**React.js 1 - 개발 환경 셋팅, CLI 실행**

기본적으로 설치해야 할 프로그램 목록

node.js

npm

yarn

create-react-app

editor( visual studio code 등등 본인이 사용하는 editor )

git

설치시 cmd 또는 cmder를 설치해서 사용

1. node.js 설치

<https://nodejs.org/ko/download/> 에서 윈도우용을 다운받아서 설치

2. npm 설치

node.js 설치시 자동으로 설치

3. yarn 설치

cmd 또는 cmder에서

npm install --global yarn

설치확인 : yarn -v

4. create-react-app 설치

cmd 또는 cmder에서

npm install -g create-react-app

또는

yarn global add create-react-app

5. 프로젝트 생성

설치를 원하는 폴더로 이동

예) d:\react 폴더에 생성하고 싶다면

cd d:\react

npx create-react-app [프로젝트명]

cd [프로젝트명]

yarn start

react를 자동으로 실행하고 기본브라우져로 지정된 브라우져에 react 첫 화면이 나옴

DECEMBER 24, 2018

React.js에서 제공해주는 CLI를 실행하려면 먼저 개발 환경을 셋팅해주어야 한다. 개발 환경 셋팅은 아래와 같은 순서로 진행된다.

1. nvm 설치
2. node.js 설치
3. nvm을 이용하여 node.js 버전 셋팅
4. yarn 설치
5. CLI 프로젝트 생성
6. intellij 설정
7. CLI 프로젝트 실행

Mac OS와 Intellij IDE 기반으로 진행되지만, Window에서의 환경 셋팅 또한 크게 다르지 않다. 1번에서 nvm를 설치할때만 Window 전용 nvm인 [nvm-window](https://github.com/coreybutler/nvm-windows)를 설치해주면 된다.

**1. nvm 설치**

nvm(Nodejs Version Management)을 사용하면 쉽게 node.js의 버전을 관리할 수 있게 된다. Mac 에서 nvm을 사용하지 않고 node.js를 먼저 설치하게 되면 루트 계정에 설치되기 때문에 권한 문제로 node.js 다루기가 까다로워진다.

[Mac, Linux 운영체제에서 nvm 설치하기](https://github.com/creationix/nvm#manual-install)

[Windows 운영체제에서 nvm 설치하기](https://github.com/coreybutler/nvm-windows)

**2. node.js 설치**

이전에 node.js를 설치한 이력이 있다면 전부 깨끗하게 삭제하고 지우는 것을 추천한다. node.js를 완전히 삭제하는 방법은 구글에 치면 자세하게 나온다.

nvm으로 node.js를 설치하는 명령어는 아래와같다.

**최근 버전 중에서 안정된 버전의 node 설치하기**

nvm install lastest

**node.js 설치 여부 확인**

node –-version

**3. nvm을 이용하여 node.js 버전 셋팅**

다시 nvm ls 명령어를 입력하여 설치가 잘 되었는지 확인해보자. 설치가 잘 되었는지 확인한 후에 node.js 버전을 셋팅하는 명령어를 입력해주어야 한다. (2018/12/24 기준 latest stable 버전은 11.5.0)

**nvm으로 node.js 설치 여부 확인하기**

nvm ls

**nvm으로 node.js 사용 버전 셋팅하기**

nvm use 11.5.0

**4. yarn 설치**

yarn은 npm의 업그레이드 버전이라고 생각할 수 있는데, npm보다 훨씬 쾌적한 속도를 자랑한다.

**yarn 설치하기**

npm install -g yarn

이제 React.js CLI 프로젝트를 실행할 수 있는 개발 환경 셋팅이 완료되었다!!

**5. CLI 설치**

React CLI 설치는 [이 사이트](https://velopert.com/2037)를 참고하였다.

**설치하기**

npm install -g create-react-app

설치가 완료되면 프로젝트를 생성하고자 하는 위치로 이동하고 아래의 명령어를 입력하면 프로젝트가 생성된다.

**프로젝트생성**

npx create-react-app my-app

npm init react-app

CSS 작성

1. inline ( 키는 camelCase )

style = { }

객체로 작성 {

key : “value “,

}

style = {{

color : ‘red’,

}}

1. 전역 사용 CSS

name.CSS 파일안에 작성 시 전역에 적용

1. 객체별로 구분하기 (module.css)

name.module.css 작성 후에 적용 시킬 js 파일에 import styles from ‘./name.module.css’;

적용 : className = { styles.Selector }

리액트 Router

1. 설치 npm i( = install) react-router-dom

index.js

import { ReactDOM } from 'react/client';

* BrowserRouter, Router, Route 컴포넌트 작성

BrowserRouter : ui 의 최상위에 작성

Router : Router set / Route 집합

Route : 링크 설정 / 컴포넌트 정의

import ReactDOM from 'react-dom/client';

import { BrowserRouter, Routes, Route, NavLink, Link, useParams } from 'react-router-dom';

