

# React 정리

## Easy to create single page application

- speedy
- do not send html request

### 1. project 생성 및 run

- a. npx create-react-app "application 이름"
- b. index.js에서 DOM으로 render
- c. cd application
- d. npm run start - 프로젝트 launch
- e. node\_modules- all the dependencies
  - i. 삭제 했을 때는 npm install 로 다시 살리면 된다

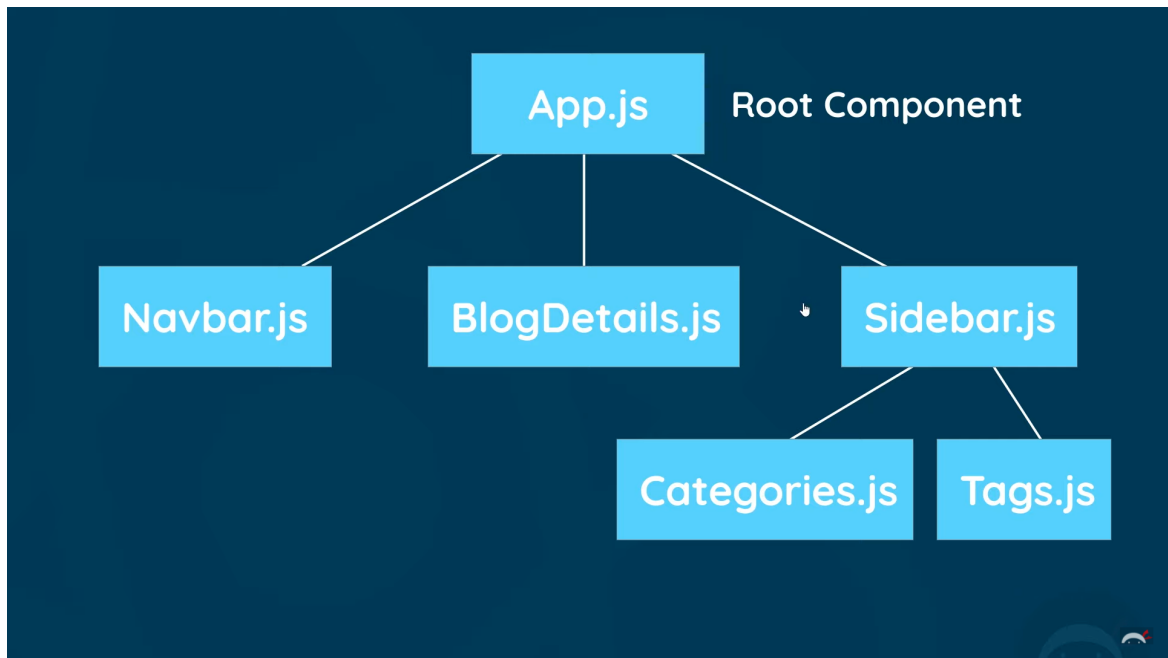
### 2. Components

- Template
- Logic

### 3. JSX ≠ HTML 하지만 거의 비슷한 template을 가지고 있다.

- a. 가장 큰 차이점은 className으로 class 이름을 추가해야한다
- b. emmet 설정을 하면 div.className을 적으면 바로 만들어진다.

### 4. 전체 구조



5. new Component → component.js 생성
  - a. App.js에 import component from './component'
  - b. div app 안에 해당 component 추가
6. CSS 적용
  - a. 큰 프로젝트는 css를 나누는 것이 효율적
  - b. 작은 프로젝트는 그냥 index.css안에 다 넣는 것이 효율적이다
7. inline style 적용시 { {} } 사용
  - a. 바깥 { } 는 dynamic value
  - b. 안쪽 {} 는 javascript object
8. button click event 와 같이 함수를 불러와야할 때
  - a. function 정의

```
const FunctionName = ()=>{implement}
```

- b. 사용

```
<button onClick = {handleClick}>Click me</button>
```

## 9. Using Events

- 함수에서 event 사용하기 위해서는 e를 추가하면 됨

```
<button onClick = {(e)=>handleClickAgain('Peter',e)}>Click Me again</button>
```

## 10. 변수를 선언하고 event 실행 시 변하게 하고 싶다면

- 단순히 함수 내에서 변수를 update한다고 template에서 변하지 않는다.
- useState Hook 를 사용해야한다.
  - import {useState} from 'react'
  - const [name, setName] = useState('mario'); → useState 안의 값은 default값

## 11. Chrome extention → React Developer Tools 사용

## 12. List 안의 값들을 표시하고 싶다면 javascript map 사용

항상 key를 사용해야한다. 그래야 id별로 구별할 수가 있다.

```
<div className="blog-preview" key={blog.id}>
  {blog.title}
</div>
```

## 13. 계속 반복될 것 같으면 reusable component 식으로 만들어 놓는게 좋다.

## 14. Props - parent component 에서 child component로 data를 전송하는 방법

- a. 더 reusable하게 사용할 수 있다.
- b. 다른 component의 data를 사용할 수 있다.

