리액트란 무엇인가

자동으로 업데이트 되는 UI

UI = render(state)

Render 함수는 순수 함수로 작성  
- 랜덤 함수 사용 (x)  
- 외부 상태 변경 (x)

State는 불변 변수로 관리

가상 돔(virtual dom)

바벨

자바스크립트 코드를 변환해 주는 컴파일러

최신 자바스크립트 문법을 지원하지 않는 환경에서도 최신 문법 사용 가능

그 외에도 다양한 용도로 사용 ex) 코드에서 주석을 제거하거나 코드를 압축

리액트에서는 jsx 문법을 사용하기 위해 바벨을 사용

Npx install @babel/core @babel/cli @babel/preset-react 라고 쓰면 바벨을 쓸 수 있다.

JSX 문법

//자바스크립트에서는 class가 예약어이기 때문에, className이라고 줄 수 있다.

//Title같이 컴포넌트도 html의 다른 태그와 같이 비슷한 방식으로 입력할 수 있따.

//속성값을 보면 문자열을 간단하게 입력할 수 있고, 문자열이 아닌 경우 {}으로 감싸서 쓸 수 있다.

//이벤트 헨들러 ex)onClick 쪽도 중괄호로 감싸져 있다.

//자바스크립트에서는 - 문자가 마이너스로 인식되기때문에 카멜케이스로 써야한다.

<div className="box">

    <Title text="hello world" width={200}/>

    <button onClick={()=>{}}>좋아요</button>

    <a href="/home" style={{marginTop: '10px', color: 'red'}}>홈으로 이동</a>

</div>

  //당연하게도 jsx 문법은 자바스크립트 표준이 아니기 때문에

  //브라우저에서 실행하면 에러가 발생한다.

  //따라서 바벨을 이용해서 jsx 코드를 createElement 코드로 변환해야 한다.

  //npx babel --watch src --out-dir . --presets @babel/preset-react

  //--watch 한번만 컴파일 하고 마는것이 아니라 변경이 감지될 때마다 새롭게 컴파일을 한다.

  //--out-dir . 현재 폴더로 컴파일을 한다.

  //--presets 어떤 preset을 쓸지 결정한다.

Create-react-app으로 시작하기-1

Create-react-app 과 next.js 의 가장 큰 차이는 서버사이드 렌더링의 지원 여부이다.

Create-react-app: 서버사이드 렌더링을 지원하지 않는다.

서버사이드 렌더링이 필수인 프로젝트라면 next.js를 선택하는게 좋다.

Cra의 단점은 cli나 eslint 등의 설정을 거의 변경할 수 없다는 점이다.

Next.js 에서는 바벨, 웹팩, eslint 등의 설정을 변경할 수도 있다.

Css를 변경했을 때 바로 적용이 된다.

이렇게 HMR이 동작하는 것은 npm start로 시작을 했을 때,

로컬에 서버를 띄워서 브라우저와 통신을 하기 때문에 가능한 것이다.

참고로 mpm start는 배포할 때 사용하면 안된다. 개발 모드에서만 사용해야 한다.

최적화가 전혀 되어 있지 않다.

배포할 때는 반드시 빌드 명령어를 사용해서 배포해야 한다.

root.render(

  <React.StrictMode>

    <App />

  </React.StrictMode>

);

여기서 React.StrictMode는 개발환경에서 동작을 한다.  
react에서 잘못 사용한 것을 빠르게 찾아내기 위해서 사용한다.

import logo from './logo.svg';

import './App.css';

function App() {

위에 보면 이미지 파일도 import를 해서 가져오고 있다.  
css나 이미지 파일을 자바스크립트에서 import해서 사용하는 것이 좋다.  
이렇게 사용하면 이미지 경로에 해시 값이 들어가기 때문에 브라우저 캐싱을 효율적으로 활용할 수 있다.

function App() {

  function onClick(){

    import('./data.json').then(({default:data})=>{

      console.log({data});

    })

  }

import를 해서 사용하면 좋은점이 또 있는데, 데이터의 사이즈가 클 경우에, 그리고 이 데이터가 항상 필요한게 아니라 특정 순간에 필요한 경우 그 때 데이터를 받아오는 것이 좋다.