<BrowserRouter>

  <Link to='/url'>클릭 리스트</Link>

  {/\* 싱글 페이지 전환 \*/}

  <Routes>

    <Route path='/url1' element={<Component1 />}></Route>

    <Route path='/url2' element={<Component2 />}></Route>

  </Routes>

</BrowserRouter>

주소 이동

Link to=주소 = path=주소

path : 브라우저 주소

path/\* : 해당하지 않는 모든 주소

element : 출력될 컴포넌트

indet element : 처음 보여질 메인 페이지 /

Link - 클릭

NavLink - 내부링크 / active 클래스 추가 / css 로 클래스 설정 가능

중첩 라우터

상위 컴포넌트 path = 링크주소/\* 처리

하위 컴포넌트 path = 상위주소 / 링크수조

Event 처리

함수 설정 후 호출

    function onClickAlert() {

        alert("alert Success")

    }

 <button onClick={onClickAlert}>show alret</button>

내부에 직접 함수 설정

            <button onClick={

                () => {

                    prompt("First", "seocnd")

                }

            }>Show prompt</button>

react Hook

* [기본 Hook](https://ko.reactjs.org/docs/hooks-reference.html#basic-hooks)
  + [useState](https://ko.reactjs.org/docs/hooks-reference.html#usestate)
  + [useEffect](https://ko.reactjs.org/docs/hooks-reference.html#useeffect)
  + [useContext](https://ko.reactjs.org/docs/hooks-reference.html#usecontext)
* [추가 Hooks](https://ko.reactjs.org/docs/hooks-reference.html#additional-hooks)
  + [useReducer](https://ko.reactjs.org/docs/hooks-reference.html#usereducer)
  + [useCallback](https://ko.reactjs.org/docs/hooks-reference.html#usecallback)
  + [useMemo](https://ko.reactjs.org/docs/hooks-reference.html#usememo)
  + [useRef](https://ko.reactjs.org/docs/hooks-reference.html#useref)
  + [useImperativeHandle](https://ko.reactjs.org/docs/hooks-reference.html#useimperativehandle)
  + [useLayoutEffect](https://ko.reactjs.org/docs/hooks-reference.html#uselayouteffect)
  + [useDebugValue](https://ko.reactjs.org/docs/hooks-reference.html#usedebugvalue)
  + [useDeferredValue](https://ko.reactjs.org/docs/hooks-reference.html#usedeferredvalue)
  + [useTransition](https://ko.reactjs.org/docs/hooks-reference.html#usetransition)
  + [useId](https://ko.reactjs.org/docs/hooks-reference.html#useid)
* [Library Hooks](https://ko.reactjs.org/docs/hooks-reference.html#library-hooks)
  + [useSyncExternalStore](https://ko.reactjs.org/docs/hooks-reference.html#usesyncexternalstore)
  + [useInsertionEffect](https://ko.reactjs.org/docs/hooks-reference.html#useinsertioneffect)

**State / useState**

동적 값 할당 / component 상태 관리

function Counter() {

const [number, setNumber] = useState(0);

const onIncrease = () => {

setNumber(number + 1);

}

const onDecrease = () => {

setNumber(number - 1);

}

* cosnt [state, setState ] = useState(value); / state와 setState 값을 배열로 리턴
* number = 초기값 / setNumber = 배열
* 렌더링 될 때(값이 변할 때) 마다 useState 안의 값을 호출 / 작업이 오래걸림
* 불러올값이 이전 값과 연관이 있다면 콜백 형태로 값을 가져오기 setState((prevState)=>{

return newState ();

} )

* 초기값을 받아올 때 무거운 작업이라면 콜백함수로 처리

useState(() => {

return heavyWork();

})

**useRef**

/ dom 에 접근

  //useRef

  const ref = useRef(value)

  ref => { current : value}

* ref의 변화 - 리렌더링 되지 않음 / 변수 값을 그대로 유지
* dom 요소 접근 용이 / document.QuerySelector

  // Dom 요소에 접근

  const selRef = useRef(valuse)

  // 접근할 요소에 값 입력 = document.QuerySelector 와 비슷한 이치

  <input ref = { selRef } />  
  selRef = input // input에 대한 모든 정보를 입력

selRef.current.value // input value 접근

selRef.current.focus(); // input 태그 focus

**useEffect**

상태 값 변환 시 실행 함수

* 상태 값이 바뀌었을 때 동작하는 함수 ( mount / unmount / update )
* 인자 = callback function / useEffect ( ( ) => { //작업 } )

callback function = 다른 함수의 인자로 전달된 함수

  //첫번째 형태 - 컴포넌트가 업데이트 될 때 마다 실행 / 렌더X

  useEffect(() => { //작업 });

  //두번째 형태 - 처음 화면에 렌더링 시 실행 / value 의 값이 바뀔 때 실행 / 렌더 O

useEffect(() => { //작업 }, [value]);

useEffect( function( ) { }, ) // 함 수 호출 돔에 반영된 직 후

첫 번째 매개 변수 = function () // 실행 할 함수  
두 번째 매개 변수 = Array // 의존성 배열이 변경되는 경우에만 실행

* 배열의 변수가 객체일 경우엔 메모리상에 저장이 되기 때문에 렌더가 될 때 마다 새로운 주소를 할당 받음

clean up

* useEffect로 실행한 함수를 정지 시키는 것
* retun ()=>{ }; 값으로 함수를 반환 // 함수안에서 정리 코드 작성
* unmount 될 때 삭제

