/page3">
          <img src={Img3} alt="Page3 Logo" className={styles.imgSize} />
        </Link>
      </li>
    </Container>
  </div>
);
}

export default Nav;

```

기능적인 구현은 여기서도 충분하지만, 조금 더 깔끔하게 보이도록 하기 위해 css와 container.js라는 파일을 추가해준다.

(이 부분은 생략해도 좋다!)

src/css > nav.module.css

```

.container {
  width: 100%;
  max-width: 840px;
  margin: 0 auto;
}

@media (max-width: 840px) {
  .container {
    padding: 0 calc(14px * 5);
  }
}.nav {
  position: relative;
  z-index: 1;
  padding: 15px 0;
  background-color: #fff;
  box-shadow: var(--box-shadow);
}

.imgSize {
  width: 40px;
}

```

```

    object-fit: cover;
  }

.container {
  display: flex;
  align-items: center;
  justify-content: space-between;
}

ul.menu {
  display: flex;
  align-items: center;
  padding: 0;
  margin: 0;
  list-style: none;
}

ul.menu > li:not(:last-child) {
  margin-right: 30px;
}

ul.menu a:hover,
ul.menu a:active {
  text-decoration: underline;
}.nav {
  position: relative;
  z-index: 1;
  padding: 15px 0;
  background-color: #fff;
  box-shadow: var(--box-shadow);
}

.imgSize {
  width: 40px;
  object-fit: cover;
}

.container {
  display: flex;
  align-items: center;
  justify-content: space-between;
}

ul.menu {
  display: flex;
  align-items: center;
  padding: 0;
  margin: 0;
  list-style: none;
}

ul.menu > li:not(:last-child) {
  margin-right: 30px;
}

ul.menu a:hover,
ul.menu a:active {
  text-decoration: underline;
}.nav {

```

```

position: relative;
z-index: 1;
padding: 15px 0;
background-color: #fff;
box-shadow: var(--box-shadow);
}
.imgSize {
width: 40px;
object-fit: cover;
}

.container {
display: flex;
align-items: center;
justify-content: space-between;
}

ul.menu {
display: flex;
align-items: center;
padding: 0;
margin: 0;
list-style: none;
}

ul.menu > li:not(:last-child) {
margin-right: 30px;
}

ul.menu a:hover,
ul.menu a:active {
text-decoration: underline;
}

```

src > container.js

```

import classNames from "classnames";
import styles from "../css/container.module.css";

function Container({ className, children }) {
  return (
    <div className={classNames(styles.container, className)}>{children}</div>
  );
}

export default Container;

```

여기서, 만약 classNames를 사용하려면 `npm init classnames`

명령어를 실행해주어야 한다.

src/scc > container.module.css

```
.container {
  width: 100%;
  max-width: 840px;
  margin: 0 auto;
}

@media (max-width: 840px) {
  .container {
    padding: 0 calc(14px * 5);
  }
}
```

### • Routes로 페이지 나누기

이제 위에서 만든 각 페이지에 대한 컴포넌트들을 배치해줄 것이다.

먼저, 사용할 컴포넌트들을 import 해준다.

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

아래와 같이 `<Routes></Routes>` 안에 `<Route></Route>` 컴포넌트들을 만들어 배치해주면 된다.

```
<Routes>
  <Route path="/" element={<HomePage />} />
  ...
</Routes>
```

`path` 프로퍼티로 경로를 지정하고,

`element` 프로퍼티로 보여줄 컴포넌트를 지정할 수 있다.

이때, `element` 로 넘여줄 컴포넌트를 jsx 형태라는 것을 주의하자.

`Routes` 컴포넌트는 여러개의 `Route` 컴포넌트를 포함한다.

`Routes` 를 렌더링할 때, 내부의 `Route` 컴포넌트를 차례대로 검사하며 현재 경로가 `Route` 에서 지정한 `path` 와 일치하는지를 체크한다.



일치하는 경로를 찾으면 `element` 프로퍼티로 지정한 컴포넌트를 렌더링해서 화면으로 보여준다.

## • 라우터 컴포넌트 감싸기

이제 프로젝트의 최상위 컴포넌트인 `Main.js` 파일에 가서 라우터 컴포넌트를 적용해보자.

`BrowserRouter` 라는 컴포넌트를 불러와서 컴포넌트 전체를 감싸준다.

사실 라우터에는 여러 종류가 있지만, 우리는 브라우저의 기본적인 방식으로 사용할 것이기 때문에 `BrowserRouter` 라는 걸 사용할 것이다.

main.js

