React

◆ 리액트 라우터

정수아

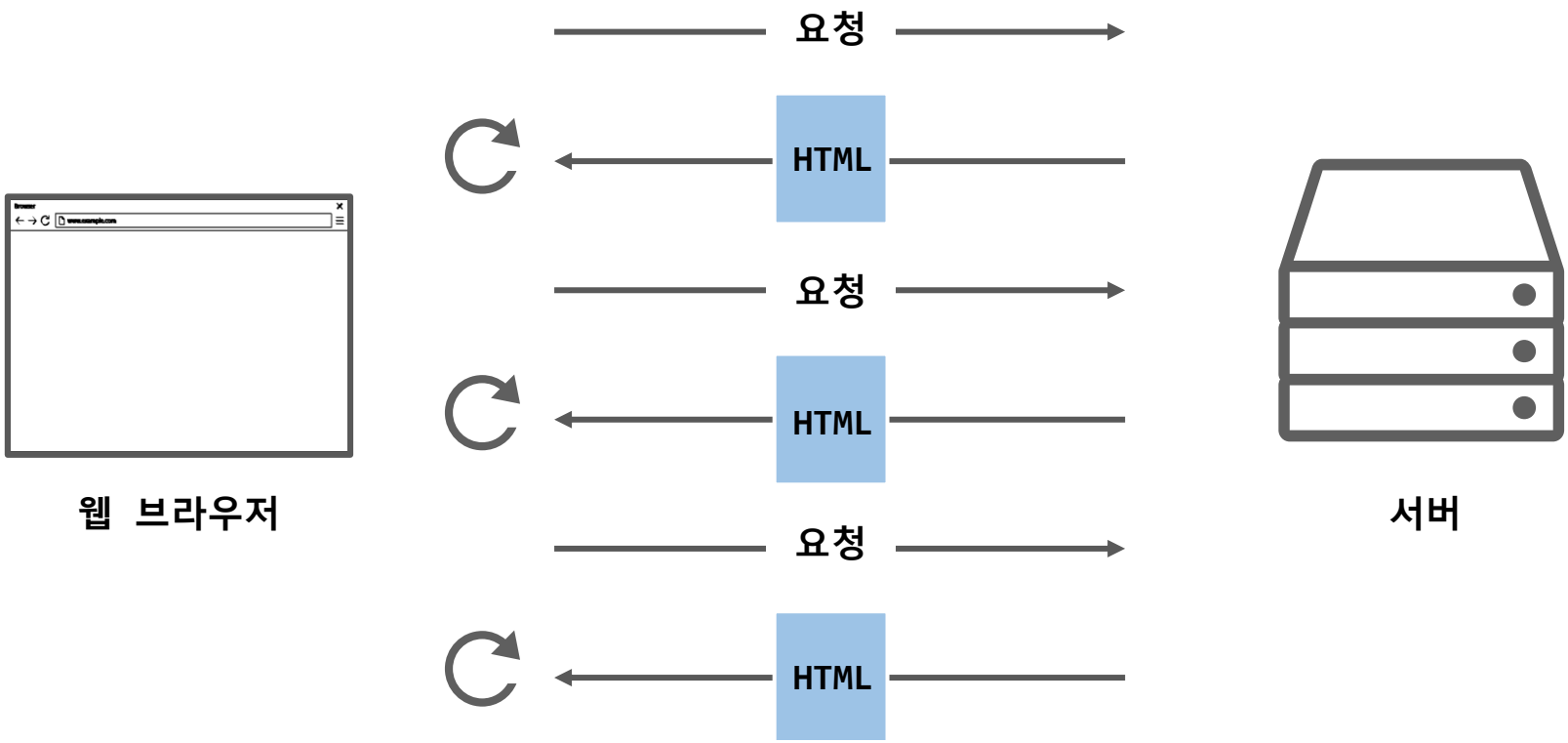
Contents

01

싱글 페이지 어플리케이션

전통적인 웹 페이지

❖ 일반 웹 사이트의 동작 과정



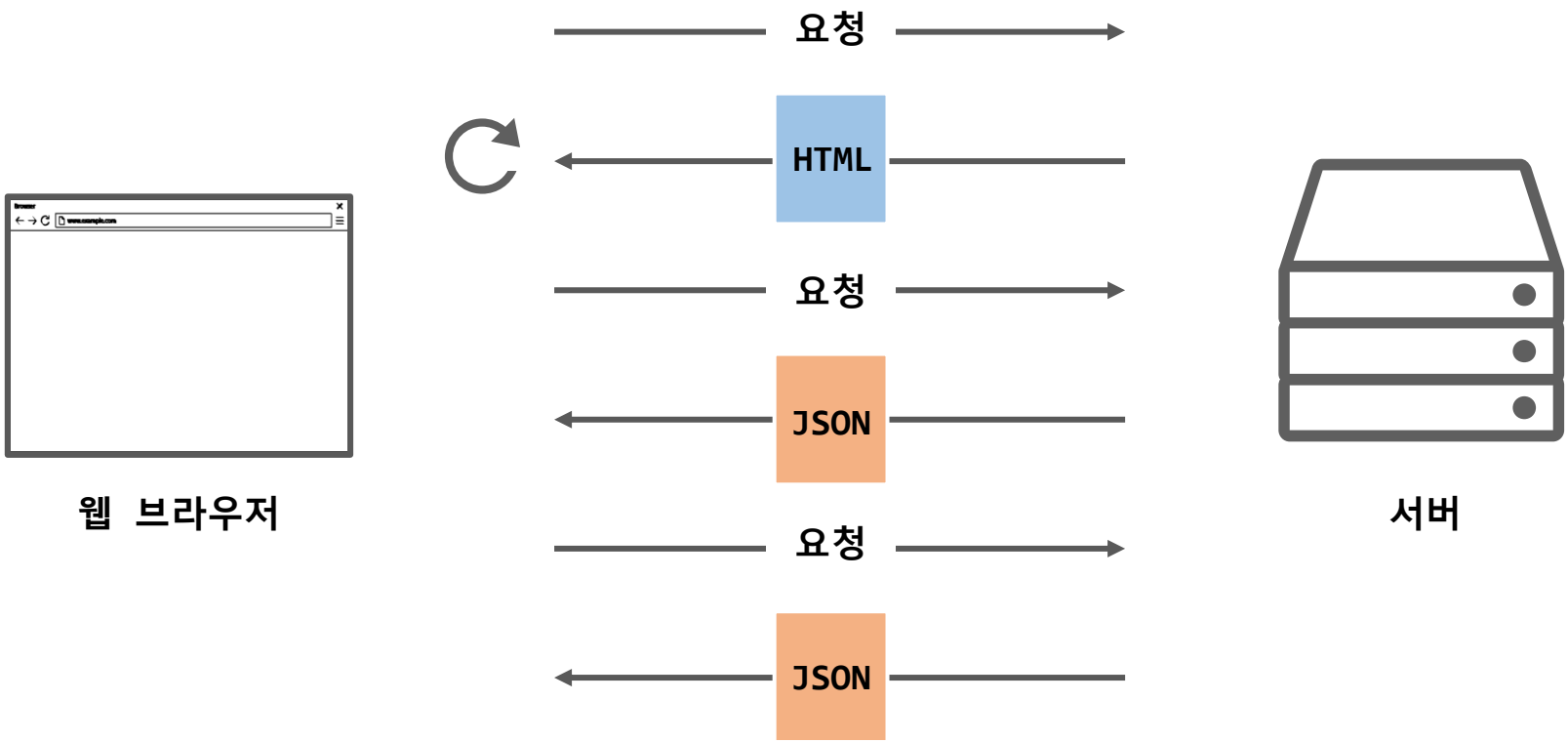
싱글 페이지 어플리케이션

❖ **웹 사이트의 화면 일부를 수정하기 위해 노력한 과정**

- 초기 웹 사이트
  - 각 화면에 해당하는 웹 문서를 일일이 제작
- 서버에 저장된 값을 화면에 반영하는 동적인 웹 문서 도구 개발
  - 서버 사이드 렌더링(Server Side Rendering)
  - ASP(Active Server Page), JSP(Java Server Page)
- 웹 서버가 아닌 자바스크립트 코드로 웹 문서를 생성
  - 싱글 페이지 어플리케이션(Single Page Application)

싱글 페이지 어플리케이션

❖ 싱글 페이지 어플리케이션(Single Page Application)



싱글 페이지 어플리케이션

❖**싱글 페이지 어플리케이션 장점**

- 페이지 전환 속도가 빠름
- 주소가 변경되어도 서버에 추가로 웹 문서를 요청하는 작업 필요 없음

❖ **싱글 페이지 어플리케이션 단점**

- 어플리케이션의 규모가 커지면 자바스크립트 파일 또한 커짐
  - 페이지 로딩 시, 사용자가 실제로 방문하지 않을 수도 있는 페이지의 스크립트까지 불러옴

