



Vanilla JS - HTML 만들고 JS로 가져와서 작업

React JS - 모든것이 JS로서 시작 becomes HTML
결과물인 HTML update 할수 있음
(유저에게 보여줄 내용을 컨트롤 할수 있다.)

React DOM - React Element를 가져다가 HTML로 바꿈 (HTML에 대한 역할)

Vanilla JS

```
<script>
let counter = 0;
const button = document.getElementById("btn");
const span = document.querySelector("span");
function handleClick() {
  counter = counter + 1;
  span.innerText = `Total clicks: ${counter}`;
}
button.addEventListener("click", handleClick);
</script>
```

React JS

```
<script>
const root = document.getElementById("root");
const span = React.createElement(
  "span",
  { id: "sexy-span", style: { color: "red" } },
  "Hello I'm a span"
);
ReactDOM.render(span, root); → 어디에 무엇을 rendering 할건지
</script>
```

⇒ HTML element
React.createElement
HTML 만들기

```
const h3 = React.createElement(
  "h3",           You, seconds ago • Uncommitted changes
  {               addEventListener
    onMouseEnter: () => console.log("mouse enter"),
  },
  "Hello I'm a span"
);
```

⇒ createElement를 통해
element의 종류, event 처리, 내용 등 다
한번에!

⇒ but JSX를 사용할 수 있기 때문에
이러한 방식은 쓰지 않는다.

JSX - JAVA SCript를 확장한 문법이다. HTML 문법과 흡사한 문법으로
react element를 생성할 수 있다. (Babel을 통하여 JSX code를 browser가
이해할 수 있게 바꿈)

```
const root = document.getElementById("root");
const Title = () => (
  <h3 id="title" onMouseEnter={() => console.log("mouse enter")}>
    Hello I'm a title
  </h3>
);
const button = () => [
  <button - TAG
    style={{ backgroundColor: "tomato" }}
    onClick={() => console.log("im clicked")}
  >
    Click me! - Content
  </button>
];
const container = React.createElement("div", null, [Title, button]);
ReactDOM.render(container, root);
```

↳ 이런 내용처럼!

↓
Rendering 수행

```
const root = document.getElementById("root");
const Title = () => (
  <h3 id="title" onMouseEnter={() => console.log("mouse enter")}>
    Hello I'm a title
  </h3>
);
const Button = () => (
  <button
    style={{ backgroundColor: "tomato" }}
    onClick={() => console.log("im clicked")}
  >
    Click me!
  </button>
);
const Container = (
  <div>
    <Title />
    <Button />
  </div> — html
);
ReactDOM.render(Container, root);
```

arrow function
function Title() {
 return (
 <h3>
);
}

* ↳ 렌더링할 component 사용 (적용하는 반드시 대상)
[내가 직접 생성한 element
는 반드시 이와 같이 사용

Babel에서 변환된 code

```
var Button = function Button() {
  return /*#___PURE__*/React.createElement("button", {
    style: {
      backgroundColor: "tomato"
    },
    onClick: function onClick() {
      return console.log("im clicked");
    }
  }, "Click me!");
};

var Container = /*#___PURE__*/React.createElement("div", null, /*#___PURE__*/React.createElement>Title, null), /*#___PURE__*/React.createElement(Button, null));
ReactDOM.render(Container, root);
```

STATE - 데이터가 저장되는 곳

Counter Up code

```
const root = document.getElementById("root");
let counter = 0;
function countUp() {
  counter = counter + 1;
  render(); → count up 할 때마다 히더링됨
}
function render() {
  ReactDOM.render(<Container />, root); → 엔터링함
}
const Container = () => (
  <div>
    <h3>Total clicks : {counter}</h3>
    <button onClick={countUp}>Click me </button>
  </div>
);
render();
```

⇒ countUp 할 때마다 Container 전부가 reRendering 될 것 같지만 그렇지 않고 update 된 부분만 reRendering 된다. - React의 강력함.