### 사용 방법

#### 1. props를 지정해 사용하는 방법

- parent component에서 넘길 props를 지정해 dynamic value로 넘긴다.

```
import BlogList from './BlogList';

const Home = () => {
  const [blogs, setBlogs] = useState([
    {title: 'my new website', body: 'Lorem ipsum...', author: 'mario', id:1},
    {title: 'Welcome Party', body: 'Lorem ipsum...', author: 'yoshi', id:2},
    {title: 'Web dev top tips', body: 'Lorem ipsum...', author: 'mario', id:3}
  ]);
  return (
    <div className="home">
      <BlogList blogs={blogs} />
    </div>
  );
}
```

- child component에서는 인자를 props로 설정하고 해당 props의 원하는 데이터를 지정해 사용한다.

```
import React from 'react';

const blogList = (props) => {
  const blogs = props.blogs;
  return (
    <div className="blog-list">
      {blogs.map((blog)=>(
        <div className="blog-preview" key={blog.id}>
          <h2>{blog.title}</h2>
          <p>Written by {blog.author}</p>
        </div>
      ))}
    </div>
  );
};

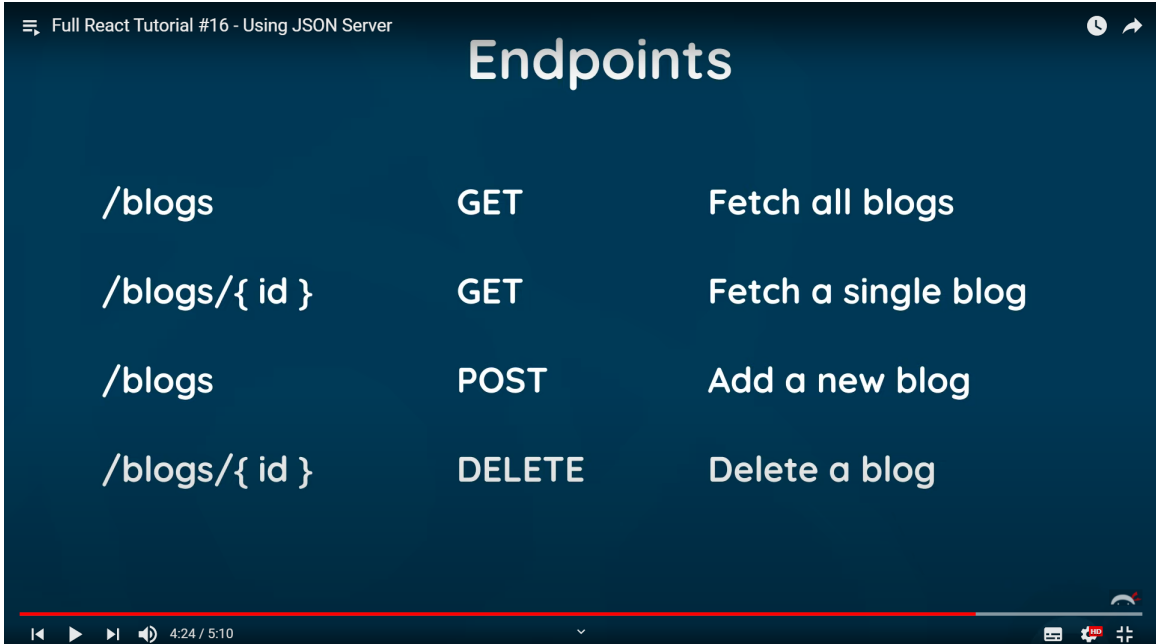
export default blogList;
```

## 2. props(text)를 사용하지 않고 바로 풀어서 사용하는 방법

- 넘겨 받을 data를 ( ) 안에 바로 풀어서 사용

```
const blogList = (blogs, title) => {
  // const blogs = props.blogs;
  // const title = props.title;
  return (
    <div className="blog-list">
      <h2>{title}</h2>
      {blogs.map((blog)=>(
        <div className="blog-preview" key={blog.id}>
          <h2>{blog.title}</h2>
          <p>Written by {blog.author}</p>
        </div>
      ))}
    </div>
  );
}
```