02

리액트 라우터

리액트 라우터

❖ **라우팅이란?**
- 사용자가 요청한 URL에 따라 해당 URL에 맞는 페이지를 보여주는 것
- 페이지 이동

❖ **리액트 라우팅 라이브러리**
- 리액트 라우터(react-router)
- 리치 라우터(reach-router)
- Next.js

리액트 라우터

❖ 리액트 라우터

- 사용자가 입력한 주소를 감지하는 역할을 함
- 여러 환경에서 동작할 수 있도록 다양한 라우터 컴포넌트를 제공
- https://reactrouter.com/en/main

- 라우터 컴포넌트 종류

라우터 컴포넌트 종류	설명
BrowserRouter	HTML5를 지원하는 브라우저의 주소를 감지
HashRouter	해시주소(http://localhost#login)를 감지
MemoryRouter	메모리에 저장된 이전, 이후 주소로 이동
NativeRouter	리액트 네이티브를 지원하는 라우터
StaticRouter	브라우저의 주소가 아닌 프로퍼티로 전달된 주소를 사용

리액트 라우터

❖ 리액트 라우터 설치

```
$ npm install react-router-dom
```

리액트 라우터

❖ 리액트 라우터 생성

```
const router = createBrowserRouter([
  {
     path: "/",
     element: <div>Home</div>,
     errorElement: <div>Page Not Found</div>
  }
]);
```

❖ 리액트 라우터 적용

```
const App = () => {
  return <RouterProvider router={router} />;
};
```

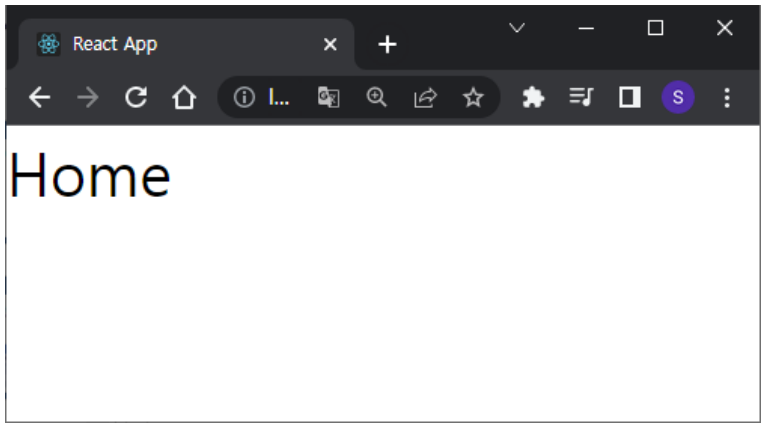
[실습] 리액트 라우터 생성 및 적용

❖ **App.js**

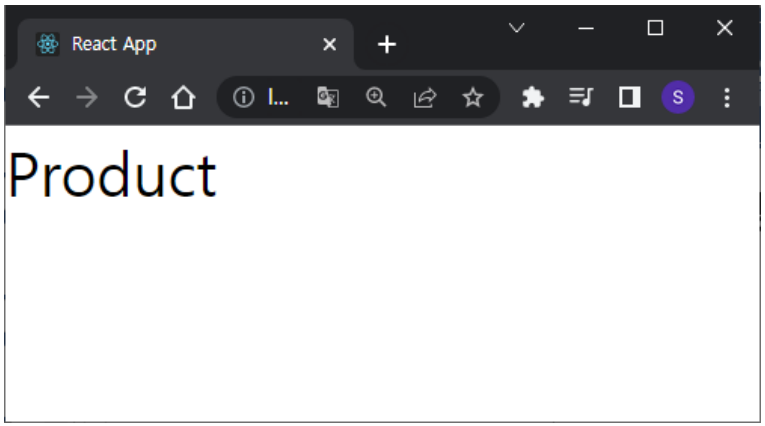
```
const router = createBrowserRouter([
  {
    path: "/",
    element: <div>Home</div>,
    errorElement: <div>Page Not Found</div>
  },
  {
    path: "/product",
    element: <div>Product</div>,
    errorElement: <div>Page Not Found</div>
  }
]);

const App = () => {
  return <RouterProvider router={router} />;
};

export default App;
```