↓
더 효율적인 방법이 존재함

ReactState 사용해 React 컴포넌트 안에서 데이터 바꾸기

console.log [현재상태, modifier]

```
const root = document.getElementById("root");
function App() {
    const [counter, setCounter] = React.useState(0);
    const onClick = () => {
        setCounter(counter + 1); // 초기값 설정
    };
    return [
        <div>
            <h3>Total clicks : {counter}</h3>
            <button onClick={onClick}>Click me </button>
        </div>
    ];
}
ReactDOM.render(<App />, root);
```

(counter = React.useState())
setCounter = () =>

modifier 함수를 통해 state(App의 data)를 바꿀 때

새로운 값으로 component를 reRender하고 (코드 재실행) React는 reRendering 시
바꾸는 값만 rendering함

Change State → reRendering

```
const root = document.getElementById("root");
function App() {
    const [counter, setCounter] = React.useState(0);
    const onClick = () => {
        // setCounter(counter + 1); → 직접 넣어서 update
        setCounter((current) => current + 1); // 파라미터로 함수를 넣어주면
    };
    return [
        <div>
            <h3>Total clicks : {counter}</h3>
            <button onClick={onClick}>Click me </button>
        </div>
    ];
}
ReactDOM.render(<App />, root);
```

첫번째 인자에 현재값을 넣어준다
(함수형 update 표현식)

단위 변환 App 마우리

```
const root = document.getElementById("root");
function App() {
  const [amount, setAmount] = React.useState();
  const [flipped, setFlipped] = React.useState(false);
  const onChange = (event) => {
    setAmount(event.target.value);
  };
  const reset = () => setAmount("");
  const onFlip = () => {
    reset();
    setFlipped(!flipped);
  };
  return (
    <div>
      <div>
        <h1>Super Converter</h1>
        <label htmlFor="minutes">Minutes</label>
        <input value={flipped ? amount * 60 : amount} id="minutes" type="number" placeholder="Minutes" onChange={onChange} disabled={flipped}>
      </div>
      <div>
        <label htmlFor="hours">Hours</label>
        <input value={flipped ? amount : amount / 60} id="hours" type="number" placeholder="Hours" onChange={onChange} disabled={!flipped}>
      </div>
      <button onClick={reset}>Reset</button>
      <button onClick={onFlip}>Flip</button>
    </div>
  );
}
ReactDOM.render(<App />, root);
</script>
</html>
```

컴포넌트 렌더링

state의 값 변환으로 만들기 (reset)

→ `!flipped`은 `flipped`의 state를 반대로 바꿈 (True or False)

현재 `flipped`의 State

JS의 For와 증복도 있음 JSX에서는 htmlFor 사용

value 값의 조건 true이면 Amount x 60 false이면 amount

조건 (삼항) 연산자

조건 연산자는 JavaScript에서 세 개의 피연산자를 받는 유일한 연산자입니다. 조건 연산자는 주어진 조건에 따라 두 값 중 하나를 반환하여, 구문은 다음과 같습니다.

condition ? val1 : val2;

만약 condition 이 참이라면, 조건 연산자는 val1 을 반환하고, 그 외에는 val2 를 반환합니다. 다른 연산자를 사용 할 수 있는 곳이라면 조건 연산자도 사용할 수 있습니다.

. 두 개의 input 중 enable 상태인

input에서 (flipped 이용)

amount state를 수정할 수 있다.

(둘다 onChange 헤게)

disabled

↳ `flipped`가 `false` 일 때

Divide and conquer

```
function App() {  
  const [index, setIndex] = React.useState("0");  
  const onSelect = (event) => {  
    setIndex(event.target.value);  
  };  
  return (  
    <div>  
      <h1>Super Converter</h1>  
      <select value={index} onChange={onSelect}>  
        <option value="0">Minutes & Hours</option>  
        <option value="1">Km & Miles</option>  
      </select>  
      {index === "0" ? <MinutesToHours /> : null}  
      {index === "1" ? <KmToMiles /> : null}  
    </div>  
  );  
}  
ReactDOM.render(<App />, root);
```

→ select data state

↳ 부모 component

→ select event가 일어난 부모의 value값

⇒ index state 변경

select

] index 번호 비교연산후 rendering부록
결정

PROPS

- 부모 컴포넌트로부터 자식 컴포넌트에 데이터를 보낼 수 있게 해주는 방법

Props 사용의 예

```
<script type="text/babel">
function Btn({ banana, big }) {
  console.log(big);
  return (
    <button
      style={{
        backgroundColor: "tomato",
        color: "white",
        padding: "10px 20px",
        border: 0,
        borderRadius: 10,
        fontSize: big ? 18 : 16,
      }}
    >
      {banana}
    </button>
  );
}

function App() {
  return (
    <div>
      <Btn banana="Save changes" big={false} />
      <Btn banana="Continue" big={true} />
    </div>
  );
}

const root = document.getElementById("root");
ReactDOM.render(<App />, root);
</script>
```