15. data에서 특정 data만 지정하고 싶은 경우 filter 사용
- a. `blogs={blogs.filter((blog)=>blog.author === 'mario')}`
16. useEffect Hook 는 매번 render 될 때 마다 필요한 사항에 사용하면 좋음
- a. `useEffect(()=>{`  
`},[]);` 를 사용하면 맨 처음 render될 때만 사용 가능하다.
  - b. `[]`안에 변수를 적으면 해당 변수가 변할 때만 useEffect가 실행된다.
17. Json Server
- a. `npx json-server --watch data/db.json --port 8000`
  - b. Endpoints



Full React Tutorial #16 - Using JSON Server

Endpoint	Method	Description
<code>/blogs</code>	GET	Fetch all blogs
<code>/blogs/{ id }</code>	GET	Fetch a single blog
<code>/blogs</code>	POST	Add a new blog
<code>/blogs/{ id }</code>	DELETE	Delete a blog

4:24 / 5:10

c. useEffect에서 해당 json 정보를 fetch해서 받아오고 그 후 setBlog를 통해 blog를 생성. 하지만 이렇게 된다면 fetch를 해오기 전에 blogs는 null 로 초기화 되어있기 때문에 이를 받아올 시간이 필요하다.

```
useEffect(()=>{
  fetch('http://localhost:8000/blogs')
    .then(res=>{
      return res.json()
    })
    .then((data)=>{
      console.log(data);
      setBlogs(data);
    });
},[]);
```

d. 이를 해결하는 방법은 애초에 template으로 넘겨줄 때 conditional statement를 통해 조건을 걸어주면 해결할 수 있다.

```
return (
  <div className="home">
    {blogs && <BlogList blogs={blogs} title="All blogs" handleDelete={handleDelete}/>}
    { /* blog가 존재할 때만 작동하게하는 방법 이게 없으면 현재 초기 값으로 blog가 null 이기 때문 */ }
  </div>
);
```

## 18. Conditional Loading Message

a. 위에서 한 data를 fetch하기 전에 Loading message 표시하는 방법

- i. 변수 생성
- ii. data가 fetch 될 때 해당 변수를 true/false로 변경
- iii. template에는 변수가 true일때만 message 표시
- iv. ex)

data fetch시 변수 조정

```
const [isPending, setIsPending] = useState(true);
useEffect(()=>{
  fetch('http://localhost:8000/blogs')
    .then(res=>{
      return res.json()
    })
    .then((data)=>{
      console.log(data);
      setBlogs(data);
      setIsPending(false);
    });
}, []);
```

isPending 변수가 true일 때만 해당 message div 표시

```
return (
  <div className="home">
    {isPending && <div> Loading...</div>}
    {blogs && <BlogList blogs={blogs} title=
      { /* blog가 존재할 때만 작동하게하는 방법 이
    </div>
  );
```

강의에서 나온 setTimeout 방식은 어떤식으로 돌아가고 로딩메세지가 나오는 것을 확인하기 위해 넣은 것이고 실제 어플리케이션에서는 사용자의 reponse time을 최소한 줄여야하

기 때문에 사용하면 안된다.

## 19. Handling Fetch Errors

a. Fetch error를 잡는 방법은 fetch 함수 다음에 catch 사용

```
.catch(err=>{  
  console.log(err.message);  
})
```

```
import {useState, useEffect} from 'react'  
import BlogList from './BlogList';  
const Home = () => {  
  const [blogs, setBlogs] = useState(null);  
  const [isPending, setIsPending] = useState(true);  
  const [error, setError] = useState(null);  
  
  useEffect(()=>{  
    setTimeout(()=>{  
      fetch('http://localhost:8000/blogs')  
        .then(res=>{  
          if(!res.ok){  
            throw Error('could not fetch the data for that resource');  
          }  
          return res.json()  
        })  
        .then((data)=>{  
          console.log(data);  
          setBlogs(data);  
          setIsPending(false);  
          setError(null);  
        })  
        .catch(err=>{  
          setIsPending(false);  
          setError(err.message);  
        })  
    }, 500);  
  }, []);  
  
  return (  
    <div className="home">  
      {error && <div>{error}</div>}  
      {isPending && <div> Loading...</div>}  
      {blogs && <BlogList blogs={blogs} title="All blogs" />}  
      {/* blog가 존재할 때만 작용하게하는 방법 이게 없으면 현재 초기 값으로 blog가 null 이기 때문 */}  
    </div>  
  );  
}  
  
export default Home;
```