[실습] 리액트 라우터 생성 및 적용

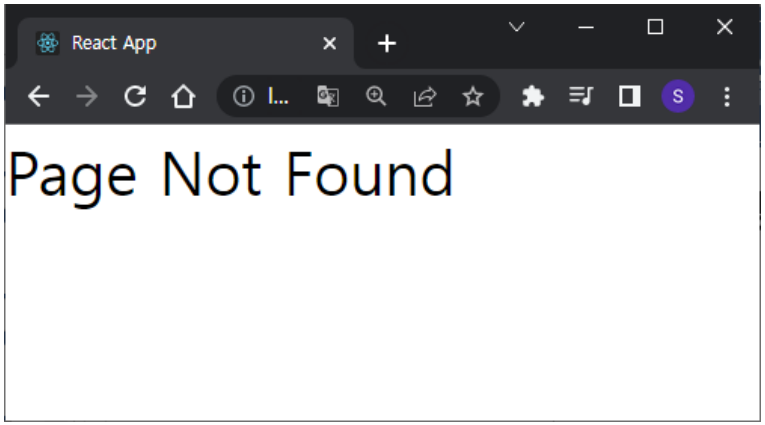
❖ 실행 결과



http://localhost:3000



http://localhost:3000/product



http://localhost:3000/login

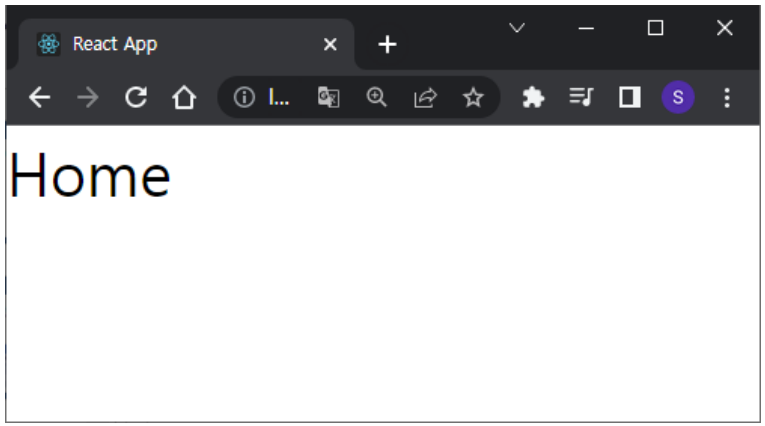
[실습] 컴포넌트로 변경

❖ App.js

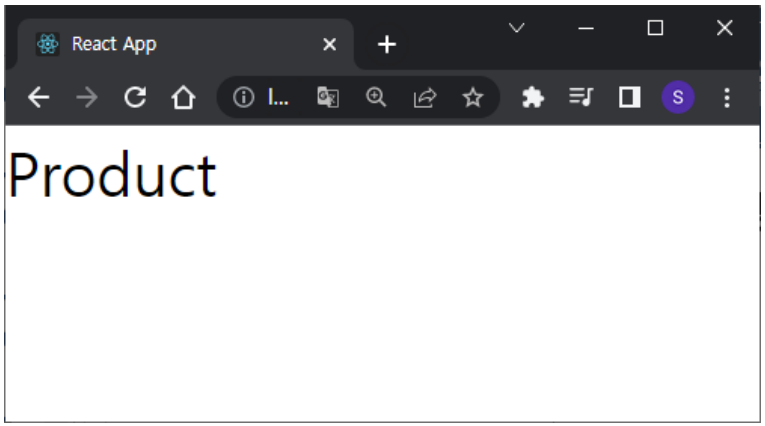
```
const router = createBrowserRouter([
  {
    path: "/",
    element: <Home />,
    errorElement: <NotFound />
  },
  {
    path: "/product",
    element: <Product />,
    errorElement: <NotFound />
  }
]);

const App = () => {
  return <RouterProvider router={router} />;
};

export default App;
```