Btn의 첫번째 인자를 console.log 확인하면
Object로 Data가 넘어간 것을 확인 할 수 있다.

```
> {banana: 'Save changes', big: false}
> {banana: 'Continue', big: true}
```

• 여기에서 component
⇒ JSX를 return하는 함수

Btn(props)는
props.banana 형식이거나
} props는 Object가 때문에 shortcut 가능
(object의 property를 꺼내는 형식)

해당구문은 Btm()을 불러오고
Data를 인자로 넣어 rendering ⇒ React.

custom Component에 속성을 추가하는 구문을 넣어
부모 컴포넌트에서의 Data를 설정 할 수 있고
자식 컴포넌트에서 재사용 할 수 있다.

<Btn banana="Save changes" />
↳ Btm({banana: "Save changes"})

⇒ 어떠한 prop이든 Btn 컴포넌트로 보내면
그것들은 Btn 함수의 첫번째 인자로 들어간다.

무언가를 빼기 때문에 추가

React.Memo - 지금 당장은 끌릴이 없겠지만 언젠가 최적화의 필요성을 느끼면 쓸려해 기억해두자

```
<script type="text/babel">
function Btn({ text, changeValue }) {
  console.log(text, "was rendered");
  return (
    <button
      onClick={changeValue} _ onclick 이번트 발생시 changeValue 함수 실행되어
      style={{ text의 State가 변동될
        backgroundColor: "tomato",
        color: "white",
        padding: "10px 20px",
        border: 0,
        borderRadius: 10,
      }}
    >
      {text}
    </button>
  );
}

const MemorizedBtn = React.memo(Btn); // memorized version의 Btn 만들기
function App() {
  const [value, setValue] = React.useState("Save changes");
  const changeValue = () => setValue("Revert Changes");
  return [
    <div>
      <MemorizedBtn text={value} changeValue={changeValue} />
      <MemorizedBtn text="Continue" />
    </div>
  ];
}

const root = document.getElementById("root");
ReactDOM.render(<App />, root);
</script>
```

↳ props가 변경되지 않으면 해당 component는
re-render하지 않고 마지막으로 rendering된
결과를 재사용한다.

→ props로 함수 또한 넣을 수 있다.

State 변경 => component Re-rendering

memo 사용 x (State가 변경되었기 때문에
App component 자체를 re-rendering)

```
Save changes was rendered
Continue was rendered
Revert Changes was rendered
Continue was rendered
```

⇒

memo 사용

```
Save changes was rendered
Continue was rendered
Revert Changes was rendered
```

PropTypes

=> props의 Type을 검사해 설정한 Type과 다르다면 경고를 띠는다.

```
<script type="text/babel">
  function Btn({ text, fontSize }) {
    return (
      <button
        style={{
          backgroundColor: "tomato",
          color: "white",
          padding: "10px 20px",
          border: 0,
          borderRadius: 10,
          fontSize: fontSize,
        }}
      >
        {text}
      </button>
    );
  }
  Btn.propTypes = {
    text: PropTypes.string.isRequired,
    fontSize: PropTypes.number,
  };
  function App() {
    return (
      <div>
        <Btn text="Save Changes" fontSize={18} />
        <Btn fontSize="halo" />
      </div>
    );
  }
  const root = document.getElementById("root");
  ReactDOM.render(<App />, root);
</script>
</html>
```

fontSize = 모로 default 값 설정 가능
(전달받지 못한 경우 default로 설정됨)

JavaScript

필수 prop으로 지정

] type 지정

```
✖ Warning: Failed prop type: The prop `text` is marked as required in `Btn`, but its value is `undefined`.
  at Btn (<anonymous>:4:19)
✖ Warning: Failed prop type: Invalid prop `fontSize` of type `string` supplied to `Btn`, expected `number`.
  at Btn (<anonymous>:4:19)
```