Network 쪽에서 보면, 버튼을 클릭했을 때 동적으로 데이터를 받아오는 것을 알 수 있다.

Create-react-app으로 시작하기-2

  "scripts": {

    "start": "react-scripts start",

    "build": "react-scripts build",

    "test": "react-scripts test",

    "eject": "react-scripts eject"

  },

Npm start로 실행을 하게 되면 기본적으로 http로 시작하게 된다  
https로 시작하고 싶다면 HTTPS=true npm start 를 터미널에 입력하면 된다.  
윈도우용 명령어 🡪 set HTTPS=true && npm start ??? 이걸로 하면 https 가 안된다??

Build 명령어는 배포할 때 사용한다.  
빌드를 하게 되면 정적 파일이 생성된다. Build 폴더 생성됨  
서버에서는 생성된 빌드 폴더 안에 있는 내용들을 정적파일로 서비스 하면 된다.  
별도로 서버에 어플리케이션을 실행하지 않기 때문에 서버사이드 렌더링으로 동작할 수 없다.

Build 폴더 안에 있는 static/css 안에 css 파일이 있는데 기존 src에 있던 App.css, index.css가  
합쳐져 있다고 생각하면 된다.

"use strict";(self.webpackChunkcra\_test=self.webpackChunkcra\_test||[]).push([[468],{468:function(e){e.exports=JSON.parse('{"name":"lee","age":35}')}}]);

이전 시간에 동적 import를 사용해서 data를 받아왔었는데,  
동적 import를 사용하게 되면 해당 파일은 js파일로 만들어지게 된다.

이미지의 경우, 사이즈에 따라서 동작하는 방법이 다르다.  
이미지가 큰 경우, static/media 폴더 안에 들어가게 되고,  
이미지가 작을 경우, static/js 폴더 안에 있는 js 파일에 내장된다.  
이렇게 자바스크립트에 내장하는 한가지 이유는, http 요청 횟수를 줄이는 목적이다.  
그러나 http2.0 이후로는 요청이 많아도 성능 부분에서 그렇게 차이가 크지 않기 때문에 큰 의미는 없다.  
또 한가지 이유는 빠르게 이미지를 보여줄 수 있다.

Npm test를 실행하게 되면 원본 src에 App.test.js가 생성이 되고 test가 실행된다.  
여기서 App.spec.js로 변경해도 테스트 파일로 인식을 한다..  
src 폴더 안데 \_test\_ 파일을 만들고 그 안에다가 js을 넣으면 test나 spec 이외의 다른 이름이 와도  
테스트 파일로 인식한다.  
그러나 폴더에 넣어서 관리하는 것 보다는 .test.js 나 .spec.js로 관리하는게 편하다고 생각한다.  
원본 코드와 붙어있는게 편하다고 생각한다.

Eject 명령어는 react-script를 사용하지 않고 모든 설정 파일을 추출하는 명령이다.  
cra를 기반으로 직접 개발환경을 구축하고 싶을 때 사용한다.  
추출을 하지 ㅇ낳으면 cra의 기능이 추가 됏을 때 단순히 react-scripts 버전만 올리면 되는데  
추출을 하면 설정 파일을 수동으로 수정해야 한다는 단점이 있다.  
따라서 꼭 필요한 경우가 아니라면 관리측면에서 추출하지 않는게 좋다.

Cainiuse

브라우저별로 특정 기능에 대해서 지원 여부를 확인하기 좋다.  
ex) padStart 의 경우에는 IE에서는 지원을 하지 않는다.  
이 때, IE에서도 padStart를 사용하고 싶다면 polyfill 을 추가해야 한다.  
보통 polyfill 을 추가할 때는 core-js 라는 것을 많이 사용한다.  
core-js 깃헙에 들어가서 pad-start 검색하면 찾을 수 있는데, 그것을 index.js에 import 해서 사용하면 된다.  
원래는 core-js를 설치하고 사용해야 하지만, cra가 기본적으로 내장하고 있다.

환경변수는 개발,테스트 또는 배포 환경별로 다른 값을 적용할 때 유용하다.  
전달된 환경변수는 코드에서 process.env.{변수이름} 이런식으로 사용 할 수 있다.