[실습] 컴포넌트로 변경

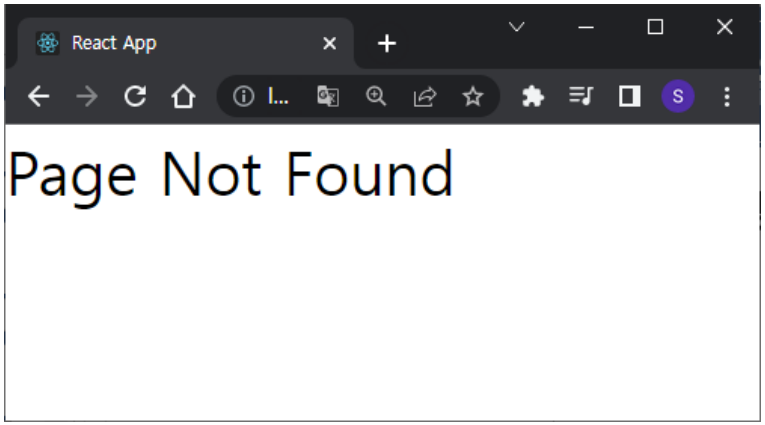
❖ 실행 결과



http://localhost:3000



http://localhost:3000/product

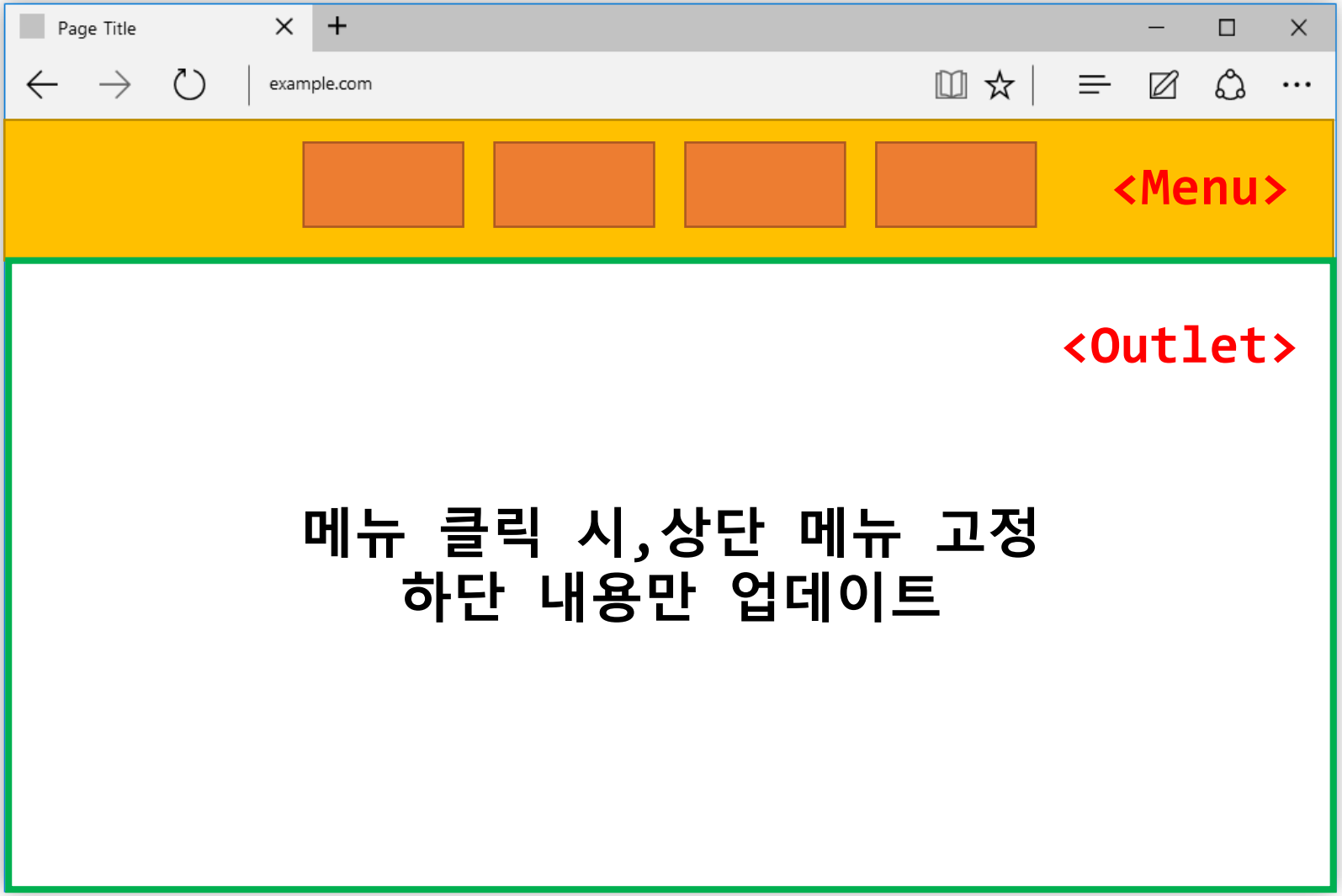


http://localhost:3000/login

03

Outlet 컴포넌트
Link 컴포넌트

Outlet 컴포넌트



Outlet 컴포넌트

❖ 리액트 라우터 생성

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <Root />,
    errorElement: <NotFound />,
    children: [
      { index: true, element: <Home /> },
      { path: "/product", element: <Product /> }
    ]
  }
]);
```

[실습] Outlet 컴포넌트

❖ Root.js