Create React App

React Application을 만드는 최고의 방식

많은 script와 사전 설정들을 관리해줌

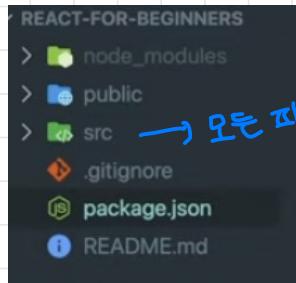
Node.js 를 설치 후 사용 가능

Terminal에서 npx create-react-app 파일명

⇒ react project 생성됨

Code terminal에서 npm start로 development server 만들어짐

⇒ 브라우저 열고 자동으로 실행



Node.js로 작업하기 때문에 파일들을 각각 분리하기 가능, 조직적으로 구성

한 파일당 한 컴포넌트, index.js에서 컴포넌트들을 import

⇒ divide and conquer

간단한 기능 및 사용

```
import Button from "./Button";
import styles from "./App.module.css";

function App() {
  return (
    <div>
      <h1 className={styles.title}>Welcom!</h1>
      <Button text={"hello!"} />
    </div>
  );
}

export default App;
```

css 모듈에 있는 클래스명 사용
다른 css 소스에 같은 클래스명을
사용하더라도 React에서 Random하게
클래스명을 붙여주어 문제X

```
<h1 class="App_title__YS7Xx">Welcom!</h1>
```

```
✓ import PropTypes from "prop-types"; PropTypes import
  import styles from "./Button.module.css";
✓ function Button({ text }) {
  ✓ return (
    ✓ <div>
      ✓ <button className={styles.btn}>{text}</button>;
    </div>
  );
}
✓ Button.propTypes = {
  ✓ text: PropTypes.string.isRequired,
};

export default Button;
```

```
App.module.css > .title
.title {
  element class="title" -apple-system, BlinkMacSystemFont, "Segoe
  Selector Specificity: (0, 1, 0)
}

.btn {
  color: white;
  background-color: red;
}
```

css를 모듈화 시켜

각 컴포넌트에 import하여 사용

⇒ 컴포넌트마다 Style 적용 가능

(index에 style.css를 Import하여 사용할 경우 CSS파일이 페이지의 모든 것에 적용됨.
여기까지 컴포넌트 style이 하나의 파일에 복잡해
되고)

⇒ CRA를 통해 컴포넌트를 분리해서 만들 수 있고
그 컴포넌트로 위한 CSS를 만들 수 있다.

Effects

State가 변화하면 컴포넌트의 모든 부분이 다시 실행됨

⇒ 맨 처음 render 될때만 실행시키고 싶은 부분이 있다면?
⇒ Component 안의 특정 data가 변화할때 실행시키고 싶은 부분?

```
import { useState, useEffect } from "react";
```

```
function App() {
  const [counter, setValue] = useState(0);
  const [keyword, setKeyword] = useState("");
  const onClick = () => setValue((prev) => prev + 1);
  const onChange = (event) => setKeyword(event.target.value);
  useEffect(() => {
    console.log("I run only once.");
  }, []); dependence 비어있으면 처음 한번만 실행
  useEffect(() => {
    console.log("I run when 'keyword' changes.");
  }, [keyword]); keyword State가 변화하면 실행
  useEffect(() => {
    console.log("I run when 'counter' changes.");
  }, [counter]); counter State가 변화하면 실행
  return (
    <div>
      <input
        value={keyword}
        onChange={onChange}
        type="text"
        placeholder="Search here...">
      />
      <h1>{counter}</h1>
      <button onClick={onClick}>Click me</button>
    </div>
  );
}
```

→ (RA를 통해 만들면서 React.useState을 useState로 줄여쓸수 있음)

State 사용

useEffect (deps, dependency)

React에서 변화를
지켜보기 하는 것.
변화할 때 코드 실행
여러개 넣을 수 있음

(Clean Up - component가 destroy 될 때 실행하기).

```
import { useState, useEffect } from "react";

function Hello() {
  useEffect(() => {
    console.log("hi :)");
    return () => console.log("bye :(");
  }, []);
  return <h1>Hello</h1>;
}

function App() {
  const [showing, setShowing] = useState(false);
  const onClick = () => setShowing((prev) => !prev);
  return (
    <div>
      {showing ? <Hello /> : null}
      <button onClick={onClick}>{showing ? "Hide" : "Show"}</button>
    </div>
  );
}

export default App;
```

effect 의 Clean Up 대처법

useEffect 는 dependency 가 변화할 때 해제할 수 있도록

(dependency 가 빠져있다면 처음에만 호출)

함수의 return 값은 Clean Up 대처법에 이용됨

Clean-up은 컴포넌트가 마운트 해제될 때 발생된다.