## 20. Custom Hook 만들기 - 같은 에러 처리가 필요할 것 같을 때 사용하는 것이 좋다

a. src folder에 파일 생성

b. 사용할 Hook을 추가, 추후 재 사용을 위해 변수 명, endpoint url등을 global한 변수로 지정하는 것이 좋음

i. url

## ii. data

## iii. error 등등

```
const Home = () => {  
  const { data: blogs, isPending, error } = useFetch('http://localhost:8000/blogs');  
}  
// data: blogs는 data를 그대로 사용하고 다른코드에서 data를 사용해도 되지만  
// 특정 이름을 갖는 변수가 필요할 경우  
// 이런식으로 naming을 하면 더 보기 쉽게 사용할 수 있다.
```

```
import {useState, useEffect} from 'react';  
  
const useFetch = (url) => { // fetch endpoint url이 매번 같지 않기 때문에 url 인자를 받는다  
  const [data, setData] = useState(null);  
  const [isPending, setIsPending] = useState(true);  
  const [error, setError] = useState(null);  
  
  useEffect(()=>{  
    setTimeout(()=>{  
      fetch(url)  
        .then(res=>{  
          if(!res.ok){  
            throw Error('could not fetch the data for that resource');  
          }  
          return res.json()  
        })  
        .then(data=>{  
          console.log(data);  
          setData(data);  
          setIsPending(false);  
          setError(null);  
        })  
        .catch(err=>{  
          setIsPending(false);  
          setError(err.message);  
        })  
      }, 500);  
    }, [url]);  
  
    return {data, isPending, error}  
  })  
  
  export default useFetch;
```

## 21. React Router

### a. multiple pages를 넘겨주는 방법

### b. 일반적인 website

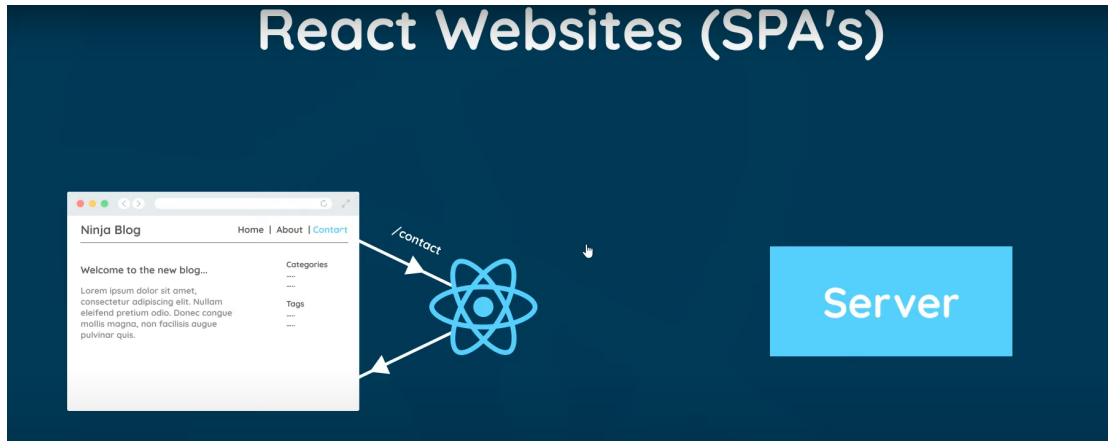
- server에 initial request를 보내면
- html로 request를 보낸다

### c. React website

- server에 initial request를 보내면



- js bundle을 포함해 request를 보낸다
- 이를 통해 react router로 연동한다
- 중간에 react가 개입해서 server와 직접 연동이 안됨



d. `npm install react-router-dom@5` 로 설치한다

- 버전 5를 사용하는 이유는 아직 6이 상용화가 안돼서

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

로 import 해서 사용해야한다

```
function App() {
  return (
    <Router>
      <div className="App">
        <Navbar />
        <div className="content">
          <Switch>
            <Route path="/">
              <Home />
            </Route>
          </Switch>
        </div>
      </div>
    </Router>
  );
}
```

## 22. Route

```
<Switch>
  <Route path="/">
    <Home />
  </Route>
  <Route path="/create">
```

```

    <Create />
  </Route>
</Switch>

```

이런 식으로 한다고 하더라도 react는 3000/ 를 읽고 맞다고 생각해서 Home을 반환하게 된다.

```

<Route exact path="/">
  <Home />
</Route>

```

- a. React는 Switch 안의 코드를 보면서 위에서 부터 차례로 Route를 확인한다
- b. 그 후 맞는 path를 찾아서 해당 component를 반환한다.

## 23. Router Links

- server 와 연동하지 않고 react 내에서 router를 변경하는 방법

```

import { Link } from 'react-router-dom'

<Link to="/">Home</Link>
<Link to="/create" >Create New Blog</Link>

// a -> Link 로 바꾸고 href 를 to로 바꾸면 된다.