```
import Menu from "./Menu";
import { Outlet } from "react-router-dom";

const Root = () => {
  return (
    <div>
      <Menu />
      <Outlet />
    </div>
  );
};

export default Root;
```

[실습] Outlet 컴포넌트

❖ Menu.js

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

const Menu = () => {
  return (
    <div>
      <Link to="/">Home</Link>
      <Link to="/product">Product</Link>
    </div>
  );
};

export default Menu;
```

Link 컴포넌트

❖ **웹 페이지에서 링크를 추가할 때 <a> 태그를 사용**

- 리액트에서는 <a> 태그를 직접 사용하면 안됨
- <a> 태그를 클릭하여 페이지 이동 시, 브라우저가 페이지를 새로 불러오기 때문
- Link 컴포넌트를 이용하여 페이지 이동 가능

❖ **Link 컴포넌트**

- 페이지를 새로 불러오는 것을 막고, History API를 통해 브라우저 주소의 경로만 변경하는 기능

```
<Link to="주소">링크이름</Link>
```

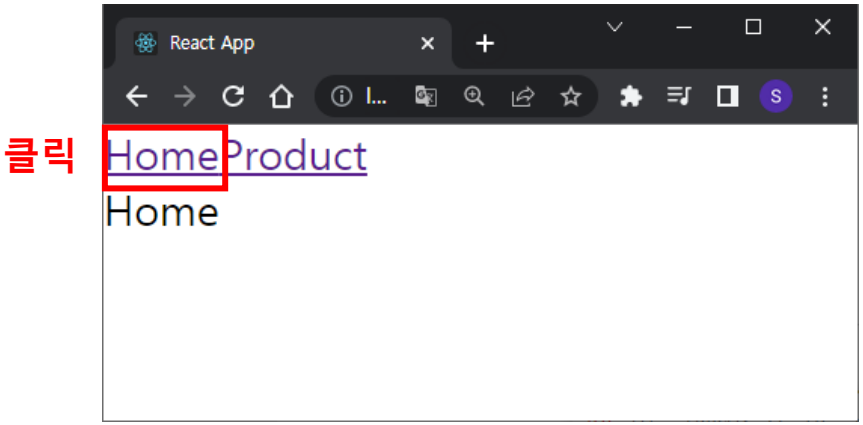
[실습] Outlet 컴포넌트

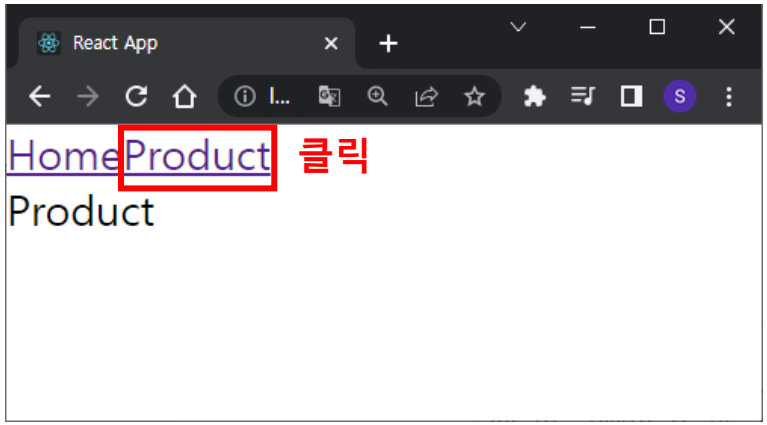
❖ App.js

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <Root />,
    errorElement: <NotFound />,
    children: [
      { index: true, element: <Home /> },
      { path: "/product", element: <Product /> },
    ],
  }
]);

const App = () => {
  return <RouterProvider router={router} />;
};

export default App;
```

[실습] Outlet 컴포넌트

❖ 실행 결과





04 useNavigate()

페이지 전환

[실습] 페이지 전환

❖ 리액트 라우터 생성

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <Root />,
    errorElement: <NotFound />,
    children: [
      { index: true, element: <Home /> },
      { path: "/product", element: <Product /> },
      { path: "/product/:productId", element: <ProductInfo /> },
    ],
  },
]);
```