React 3 todo list 만들기 (연습)

```
import { useState } from "react";
```

```
function App() {
  const [todo, setTodo] = useState(""); → 입력 value state
  const [todos, setTodos] = useState([]); → todo list의 state
  const onChange = (event) => setTodo(event.target.value);
  const onSubmit = (event) =>
```

```
  event.preventDefault();
```

```
  if (todo === "") {
```

```
    return;
```

```
  }
```

```
  setTodo(""); → 입력 value 바꾸기
```

```
  setTodos([currentArray] => [todo, ...currentArray]);
```

```
); → todo list를 current 값 + todo로 이뤄진 Array로 바꿈 ( modifier 를 이용해
```

```
return (
```

```
<div>
```

```
  <h1>My Todos({todos.length})</h1>
```

```
  <form onSubmit={onSubmit}> → Submit event 처리
```

```
    <input
```

```
      value={todo} input value 설정
```

```
      type="text"
```

```
      placeholder="Write your to do..."
```

```
      onChange={onChange} onChange event 처리.
```

```
    />
```

```
    <button>Add To Do</button>
```

```
  </form>
```

```
  <hr />
```

```
  <ul>
```

```
    {todos.map((item, index) => (
```

```
      <li key={index}>{item}</li>
```

```
    ))}
```

```
  </ul>     index 번호를 item 텍스트
```

```
  </div>
```

```
  Key 사용
```

```
)
```

```
}
```

```
Key는 React가 어떤 항목을 변경, 추가 또는 삭제할지 식별하는 것을 돕습니다. key는 엘리먼트에 일정적인 고유성을 부여하기 위해 배열 내부의 엘리먼트에 지정해야 합니다.
```

```
export default App;
```

입력 value state

의 state

입력 value state 변화 (change event)

Submit 이벤트 발생 시 default하고

값 변경 시, 새로운 값이
넣어짐

event 처리

todo list를 전달하는 구문

map() 함수 ← 각 번째 Argument 값을 바로 함수식

내부의 값을
바꿔 새로운
배열로 반환

부관 Argument index 번호

Coin tracker 연습

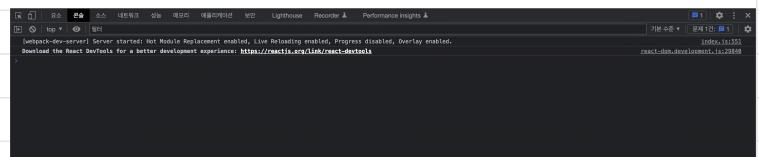
```
function App() {
  const [loading, setLoading] = useState(true);
  const [coins, setCoins] = useState([]);
  useEffect(() => {
    fetch("https://api.coinpaprika.com/v1/tickers") => 코인 정보 API 불러오기
      .then((response) => response.json()) → 네트워크에서 JSON 파일 가져오기
      .then((json) => {
        setCoins(json);
        setLoading(false); } coins 배열 => [JSON data]
        console.log(coins);
      });
  }, []);
  return (
    <div>
      <h1>The Coins!{loading ? "" : `(${coins.length})`}</h1>
      {loading ? (
        <strong>Loading...</strong> ) : ( 코인 정보 개수 (Index 2)
        <select> HTML <select> 요소는 옵션 메뉴를 제공하는 컨트롤을 나타냅니다.
          {coins.map((coin) => (
            <option key={coin.id}> 부여되어야 하는 id를 key값으로 사용
              {coin.name} {coin.symbol} : {coin.quotes.USD.price} USD
            </option> 이름    상호    가격
          )));
        </select>
      )}
    </div>
  );
}

export default App;
```

