**node.js 설치**

node -v로 확인

**npm 설치**

npm으로 확인

**npx 설치**

npm install npx -g **=> npx -v로 확인**

**git 설치**

git -v로 확인

**이전에는 create를 위한 절차가 있었으나**

**현재는 npx를 통해 간소화되었다.**

npx create-react-app **[앱 이름]**

**npm start**

Starts the development server.

**npm run build**

Bundles the app into static files for production.

**npm test**

Starts the test runner.

**npm run eject**

Removes this tool and copies build dependencies, configuration files

and scripts into the app directory. If you do this, you can’t go back!

We suggest that you begin by typing:

**cd movie\_app\_2019**

**\*\* npm start \*\***

**code movie\_app\_2019 =>** 이를 통해 바로 vscode로 연결 됨

Compiled successfully!

You can now view movie\_app\_2019 in the browser.

Local: **http://localhost:3000/**

On Your Network: **http://192.168.56.1:3000/**

Note that the development build is not optimized.

To create a production build, use **npm run build**.

**< git 연결을 해보자 >**

**git init**

Reinitialized existing Git repository in C:/Users/admin/Desktop/reactJS/movie\_app\_2019/.git/

**git remote add origin [ git주소 ]**

https://github.com/co24428/movie\_app\_2019.git

**git add .**

**git commit -m [ “commit message” ]**

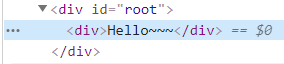
**git push origin master**

**<< \*\* 어떻게 react가 작동되는가 \*\* >>**

**JSX = JavaScript 안의 HTML -> 위와 같이 쓰는 방식, 동적 application에 좋다!**

<div id="root"></div>

public/index.html에 보면 root에 비어있다.



실제 페이지에서 검사를 해보면 root 아래에 Hello가 담긴 div가 있음을 알 수 있다.

react는 본인이 거기에서 쓰는 모든 요소를 생성한다.

JavaScript와 함께 그것들을 만들고 그것들을 HTML로 밀어 넣는다.

react는 index.html에 element를 넣는 역할을 한다.

**그래서! 모든 react application을 div 사이에 넣는다!!**

**src/index.js**

import App from './App';

ReactDOM.render(<App />, document.getElementById('root'));

**DOM = Document Object Model**

**App이라는 컴포넌트**에 root라는 id에 넣는다. **So) react가 빠르다!**

**public/index.html**에서 root를 바꾸면, **index.js**에서도 값을 바꿔줘야 한다.

즉, 서로 매칭되게 해주어야 한다.

빈 HTML을 로드하고 난 후에 react가 HTML을 밀어 넣는다. component ID에 따라서

react는 component와 동작한다. 모든 것은 component이다!!

component는 HTML을 반환하는 함수이다.

**형태 : <[이름] />**

react application이 **하나의 component만을 rendering**해야하기 때문에!

ReactDOM.render(<App /><Potato />, document.getElementById('root'));

<App /><Potato />

위 코드는 에러가 난다! -> **그러면 Potato를 App에 넣어야 한다.**

cf) .”./” 인 이유 – 같은 디렉토리이기 때문에

import React from 'react';

import Potato from './Potato';

function App() {

  return (

    <div>

      <h1>Hello~~~</h1>

      <Potato />

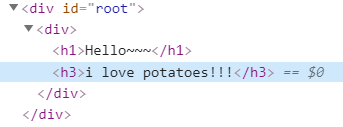
    </div>

  );

}

export default App;





위처럼 나오게 되고, 결과는 **평범한 HTML**처럼 사용된다.

import Potato from './Potato';

위 코드를 지우고,

function Potato() {

  return <h1>I like Potato</h1>

}

위 코드를 App.js에 넣으면 똑같이 작동된다.

* **재사용 가능한 component를 만들 수 있다.**

      <Food fav="kimchi" />

**파라미터 주는 방식**

위처럼 하면 **Food라는 component**에 **kimchi라는 value**로 **prop name**을 줬다.

      <Food fav="kimchi” somthing = {true} />

위처럼 여러 개 놓을 수도 있다.

function Food( {fav} ) {

  return <h1>I like {fav}</h1>

}

**파라미터를 받는 방식**

**JSX = HTML + JavaScript**

**<< \*\* <Food />를 동적으로 할당하는 법 -> 복사붙여넣기가 아닌 추가법\*\* >>**

**function component를 동적으로!**

**foodILIKE**를 리스트로 만듦.

**foodILIKE**를 Food에 rendering해주어야 한다. -> map함수

friends.map(function(current){

console.log(current);

})

현재 변수의 각각 item을 current에 넣으면서 function을 실행해라

– python의 for문 list로 돌리는 것처럼 쓴다.

friends.map(function(friend){

return friend + "★";

})

(3) ["1★", "2★", "hwan★"]

**<< \*\* react 코드 적용 \*\* >> 추가로 이미지 출력 HTML까지**

function Food( {name, picture} ) {

  return <div>

    <h2>I like {name}</h2>

    <img src={picture} />

  </div>

}

**HTML <img>** 태그를 통해 picture의 주소를 들어가서 이미지를 가져와서 웹에 띄운다.

<Food name="kimchi" />

      {foodILike.map(dish => (

        <Food name={dish.name} picture= {dish.image} />

      ))}

**dish는 각각의 아이템을 받은 객체(object)이다. 즉, 뭐든 들어갈 수 있다.**

또 Food의 파라미터를 2개로 했기 때문에 각각 잡아준다.

**\*\* ,(콤마) 없어야 된다. \*\***