**useContext / Context API**

Context - 전역으로 사용되는 데이터들을 다른 컴포넌트에서 쉽게 사용 가능하게 해주는 기능

* props 로 하위컴포넌트에게 값을 전달하지 않아도 됨 (props Drilling 하지 않고 값을 전달)
* Context API로 공유한 데이터를 useContext 사용해서 데이터 전달/받기 가능
* Context를 사용하면 컴포넌트 재사용하기 어려워질수도 있다
* react context 만들기

//Context.js 파일 안에 작성

import { createContext } from 'react';

export const createContext = createContext(null);

* Context 사용하기

//Context 사용하기

import {Context} from './components/Context';

//Provider 는 밸류를 받음 data들은 하위 컴포넌트에서 접근 가능

//Provider 로 감싸주지 않으면 밸류값 대신 Context.js 의 createContext(null) 의 값을 호출

<Context.Provider value={{data, data1, data2}}>

 <Component1 />

 <Component2 />

 <Component3 />

</Context.Provider>

하위 컴포넌트에서 data 사용

const { data } = useContext(Context) //import 한 Context 입력

data = { data, data1 data2 } 들어있음

//Context 사용하기

import {Context} from './components/Context';

**useMemo**

Component 최적화 / 메모리 최적화

반복 호출되는 함수를 cashing 해두고 함수를 호출하지 않고 cashing 해둔 값을 가져옴

useMemo(()=> { return memoization }, [])

2개의 인자 = 콜백함수 , 배열(의존성배열);

useMemo(callback Function, [array])

const value = useMemo(()=> Funtion(), [])

콜백 함수 내 memorization 한 값을 리턴

**useCallback**

구조

    useCallback(() => {

        return value;

    }, [의존성배열])

    두개의 인자

    콜백함수 = memoization 해줄 함수

    의존성배열의 값이 변경되지 않는 이상 다시 초기화되지 않음

useHistory / router 기능

link to 기능 / 주소 입력 시 해당 주소로 이동 / a태그를 사용하지 않고 페이지 전환  
history = useHistory( );

history.push ( url )

Life cycle

컴포넌트가 생성되는 시점 constructor

컴포넌트가 출력되는 과정 render

컴포넌트가 완료된 상태 = componentDidMount

컴포넌트가 완료되고 업데이트 될 때 = componentDidUpdate

컴포넌트 제거 할 때 = componentWillUnmount

클래스 방식 구현

* contsturtor
* > componentWillMount(지금은 사용X)
* render = 출력
* componentDidMount = 렌더 후 실행
* ShouldComponentUpdate = state가 변경 되었을 때 render( ) 실행 여부 결정 / return Boolean (true / false)
* > componentWillUpdate(지금은 사용X) = state 변경후 render 사이에서 실행
* componentDidUpdate = state 값이 변경되어 회면에 그려진 이후 실행
* componentWillUnmount // window.unload = component가 삭제 될 때 실행

const [A, B] = useState(param);

A= 변수, B

Class 스타일 vs Function 스타일 컴포넌트

클래스 문법

* 리액트 기능 100% 활용
* 클래스 문법 이해 필요
* 많은 자료 클래스 문법

## class ClassName extends Component { render ( ) { return (

**내용 출력**

## );

## } }

render 안에 모든 것을 새로 출력

함수문법

* 기능 부족
* 함수 문법 사용
* 컴포넌트 내부 state 사용 불가
* Life cycle API 사용 불가 (컴포넌트 생성,수정,삭제 관련 이벤트)
* Hook 도입으로 class 정도의 성능 지원 (16.8 버전 이상)

## function FunctionName ( ) { return (

**내용 출력**

## ); }

설치 해야 할 것

        "react": "^18.2.0",

        "react-dom": "^18.2.0",

        "react-icons": "^4.7.1",

        "react-router-dom": "^6.8.0",

        "framer-motion": "^8.5.4",

아이콘

npm install react-icons --save

프레임 모션

npm install framer-motion

npm install node-sass

슬라이드 slick-slide

npm install react-slick --save

npm install slick-carousel --save

### import "slick-carousel/slick/slick.css";

### import "slick-carousel/slick/slick-theme.css";

font Awesome (폰트)

npm i @fortawesome/fontawesome-svg-core @fortawesome/free-solid-svg-icons @fortawesome/free-regular-svg-icons @fortawesome/free-brands-svg-icons @fortawesome/react-fontawesome

**사용법**

import ReactDOM from 'react-dom'

import { FontAwesomeIcon } from '@fortawesome/react-fontawesome'

import { faCoffee } from '@fortawesome/free-solid-svg-icons'

const element = <FontAwesomeIcon icon={faCoffee} />

ReactDOM.render(element, document.body)