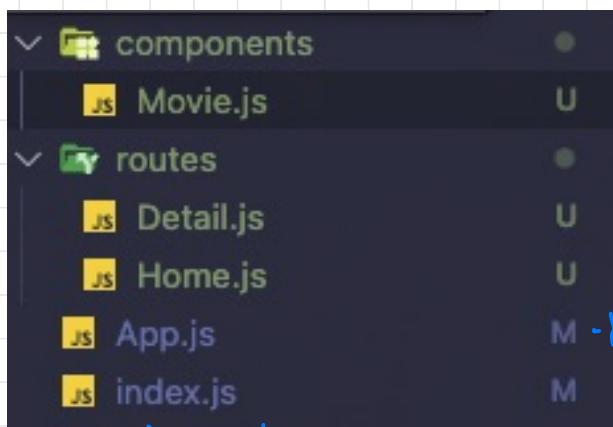
HTML 요소는 중대하거나 긴급한 주제를 나타냅니다. 보통 브라우저는 짙은 글씨로 표시합니다.

loading state가 false

done



Movie app



→ Component 폴더

자체 폴더

- router 폴더

M - router-dom.js Routes 를 Rendering

L Router
App Vendoint

Index.js

```
index.js > ...
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
const root: ReactDOM.Root = ReactDOM.createRoot(document.getElementById("root"));
root.render([
  // <React.StrictMode>
  <App />
  /* </React.StrictMode> */
]);

```

→ root element 생성 후
App Component Rendering

App.js

App.js > ...

```
import Home from "./routes/Home";
import Detail from "./routes/Detail";
import { BrowserRouter as Router, Routes, Route } from "react-router-dom"
function App() {
  return (
    <Router> → Router Component 사용
      <Routes> → Route (url)를 찾고 컴포넌트를 랜더링한다.
        <Route path="/" element={<Home />} />
        <Route path="/movie/:id" element={<Detail />} />
      </Routes> ⚡️ 먼저가해당 경로에 있다면,
    </Router> ⚡️ Detail Route를 Rendering 함.
  );
}

export default App;
```

route 필요한 component를 import

URL을 바라보다가 URL이 바뀌면 어떤 것을 보여줄지 결정한다!

라우팅?

- 사용자가 요청한 URL에 따라 알맞은 페이지를 보여주는 것.

⇒ 페이지별로 컴포넌트들을 분리해가며 관리 할 수 있다.

2. 싱글 페이지 애플리케이션이란?

싱글 페이지 애플리케이션이란, 한 개의 페이지로 이루어진 애플리케이션이라는 의미입니다. 리액트 라우터를 사용하여 여러 페이지로 구성된 프로젝트를 만들 수 있다고 했었는데 왜 싱글 페이지 애플리케이션이라고 불리는지 의문이 들 수 있습니다.

이를 이해하기 위해서는, 싱글 페이지 애플리케이션이란 개념이 생기기 전에 사용되던 멀티 페이지 애플리케이션은 어떻게 작동하는지 살펴볼 필요가 있습니다.

멀티 페이지 애플리케이션에서는 사용자가 다른 페이지로 이동할 때마다 새로운 html을 받아오고, 페이지를 로딩할 때마다 서버에서 CSS, JS, 이미지 파일 등의 리소스를 전달받아 브라우저 화면에 보여주었습니다. 각 페이지마다 다른 html 파일을 만들어서 제공을 하거나, 데이터에 따라 유동적인 html을 생성해 주는 템플릿 엔진을 사용하기도 했죠.

사용자 인터랙션이 별로 없는 정적인 페이지들은 기존의 방식이 적합하지만, 사용자 인터랙션이 많고 다양한 정보를 제공하는 모던 웹 애플리케이션은 이 방식이 적합하지 않았습니다. 새로운 페이지를 보여주어야 할 때마다 서버 측에서 모든 준비를 한다면 그만큼 서버의 자원을 사용하는 것이고, 트래픽도 더 많이 나올 수 있기 때문이죠.

그래서, 리액트 같은 라이브러리를 사용해서 뷰 렌더링을 사용자의 브라우저가 담당하도록 하고, 우선 웹 애플리케이션을 브라우저에 불러와서 실행시킨 후에 사용자와의 인터랙션이 발생하면 필요한 부분만 자바스크립트를 사용하여 업데이트하는 방식을 사용하게 됐습니다. 만약 새로운 데이터가 필요하다면 서버 API를 호출하여 필요한 데이터만 새로 불러와 애플리케이션에서 사용할 수 있게 됐죠.