// npm start 로 실행하면 development

// npm test 로 실행하면 test

// npm run build 로 실행하면 production

환경변수가 많아지면, .env 파일이라는 것으로 관리하는게 좋다.

Ex) env.development env.production

Npm start로 실행하면 env.development 변수가 출력되고

Npm run build 한뒤 npx serve -s build 를 실행하면 env.production 변수가 출력된다.

Css 작성 방법 결정하기

1. 일반적인 css 파일
2. Css-module
3. Sass
4. Css-in-js

일반적인 css 파일로 작성했을 때는, 이름이나 클래스가 겹쳐지게 되면  
우리가 기대했던 데로 적용이 되지 않는 모습을 볼 수 잇다.

Css-module 을 사용하면 이름충동 문제를 해결 할 수 있다.

import React from 'react';

//import 할 때 객체 형식으로 내보내기 때문에 객체형식으로 받아준다.

import Style from './Button.module.css';

export default function Button({size}) {

    if(size === 'big'){

        //클래스 명을 속성이름으로 입력해주면 된다.

        return <button className={`${Style.button} ${Style.big}`}>큰 버튼</button>

    }else{

        return <button className={`${Style.button} ${Style.small}`}>작은 버튼</button>

    }

}

콘솔에 값을 찍어보면 객체 형식으로 나오고 있다.  
그리고 각각의 속성값을 보면, 뒤에 해시 값이 붙어있는 것을 알 수 있다.  
이 해시값 때문에 각 클래스 명은 고유한 값을 가진다.  
**Style**:  
**big**: "Button\_big\_\_yJYj4"  
**button**: "Button\_button\_\_UW-r+"  
**small**: "Button\_small\_\_L3-zs"

Css-module을 사용할 때 className을 입력할 때 조금 번거롭다.  
이것은 classnames라는 패키지를 이용하면 조금 더 간편하게 이용할 수 있다.  
npm i classnames

export default function Button({size}) {

    //클래스 명을 속성이름으로 입력해주면 된다.

    //className 을 입력할 때 조금 번거롭다.

    //npm i classnames 을 설치하면 조금 간편하게 이용할 수 있다.

    //cn에는 다른 기능도 있다. 아래코드를 변환해보자.

    const isBig = size==='big';

    return(

        <button

            className={cn(Style.button,

                {

                    [Style.big]: isBig,

                    [Style.small]: !isBig

                })}

        >

            {isBig ? '큰 버튼' : '작은 버튼'}

        </button>

    )

    // if(size === 'big'){

    //     return <button className={cn(Style.button, Style.big)}>큰 버튼</button>;

    // }else{

    //     return <button className={`${Style.button} ${Style.small}`}>작은 버튼</button>;

    // }

}

객체를 입력할 때 키를 css 명으로 하고 값은 boolean으로 했을 때,  
값이 true 이면 왼쪽에 있는 키 속성명이 사용이 된다.

Sass

cra에서 sass를 이용하려면 node-sass 라는 패키지를 먼저 설치해야 한다.  
npm I node-sass  
그리고 css 이름은 scss로 작성해줘야 한다. 그리고 scss를 사용하면서 module을 같이 사용할 수 있다.  
ex) Button.module.scss

shared.scss 를 보면 변수를 정의하듯 infoColor를 정의해놨다.  
박스css나 버튼css를 가보면 같은 값을 사용하고 있다.

$infoColor:#ffccaa;

.box{

    height: 50px;

    background-color: $infoColor;

}

.button{

    height: 30px;

    background-color: $infoColor;

}

Css-in-js

Css-in-js 는 css를 자바스크립트 코드 안에서 작성하는 방식이다.  
css 코드가 자바스크립트 안에서 관리 되기 때문에   
css 코드도 자바스크립트 코드처럼 얼마든지 재사용 가능한 구조로 관리 할 수 있다.  
또 동적으로 css 코드를 작성하기도 쉽다.