[실습] 페이지 전환

❖ App.js

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <Root />,
    errorElement: <NotFound />,
    children: [
      { index: true, element: <Home /> },
      { path: "/product", element: <Product /> },
      { path: "/product/:productId", element: <ProductInfo /> }
    ],
  }
]);

const App = () => {
  return <RouterProvider router={router} />;
};

export default App;
```

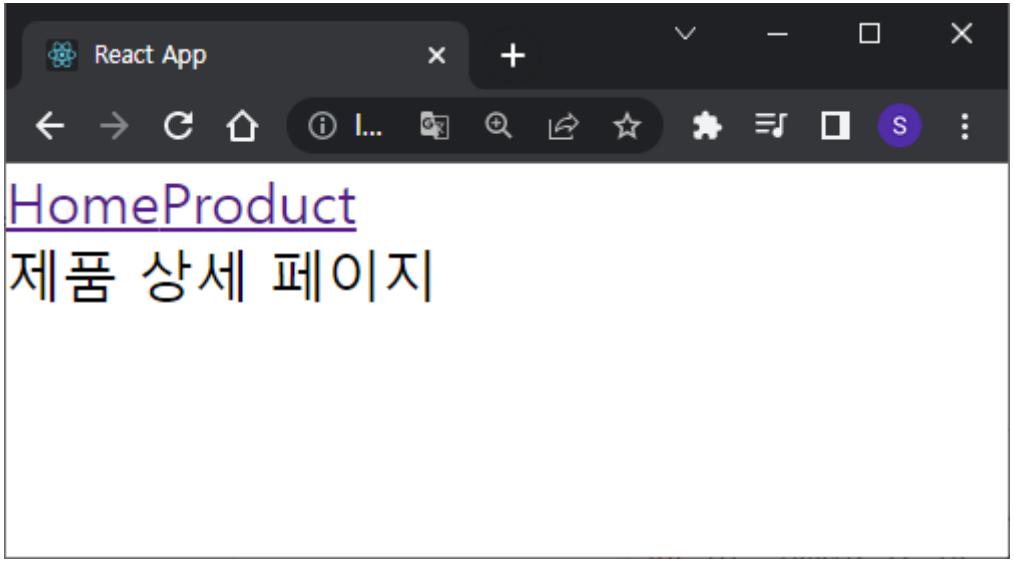
[실습] 페이지 전환

❖ ProductInfo.js

```
const ProductInfo = () => {
   return <div>제품 상세 페이지</div>;
};

export default ProductInfo;
```

[실습] 페이지 전환

❖ 실행 결과



http://localhost:3000/product/2

useNavigate()

❖ **useNavigate()**

- 페이지 방문 기록을 쉽게 관리할 수 있는 hooks
- 내장 함수를 사용하여 뒤로 가기, 특정 페이지로 이동하기 등이 가능

- 사용 방법

```
const navigate = useNavigate();

// home 경로로 이동
function goHome(){
  navigate('/home');
}
```

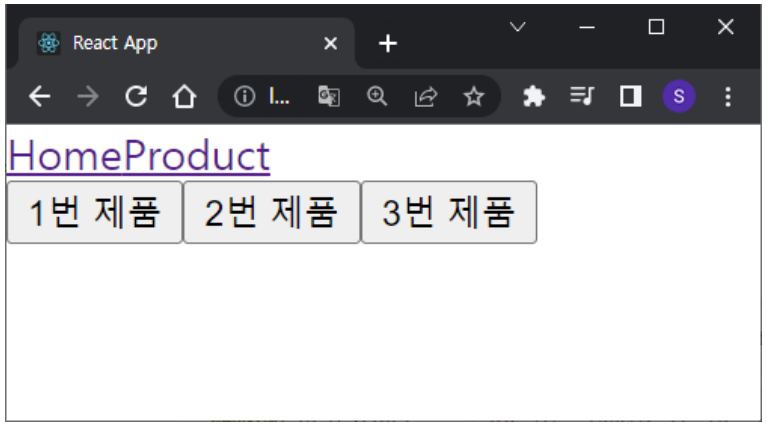
[실습] useNavigate()

❖ Product.js

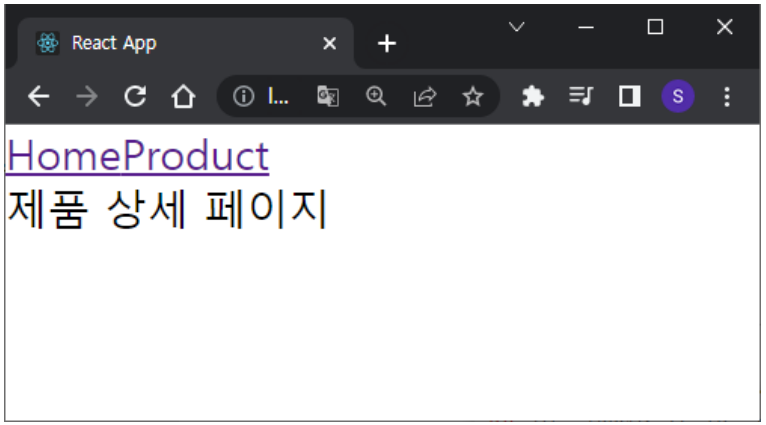
```
const Product = () => {
  const [productId, setProductId] = useState("");
  const navigate = useNavigate();

