1_라우터컴포넌트.js

```
1  import { BrowserRouter , Routes , Route , Link }from "react-router-dom"
2
3  import 컴포넌트1 from "../day01/1_컴포넌트"
4  import 컴포넌트2 from "../day01/2_컴포넌트"
5  import 컴포넌트3 from "../day01/3_컴포넌트"
6  import 컴포넌트4 from "../day01/4_컴포넌트"
7
8  export default function 라우터컴포넌트( props ) {
9      return ( <>
10         <BrowserRouter>
11             <고정컴포넌트 />
12             <Routes>
13                 <Route path="/day01/컴포넌트1" element = { <컴포넌트1/> } />
14                 <Route path="/day01/컴포넌트2" element = { <컴포넌트2/> } />
15                 <Route path="/day01/컴포넌트3" element = { <컴포넌트3/> } />
16                 <Route path="/day01/컴포넌트4" element = { <컴포넌트4/> } />
17             </Routes>
18         </BrowserRouter>
19     </> )
20 }
21 function 고정컴포넌트( props ){
22     return(<>
23         <div>
24             <Link to='/day01/컴포넌트1' > 컴포넌트1 </Link>
25             <Link to='/day01/컴포넌트2' > 컴포넌트2 </Link>
26             <Link to='/day01/컴포넌트3' > 컴포넌트3 </Link>
27             <Link to='/day01/컴포넌트4' > 컴포넌트4 </Link>
28         </div>
29     </>)
30 }
```

tip] 리액트 라우터 돔

1. 가상 경로[HTTP URL]를 만들어서 연결된 컴포넌트로 전환 해주는 라이브러리
2. 설치
   https://www.npmjs.com/
   react-router-dom 검색
   npm i react-router-dom [6.17.0] 수업 버전!!!
3. import 하기
   import { BrowserRouter , Routes , Route , Link } from "react-router-dom"
4. 라우터 컴포넌트 가상URL만들기
   <BrowserRouter>
   <Routes >
   <Route path='컴포넌트URL정의' element = { <컴포넌트명 /> } />
   </Routes >
   </BrowserRouter>
5. 라우터 컴포넌트 가상URL 호출/매핑하기
     페이지 리로드 O
   또는
   <Link to='라우터경로'> </Link> 페이지 리로드 X

- 리액트 라우터 컴포넌트 정리

1. <BrowserRouter> : 가상 URL 관리[브라우저 URL 동기화]
2. <Routes> : 가장 적합한 <Route> 컴포넌트를 검토하고 찾는다.
   요청된 path에 적합한 <Route> 찾아서 <Routes> 범위내 렌더링
3. <Route> : 실제 URL 경로 지정해주는 컴포넌트
   <Route path="login" element={ <Login /> } />
   http://localhost:3000/login get 요청시 Login 컴포넌트 반환
4. <Link> : 페이지를 갱신하지 않고 렌더링 방식으로 이동하려면 Link 컴포넌트를 사용

1_Axios컴포넌트.js

```jsx
import axios from 'axios' // npm i axios

export default function Axios컴포넌트( props ) {

    function 함수1( e ){ console.log(e); }
    const 함수2 = ( e ) => { console.log(e); }
    const 함수3 = ( e , data ) => { console.log(data); }

    function doGet(){
        axios.get("https://jsonplaceholder.typicode.com/posts"  )
            .then( r => {  console.log( r ); })
        axios.get("https://jsonplaceholder.typicode.com/posts/1"  )
            .then( r => {  console.log( r ); })
        axios.get("https://jsonplaceholder.typicode.com/comments" , { params: {  'postId' : 1 } }  )
           .then( r => {  console.log( r ); })
    }
    function doPost(){
        axios.post("https://jsonplaceholder.typicode.com/posts"  )
            .then( r => {  console.log( r ); })
    }
    function doPut(){
        axios.put("https://jsonplaceholder.typicode.com/posts/1"  )
            .then( r => {  console.log( r ); })
    }
    function doDelete(){
        axios.delete("https://jsonplaceholder.typicode.com/posts/1"  )
            .then( r => {  console.log( r ); })
    }
    return ( <>
        <button onClick={ 함수1 } > 함수1 </button>
        <button onClick={ 함수2 } > 함수2 </button>
        <button onClick={ (e)=>함수3( e , 3 ) } > 함수3 </button>
        <button onClick={ doGet } > doGet AXIOS </button>
        <button onClick={ doPost } > doPost AXIOS </button>
        <button onClick={ doPut } > doPut AXIOS </button>
        <button onClick={ doDelete } > doDelete AXIOS </button>
    </> )
}
```