다양한 css-in-js 라이브러리 중에서 styled-components 를 사용해 보자.  
npm i styled-components

import React from 'react';

import styled from 'styled-components';

/\*

styled-components 에서 styled 라는 객체를 받아 온 다음에

.div 로 스타일 코드를 입력 하고 있다.

const BoxCommon = styled.div`

    height:50px;

    background-color: #aaaaaa;

`;

여기서 사용된 문법은 es6에 추가된 tagged template literals 문법이다.

이것은 그냥   ` height:50px; background-color: #aaaaaa;` 을

매개변수로 받아서 실행하는 함수가 있다고 생각하면 된다.

위에 코드에서는 div로 이루어진 컴포넌트를 생성한 것이다.

이렇게 만들어진 컴포넌트를 확장해서 다른 컴포넌트를 만들 수 있다.

\*/

const BoxCommon = styled.div`

    height:50px;

    background-color: #aaaaaa;

`;

const BoxBig = styled(BoxCommon)`

    width: 200px;

`;

const BoxSmall = styled(BoxCommon)`

    width: 100px;

`;

export default function Box({size}){

    if(size === 'big'){

        return <BoxBig>큰 박스</BoxBig>

    }else{

        return <BoxSmall>작은 박스</BoxSmall>

    }

}

import React from 'react';

import styled from 'styled-components';

//Box 함수 안쪽에서 return 할 때 속성값을 입력하고 styled를 이용해서 컴포넌트를 만들 때

//그 속성 값을 받아서 동적으로 처리를 할 수 있다.

const ButtonCommon = styled.button`

    width: ${props => (props.isBig ? 100 : 50)}px;

    height: 30px;

    background-color: yellow;

`;

export default function Box({size}) {

    const isBig = size === 'big';

    const label = isBig ? '큰 버튼' : '작은 버튼';

    return <ButtonCommon isBig={isBig}>{label}</ButtonCommon>

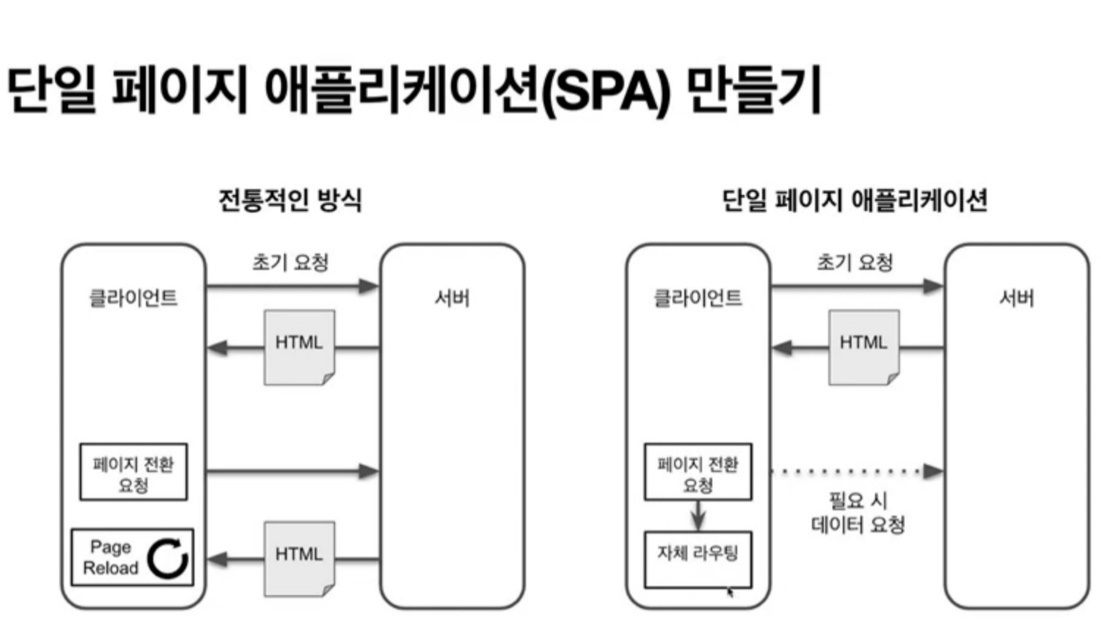
}

css-in-js 도 가능한 상황이라면 아래 두 가지 중에 하나를 추천드립니다.