```
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Page1 from "../pages/page1"; //main home page
import Page2 from "../pages/page2";
import Page3 from "../pages/page3";
import Nav from "../nav";

function Main() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Nav />} />
        <Route path="page1" element={<Page1 />} />
        <Route path="page2" element={<Page2 />} />
        <Route path="page3" element={<Page3 />} />
      </Routes>
    </BrowserRouter>
  );
}

export default Main;
```

여기까지 잘 따라왔다면 성공적으로 웹 페이지가 생성되었을 것이다!

결과를 확인해보자.

`npm run start` 실행 후 `ctrl + c` 로 종료를 하지 않았다면 웹 페이지를 새로 고침하며 계속해서 결과를 확인할 수 있다.

그러니, 개발을 완전히 마치기 전까지는 웬만하면 종료를 하지 말고 계속해서 페이지의 변화를 지켜보자!

## 4. 결과 확인

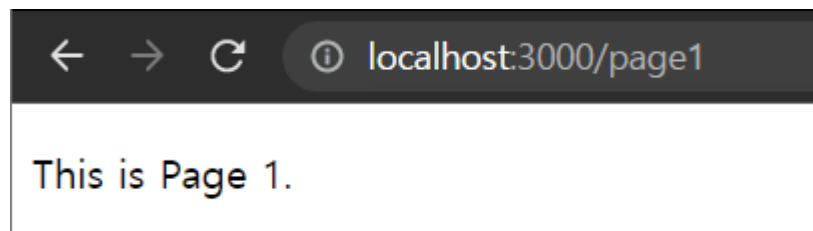
최종적으로 이렇게 페이지가 만들어진다.



처음 체크 아이콘은 눌러도 아무런 변화가 없다.

```
<Link to="/">
  <img src={Img1} alt="Check Logo" className={styles.imgSize} />
</Link>
```

1, 2, 3, 아이콘을 누르면 각각 다음과 같은 결과가 나온다.



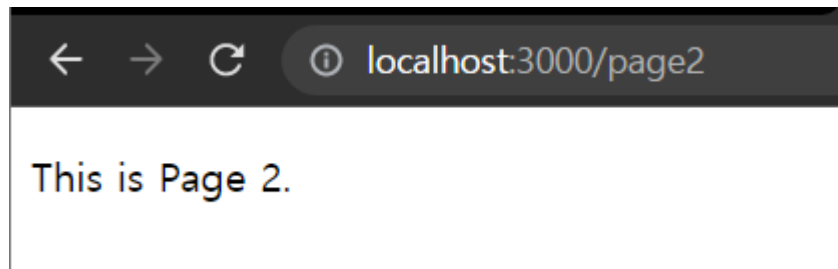
1아이콘을 누르니 page 1으로 이동했다.

```
<Link to="/page1">
  <img src={Img1} alt="Page1 Logo" className={styles.imgSize} />
</Link>
```

이 코드가 반영된 것이다.

이 코드가 실행되며 `localhost:3000` 도메인 + `/<이동할 경로>` 주소로 이동하게 된다.

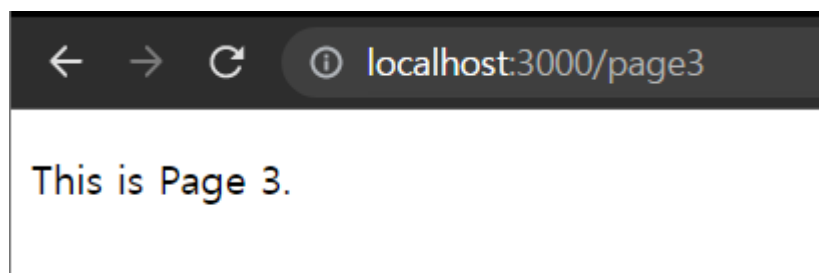
여기에서는 그 주소가 `localhost:3000/page1` 이 되는 것이다.



2 아이콘을 누르니 page 2으로 이동했다.

```
<Link to="/page2">
  <img src={Img1} alt="Page2 Logo" className={styles.imgSize} />
</Link>
```

이 코드가 반영된 것이다.



3 아이콘을 누르니 page 3으로 이동했다.

```
<Link to="/page3">
  <img src={Img1} alt="Page3 Logo" className={styles.imgSize} />
</Link>
```

이 코드가 반영된 것이다.

이 실습 코드는 매우 간단한 것이었다.

리액트 라우터를 이용해 더욱 많고 다양한 웹 페이지들을 구현할 수 있을 것이다.

라우터를 사용하면 서버의 부담을 줄여주기 때문에 서버의 자원을 절약할 수 있다는 큰 장점이 있다.

또한, 라우터는 리액트의 장점인 컴포넌트 구조로  
부분적 변화를 효율적으로 반영 할 수 있다는 것을 극대화시켜준다.  
이번 실습에서 다루지 않은 리액트 라우터 관련 내용들도 많으니, 이번 기회로 관심이 생겼  
다면 찾아보며 더 공부해보길 바란다!