**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” ]**

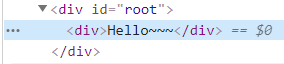
**git push origin master**

**<< \*\* 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 />

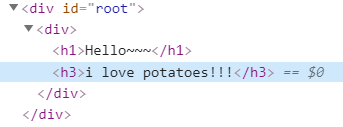
    </div>

  );

}

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. \*\***

renderFood라는 **function을 하나 더 만들어서 함수화**

function renderFood(dish){

  console.log(dish);

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

}

<div> </div> 내부에 넣으면 **위의 코드처럼 돌아가고** console에 로그를 출력한다.

      {console.log(foodILike.map(renderFood))}

      {foodILike.map(renderFood)}

**-> 추천하는 방법은 아니다. 기존의 것으로 하자**

또한 console에 뜨는 에러는 **list의 고유적인 키가 없기 때문**이다.

**foodILike**에 **id라는 고유번호**를 주고

      {foodILike.map(dish => (

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

      ))}

위처럼 key로 dish.id를 주면 사라진다.

**prop-types**를 설치 - 전달받은 props가 내가 원하는 props인지 확인

- 필수조건(isRequired), 원하는 타입 여부 등 확인할 수 있다.

**npm install prop-types -> 경로는 해당 프로젝트 내부에서!!**

**package.json의 dependencies**에서 확인 가능

    "prop-types": "^15.7.2",

**App.js**에 import

import PropTypes from "prop-types"

맨 처음 들여쓰기에 작성 **– Food의 타입과 필수여부를 지정, 안맞을 시 콘솔에 에러 출력**

Food.propTypes = {

  name: PropTypes.string.isRequired,

  picture: PropTypes.string.isRequired,

  rating: PropTypes.number

};

rating의 경우 string으로 주면 **원하는 타입이 아니라고** 뱉어낸다. 추가로 비워도 괜찬다.

// Food를 이용한 예제, PropTypes까지만 사용

import React from 'react';

import PropTypes from "prop-types";

const foodILike = [

  {

    id:1,

    name: "Kimchi",

    image:

      "https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fi.ytimg.com%2Fvi%2FeTucCw1w6Ak%2Fmaxresdefault.jpg",

    rating: 5

  },

  {

    id:2,

    name: "Kimbap",

    image:

      "https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fupload.wikimedia.org%2Fwikipedia%2Fcommons%2Fthumb%2F5%2F56%2FVegetable\_gimbap.jpg%2F1200px-Vegetable\_gimbap.jpg",

      rating: 4.2

  },

  {

    id:3,

    name: "Shusi",

    image:

      "https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fupload.wikimedia.org%2Fwikipedia%2Fcommons%2Fthumb%2F6%2F60%2FSushi\_platter.jpg%2F1200px-Sushi\_platter.jpg",

      rating: 3.7

  },

  {

    id:4,

    name: "Ramen",

    image:

      "https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fi.ytimg.com%2Fvi%2FB8y3SSmz4sg%2Fmaxresdefault.jpg",

      rating: 4.9

  }

]

// function renderFood(dish){

//   console.log(dish);

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

// }

function Food( {name, picture, rating} ) {

  return <div>

    <h2>I like {name}</h2>

    <h4>{rating}/5.0</h4>

    <img src={picture} alt={name}/>

  </div>

}

Food.propTypes = {

  name: PropTypes.string.isRequired,

  picture: PropTypes.string.isRequired,

  rating: PropTypes.number

};

function App() {

  return (

    <div>

      <h1>Hello~~~</h1>

      {/\* <Food name="kimchi" /> \*/}

      {/\* console error를 잡기 위한 key값이 필요함.

          그래서 id라는 고유키를 설정하고 key 변수로 준다. \*/}

      {foodILike.map(dish => (

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

      ))}

      {/\* {console.log(foodILike.map(renderFood))}

      {foodILike.map(renderFood)} \*/}

    </div>

  );

}

export default App;

동적 데이터 ( dynamic data ) – 실시간으로 변하는 데이터

**function component -> class component**

**새롭게 처음부터!**

import React from 'react';

import PropTypes from "prop-types";

export default App;

class App extends React.Component{

  render(){

    return <h1>Im a class component</h1>

  }

}

**class component는 React component에서 확장된다.**

기본적으로 React component는 리턴이 없다.

**render method를 필수로 넣어주어야 하고,**

**react는 class component의 render method를 자동으로 실행한다.**

class component를 쓰는 이유 **-> state**

state = object / data를 넣을 공간이면서 가변적이다.

    return <h1>The number is {this.state.count}</h1>

사용법은 위의 {this.state.count}

JS를 통해 함수로 기능을 넣을 때

  add =() => {

    console.log("add")

  };

**add() -> 매번 실행 / add -> 누를 때만 실행 ( 이해는 안되지만 후자만 쓰자 )**

      <button onClick={this.add}>Add</button>

**<< 이 state 값을 변경하려면? >>**

this.state.count = 1

이렇게 하면 경고가 뜬다. **직접 state를 바꾸지 말것! -> renderfunctiondmf refresh하지 않는다 자동으로**

    this.setState({ count: this.state.count+1});

이렇게 해주어야 한다. -> **자동으로 새롭게 render한다**

**react는 바뀌는 부분만 딱 바꾸기 때문에 다른 것보다 빠르다 (virtual DOM) – check inspect**

    this.setState(current => ({count: current.count+1}));

이 것이 좀 더 고급표현 **– 외부에 덜 의존적이다.**

**setState를 할 때마다, react는 새로운 state로 다시 render한다. 바뀌는 부분만!**

**직접 변환은 의미가 없다.**

life cycle -> component를 생성하고 없애는 과정(방법)

**Mounting**

**constructor() => render** 이전에 수행되는 **생성자**

getDerivedStateFromProps() => 몰라도 됨

**render()**

**componentDidMount() => 처음 component를 render 할 때**

**Updating**

getDerivedStateFromProps() => 몰라도 됨

shouldComponentUpdate() => 업데이트를 할지 말지 결정하는 것

**render()**

getSnapshotBeforeUpdate() =>

**componentDidUpdate()** => 업데이트 되었는지

**Unmounting**

**componentWillUnmount() => component가 사라질 때, 페이지를 옮기거나 등등**

// add, minus를 통한 component 예시

import React from 'react';

import PropTypes from "prop-types";

class App extends React.Component{

  state ={

    count: 0

  };

  add =() => {

    console.log("add")

    this.setState(current => ({count: current.count+1}));

  };

  minus =() => {

    console.log("minus")

    this.setState(current => ({count: current.count-1}));

  };

  render(){

    // return <h1>The number is {this.state.count}</h1>

    return <div>

      <h1>The number is {this.state.count}</h1>

      <button onClick={this.add}>Add</button>

      <button onClick={this.minus}>minus</button>

    </div>

  };

}

export default App;

끝