이렇게 html은 한번만 받아와서 웹 애플리케이션을 실행시킨 후에 그 이후에는 필요한 데이터만 받아와서 화면에 업데이트 해주는 것이 싱글 페이지 애플리케이션입니다.

싱글 페이지 애플리케이션은 기술적으로는 한 페이지만 존재하는 것이지만, 사용자가 경험하기에는 여러 페이지가 존재하는 것처럼 느낄 수 있습니다. 리액트 라우팅과 같은 라우팅 시스템은 사용자의 브라우저 주소창의 경로에 따라 알맞는 페이지를 보여주세요. 이후 링크를 눌러서 다른 페이지로 이동하게 될 때 서버에 다른 페이지의 html을 새로 요청하는 것이 아니라, 브라우저의 History API를 사용하여 브라우저의 주소창의 값만 변경하고 기존에 페이지에 띄워둔 웹 애플리케이션을 그대로 유지하면서 라우팅 설정에 따라 또 다른 페이지를 보여주게 됩니다.

Home.js (Routes)

routes > Home.js > Home

```
✓ import Movie from "../components/Movie";
  import { useState, useEffect } from "react";

✓ function Home() {
  const [loading, setLoading] = useState(true); - Loading State
  const [movies, setMovies] = useState([]); - 영화 정보 Array State
  const getMovies = async () => {
    const json = await (
      await fetch(
        "https://yts.mx/api/v2/list_movies.json?minimum_rating=9&sort_by=rating"
      )
    ).json();
    setMovies(json.data.movies); → movies 배열을 movie data 배열로 바꿔서
    setLoading(false); → Loading State를 false로 바꿔 초기화면 사용자에게 해석
  };
  useEffect(() => {
    getMovies(); → useEffect 사용하여 첫Rendering 때만 getMovies 실행
  }, []);
  return (
    <div>
      {loading ? (
        <h1>Loading ...</h1> )} → Coding의 boolean 상태에 따라 text Rendering
      : (
        <div>
          {movies.map((movie) => (
            <Movie
              key={movie.id}
              id={movie.id}
              coverImg={movie.medium_cover_image}
              title={movie.title}
              summary={movie.summary}
              genres={movie.genres}
            />
          ))}
        </div>
      )
    </div>
  );
}

export default Home;
```

JSON data 불러오기

movies 배열을 movie data 배열로 바꿔서 사용하기

Loading State를 false로 바꿔 초기화면 사용자에게 해석

useEffect 사용하여 첫Rendering 때만 getMovies 실행

map 함수로 배열안의 각 요소를 펼쳐는 규칙으로 나열.

↳ Movie 컴포넌트 rendering
[인자로 받은 movie (JSX를 원함)
를 props로 이용]

Movie.js

```
import PropTypes from "prop-types";
import { Link } from "react-router-dom";
function Movie({ id, coverImg, title, summary, genres }) {
  return [
    <div>
      <h2>
        <Link to={`/movie/${id}`}>{title}</Link>
      </h2>
      <img src={coverImg} alt={title} />
      <p>{summary}</p>
      <ul>
        {genres.map((g) => (
          <li key={g}>{g}</li>
        ))}
      </ul>
    </div>
  ];
}

Movie.propTypes = {
  id: PropTypes.number.isRequired,
  coverImg: PropTypes.string.isRequired,
  title: PropTypes.string.isRequired,
  summary: PropTypes.string.isRequired,
  genres: PropTypes.arrayOf(PropTypes.string).isRequired,
};

export default Movie;
```

링크에 ID 뒷붙하기 → 동적
내가 생성한 props

a TAG로 Link?
⇒ 페이지 전환과 reload 되는 문제

React-dom의 Link component 사용

) propTypes 사용

JavaScript에서 true && expression은 항상 expression으로 평가되고, false && expression은 항상 false로 평가됩니다. 따라서 && 뒤의 엘리먼트는 조건이 true일때 출력이 됩니다. 조건이 false라면 React는 무시합니다.

false로 평가될 수 있는 표현식을 반환하면 && 뒤에 있는 표현식은 건너뛰지만 false로 평가될 수 있는 표현식이 반환된다는 것에 주의해주세요. 아래 예시를 통해 주의해야되는 이유를 알아보겠습니다.