  const clickHandler = (e) => {
     setProductId(e.target.value);
     navigate(`/product/${productId}`);
  };

  return (
    <>
      <button onClick={clickHandler} value="p001">1번 제품</button>
      <button onClick={clickHandler} value="p002">2번 제품</button>
      <button onClick={clickHandler} value="p003">3번 제품</button>
    </>
  );
};

export default Product;
```

[실습] useNavigate()

❖ 버튼을 두 번 클릭해야 페이지가 전환되는 문제 발생



버튼 1번 클릭



버튼 2번 클릭

❖ 이유

- useState()를 이용한 상태 업데이트는 비동기적이기 때문에 변경사항이 즉시 반영되지 않음

[실습] useNavigate()

❖ Product.js

```
const Product = () => {
  const [productId, setProductId] = useState("");
  const navigate = useNavigate();

  const clickHandler = (e) => {
     setProductId(e.target.value);
  };

  useEffect(() => {
     navigate(`/product/${productId}`);
  }, [productId]);

  return (
    <>
      <button onClick={clickHandler} value="p001">1번 제품</button>
      <button onClick={clickHandler} value="p002">2번 제품</button>
      <button onClick={clickHandler} value="p003">3번 제품</button>
    </>
  );
};
export default Product;
```

05

useParams()

useParams()

❖ **useParams()**

- URL에 포함되어 있는 파라미터 값을 가져와서 사용할 수 있도록 해주는 hooks

❖ **사용 방법**

- 데이터 전달

```
{ path: "/product/:productId", element: <ProductInfo /> }
```

- 데이터 사용

```
const { productId } = useParams();
```

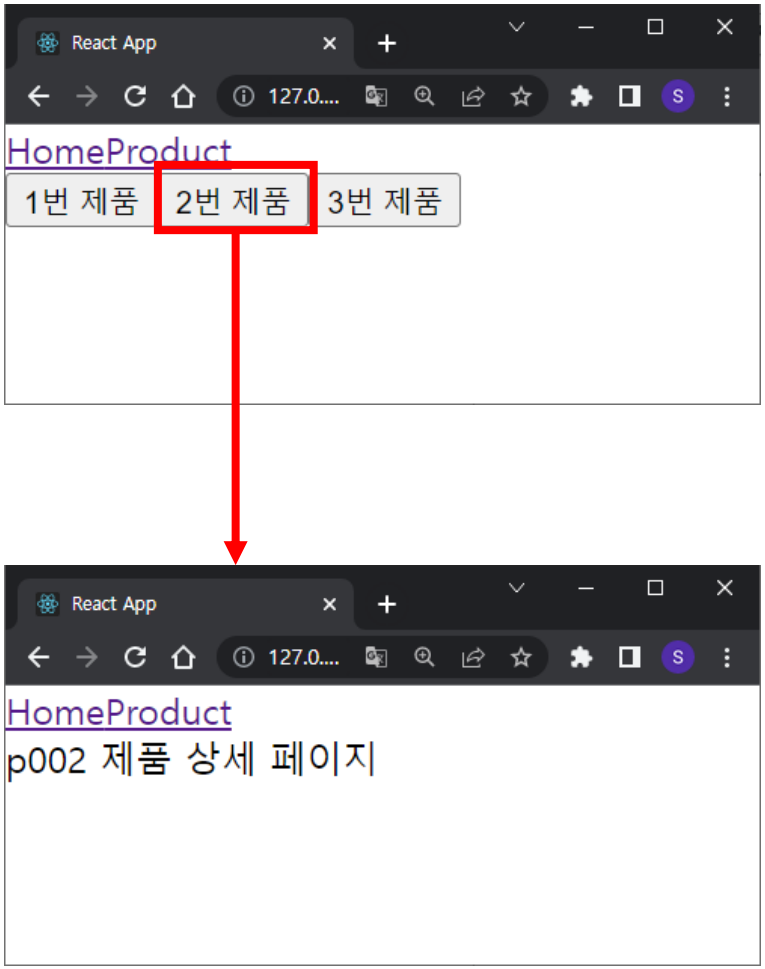
[실습] useParams()

❖ ProductInfo.js

```
const ProductInfo = () => {
  const { productId } = useParams();

  return <div>{productId} 제품 상세 페이지</div>;
};

export default ProductInfo;
```

[실습] useParams()

❖ 실행 결과




THANK YOU