1. scss 와 css-module 을 조합하는 방식  
2. css-in-js 방식

만약 빠른 렌더링이 중요한 프로젝트라면 1번으로 하시는 게 맞습니다.  
그렇지 않다면 이제부터는 취향의 문제라고 생각됩니다.  
css-in-js 방식은 잘만 사용하면 개발 생산성이 높습니다.

단일 페이지 애플리케이션(SPA) 만들기



SPA가 가능하기 위한 조건

* 자바스크립트에서 브라우저로 페이지 전환 요청을 보낼 수 있따.
  + 단, 브라우저는 서버로 요청을 보내지 않아야 한다.
* 브라우저의 뒤로 가기와 같은 사용자의 페이지 전환 요청을 자바스크립트에서 처리 가능
  + 이때도 브라우저는 서버로 요청을 보내지 않아야 한다.

위 조건을 만족시켜주는 브라우저 API

* pushState, replaceState 함수: 자바스크립트가 브라우저에게
* popstate 이벤트: 브라우저가 자바스크립트에게

import React, { useEffect, useState } from 'react';

import ReactDOM from 'react-dom';

function App(){

  const [pageName,setPageName] = useState('');

  //useEffect:

  //이벤트 핸들러를 등록하거나 api를 호출하는 등의 부수 효과를 발생시킬 때

  //사용하는 리액트 함수이다.

  //설명은 나중에 자세히 할 예정

  //여기서는 onpopstate를 사용하기 위해서 사용했다 정도로

  useEffect(()=>{

    window.onpopstate = function (event) {

      //console.log(`location: ${document.location}, state: ${event.state}`);

      setPageName(event.state);

    };

  },[]);

  function onClick1(){

    const pageName = 'page1';

    {/\*

      pushstate의 첫번째 매개변수는 데이터를 의미

      onpopstate에서 event 객체에 state라는 값이 있는데

      그 값이 첫번째 매개변수로 넘겨주는 값이다.

      두번째 매개변수는 타이틀이라는 건데 별로 중요x

      3번째는 url 주소

      \*/}

    window.history.pushState(pageName,'','/page1');

    setPageName(pageName);

  }

  function onClick2(){

    const pageName = 'page2';

    window.history.pushState(pageName,'','/page2');

    setPageName(pageName);

  }

  return (

    <div>

      <button onClick={onClick1}>page1</button>

      <button onClick={onClick2}>page2</button>

      {!pageName && <Home/>}

      {pageName === 'page1' && <Page1/>}

      {pageName === 'page2' && <Page2/>}

    </div>

  );

}

function Home(){

  return <h2>여기는 홈페이지입니다. 원하는 페이지 버튼을 클릭하세요.</h2>;

}

function Page1(){

  return <h2>여기는 Page1입니다.</h2>;

}function Page2(){

  return <h2>여기는 Page2입니다.</h2>;

}

ReactDOM.render(

  <React.StrictMode>

    <App/>

  </React.StrictMode>,

  document.getElementById('root')

)

React-router-dom을 이용해보자

?? 쓰는 이유는 여러가지 편의기능을 제공해주기 때문이다.

React hook

* 컴포넌트에 기능을 추가할 때 사용하는 함수
  + - Ex) 컴포넌트에 상태값 추가, 자식 요소에 접근
    - 리액트 16.8에 새로 추가됨
      * 그 전에는 클래스형 컴포넌트를 사용
      * 클래스형 컴포넌트보다 장점이 많으며 리액트 팀에서도 훅에 집중하고 있는중
      * <https://mk-v1.kakaocdn.net/dn/if-kakao/conf2019/conf_video_2019/1_104_01_m1.mp4>
* useState: 상태값 추가
* useEffect: 부수효과 처리
  + - 서버 API 호출, 이벤트 핸들러 등록 등

훅 사용 시 지켜야 할 규칙

* 1. 하나의 컴포넌트에서 훅을 호출하는 순서는 항상 같아야 한다.
* 2. 훅은 함수형 컴포넌트 또는 커스텀 훅 안에서만 호출되어야 한다.

npm install @babel/core @babel/cli @babel/preset-react

npx babel --watch src --out-dir . --presets @babel/preset-react