tip] 리액트 AXIOS 통신

1. Axios는 브라우저, Node.js를 위한 Promise API를 활용하는 HTTP 비동기 통신 라이브러리

2. 특징
    1. 운영 환경에 따라 브라우저의 XMLHttpRequest 객체 또는 Node.js의 HTTP API 사용
    2. Promise(ES6) API 사용
    3. 요청과 응답 데이터의 변형
    4. HTTP 요청 취소 및 요청과 응답을 JSON 형태로 자동 변경

3. 설치

   https://www.npmjs.com/

   axios 검색

   npm i axios

4. impot 하기

   import axios from 'axios'

axios Content-Type

- 기본값 Content-Type : Application/json
- 첨부파일[폼] : Content-Type: multipart/formed-data
  { headers : {"Content-Type" : "multipart/formed-data" } }
- text형식 쿼리스트링[경로?키=값&키=값&키=값]
  { params : { bcno : 0 , page : 1 , key : "" , keyword:"" } }

```
axios.post( URL , [DATA] , [HEADER] )
    .then( response => { } )
    .catch( err => { } )
```

```
axios.post( URL , [DATA] , { headers: { 'Content-Type' : 'multipart/form-data' } } )
    .then( response => { } )
    .catch( err => { } )
```

```
axios.get( URL?쿼리스트링 , [DATA] , [HEADER] )
    .then( response => { } )
    .catch( err => { } )
```

```
axios.get( URL , { params : { 키=값 , 키=값 , 키 = 값 } } )
    .then( response => { } )
    .catch( err => { } )
```

```
axios.put( URL , JSON객체, [HEADER] )
    .then( response => { } )
    .catch( err => { } )
```

```
axios.delete( URL , [DATA] , [HEADER] )
    .then( response => { } )
    .catch( err => { } )
```

tip] CORS 정책이란.

리액트 서버 스프링 서버

localhost:3000 ---------- localhost:80 요청 ----------> localhost:80

<--------- localhost:3000 응답----------

서버는 CORS를 위반하더라도 정상적으로 응답을 해주고, 응답의 파기 여부는 브라우저가 결정한다

SOP(Same-Origin Policy)

SOP는 지난 2011년, RFC 6454에서 처음 등장한 보안 정책으로 말 그대로 “같은 출처에서만 리소스를 공유할 수 있다”라는 규칙을 가진 정책이다.

출처가 다른 두 개의 어플리케이션이 마음대로 소통할 수 있는 환경은 꽤 위험한 환경이다.

애초에 클라이언트 어플리케이션, 특히나 웹에서 돌아가는 클라이언트 어플리케이션은 사용자의 공격에 너무나도 취약하다.

악의를 가진 사용자가 소스 코드를 쓱 구경한 후 CSRF(Cross-Site Request Forgery)나 XSS(Cross-Site Scripting)와 같은 방법을 사용하여 여러분의 어플리케이션에서 코드가 실행된 것처럼 꾸며서 사용자의 정보를 탈취하기가 너무나도 쉬워진다.

CORS(Cross-origin resource sharing)

서버가 이 요청에 대한 응답을 할 때 응답 헤더의 Access-Control-Allow-Origin이라는 값에 “이 리소스를 접근하는 것이 허용된 출처”를 내려주고, 이후 응답을 받은 브라우저는 자신이 보냈던 요청의 Origin과 서버가 보내준 응답의 Access-Control-Allow-Origin을 비교해본 후 이 응답이 유효한 응답인지 아닌지를 결정한다.

tip] CORS 해결방안

1. 스프링 Rest 구현된 Controller 클래스에 아래 어노테이션 사용

@CrossOrigin("http://localhost:3000") // 교차 리소스 공유 [해당 주소 = 리액트서버]