```

## 24. useEffect Cleanup

- 이를 사용해서 계속 fetch하는 걸 막는다
- useFetch.js에서 useEffect 함수 안에

```

const abortCont = new AbortController();

fetch(url, { signal: abortCont.signal })
  .then(res=>{
    if(!res.ok){
      throw Error('could not fetch the data for that resource');
    }
    return res.json()
  })
  .then(data=>{
    console.log(data);
    setData(data);
    setIsPending(false);
    setError(null);
  })
  .catch(err=>{
    if(err.name === 'AbortError'){
      console.log('fetch aborted')
    }
    else{
      setIsPending(false);
      setError(err.message);
    }
  })

```

```
}, 500);  
return () => abortCont.abort();
```

fetch abort 에러일 때 (fetch를 다 하기 전에 페이지를 옮겨가거나 끊기는 일이 발생하는 경우 이런 방법으로 error 처리하면 된다)

## 25. Route Parameters

a. /blogs/123 /456 /678 처럼 뒤에 계속 변하는 route

```
<Route path="/blogs/:id">  
  <BlogDetails />  
</Route>
```

b. url이 가변적이라면 내용도 가변적이다

c. Hook을 사용해 다른 데이터를 가져와야한다

```
import { useParams } from "react-router";  
  
const { id } = useParams();  
//useParams를 사용하면 전 단계에서 넘긴 data를 받을 수 있다.
```

```
<Link to={`/blogs/${blog.id}`}>  
  <h2>{blog.title}</h2>  
  <p>Written by {blog.author}</p>  
</Link>  
  
// 해당 내용들을 Link로 바꿔서 to의 경로를 ${ } 안에 넣으면 된다
```

## 26. Reusing Custom Hooks

a. 해당 내용을 끌어올 component 안에서 useFetch - custom hook를 사용한다.

```
const { data: blog, error, isPending } = useFetch('http://localhost:8000/blogs/' + id);
```

b. 그 후 이전과 같이 fetch하기 전 loading message와 같은 설정을 해주면된다.

## 27. Forms 생성하기 - controlled inputs

a. form 생성할 때 각 tag 마다에 type required value onChange 사용 가능

i. type 은 input이나 textarea 와 같은 영역에서 사용

ii. required는 필수 입력 항목이다

iii. value = { } 로 해당 값을 관리할 수 있다.

iv. onChange는 해당 tag의 내용이 변할 때 확인 할 수 있는 함수

```
const [title, setTitle] = useState('');
const [body, setBody] = useState('');
const [author, setAuthor] = useState('mario');

<input
  type="text"
  required
  value={title}
  onChange={(e)=>setTitle(e.target.value)}
/>
```

## 28. Submit Event

```
const handleSubmit = (e)=>{
  e.preventDefault(); //page reload 방지
  const blog = {title, body, author};
  console.log(blog);
}

<form onSubmit={handleSubmit}>
  <button>Add blog</button>
</form>
```

## 29. 해당 submit data를 json server로 POST Request 보내는 방법

```
const [isPending, setIsPending] = useState(false);

const handleSubmit = (e)=>{
  e.preventDefault();
  const blog = {title, body, author};

  setIsPending(true);

  fetch('http://localhost:8000/blogs', {
    method: 'POST',
    headers: {"Content-Type": "application/json"},
    body: JSON.stringify(blog)
  }).then(()=>{
    console.log('new blog added');
    setIsPending(false);
  })

  //fetch할 url 과 post method, body 작성
}
```

## 30. form 에서 POST Request 전송 후 redirect하는법

### a. useHistory로 이전 페이지로 돌아가면 된다.

```
const history = useHistory();

submit 관련 함수 내에서
1. history.go(-1) --> - 하는 숫자만큼 브라우저 히스토리를 보고 그만큼 이동
```

2. history.push('/') --> 원하는 url로 이동 가능  
--> 특정 url로 이동하고 싶으면 push 사용 / 단순 전페이지는 go 사용이 유리

### 31. Delete button

```
const history = useHistory();

const handleClick={() => {
  fetch('http://localhost:8000/blogs/' + blog.id, {
    method: 'DELETE'
  }).then(() => {
    history.push('/')
  })
}

<button onClick={handleClick}>Delete</button>
```

method: 'DELETE' 사용하면 바로 삭제 가능

### 32. 404 페이지

```
<Route path="*">
  <NotFound/>
</Route>
```

기존의 path가 아닌 없는 페이지에 접근하려고 하면 \* 를 통해 잡으면 된다.  
그 후 해당 페이지 component 생성해서 추가만 하면 된다.