

Ver.2021 노베이스 모던 웹개발

- 2. React Hook

작성자: 경상국립대학교 컴퓨터과학과 조교 이자룡

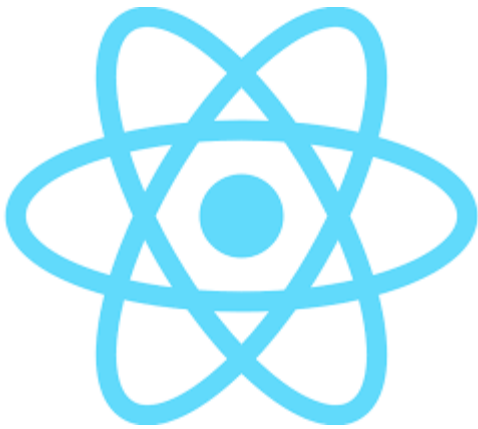
목차

p.2	-	0. Orientation
p.7	-	1. HTML
p.19	-	2. React 프로젝트 디렉토리 구조
p.22	-	3. 컴포넌트(component)
p.29	-	4. props
p.37	-	5. map() 을 이용한 컴포넌트 반복
p.46	-	6. hook
p.53	-	7. router
p.59	-	8. route props
p.60	-	9. 마치며

0. Orientation

* 이 수업을 시작하기 전에, 반드시 JavaScript, TypeScript 기초를 알고 시작하길 바란다.
《ver.2021 노베이스 모던 웹개발》 첫번째 강좌인 『React 시작 전에 알아야 할 JavaScript, TypeScript』가 준비되어 있다.

TypeScript는 언어다. 단어와 문법을 안다고 해서 언어를 잘한다고 할 수 없다. 언어는 실전에서 읽고, 쓰고, 듣고, 말할 때 의미가 있다. 우리는 TypeScript를 통해 웹앱(Web App)을 만들어 볼 것인데, TypeScript만 가지고 웹앱을 만드는 건 식칼만 있는 주방에서 일하는 것과 비슷하다. 그래서 선배 개발자들이 만들어놓은 여러 가지 '도구'를 사용해서 좀 더 쉽게 웹앱을 만들어 볼 것이다.



React는 공식적으로는 프론트엔드(frontend) 라이브러리(library)이다. 처음 나오는 용어부터 알고 넘어가자.

- * 프론트엔드(frontend): 사용자(client)가 앱을 사용할 때 눈에 보이는 화면을 말한다.
- * 백엔드(backend): 서버(server)를 말한다. 눈에 보이지 않지만, 앱이 작동하기 위해 꼭 필요한 부분이다.
- * 라이브러리(library): 미리 작성해둔 코드 모음이다. 주방 도구들과 비슷하다. 우리가 라면을 만들고 싶다고 냄비까지 만들진 않는다. 내가 만들고 싶은 라면에 딱 맞고, 품질도 좋은 냄비가 공짜라면 더욱 그러하다.
- * 프레임워크(backend): 코드를 모아서 규칙을 부여한 '틀'이다. 요리하기 위한 '주방'이라고 생각하자. 세계엔 수많은 주방이 있지만 어느 곳을 가든 비슷비슷하게 지어져 있다. 설거지하는 곳, 재료 다듬는 곳, 익히는 곳이 분리되어 있고, 각각의 도구와 직원은 그에 맞는 일을 한다. 그리고 규칙이 있어서, 주방장 말 안 들으면 쫓겨난다. 왜 그럴까? 수천 년의 역사 동안 시행착오를 거치면서 효율적으로 발전했기 때문이다.
- * API: Application Programming Interface. 두 프로그램이 서로 통신할 수 있도록 하는

중개자다. 가스렌지의 손잡이와 비슷하다. 가스렌지에 불 켜려고 손잡이를 돌리면 불이 켜진다. 손잡이와 화로 사이에 아무것도 없으면 마법을 부리지 않는 이상 불은 켜지지 않는다.

이 세 개를 구분하려고 지금도 어디에선가 개발자들 사이에서 논쟁이 벌어지고 있을 것임이 분명하다. 우리가 확실히 이해해야 할 건 이 셋의 정의나 차이가 아니라, 모두 다 '미리 쓰여진 코드'라는 것이다. **어쨌든 우리보다 똑똑한 사람들이 만들어놓은 코드를 가져다 써야 한다.** 우리 중 아무도 라면 끓이려고 냄비를 만들거나, 주방을 짓거나, 가스렌지 버튼을 설계하지 않는다.

* 개발자들은 돈에 욕심이 없는지, 자기가 오랜 시간 만든 코드를 돈 받지도 않고 공개해버리고, 심지어 자신의 코드를 남이 고칠 수 있게 한다. 이렇게 공개된 코드를 오픈 소스(open source)라고 한다. React 역시 오픈 소스다.

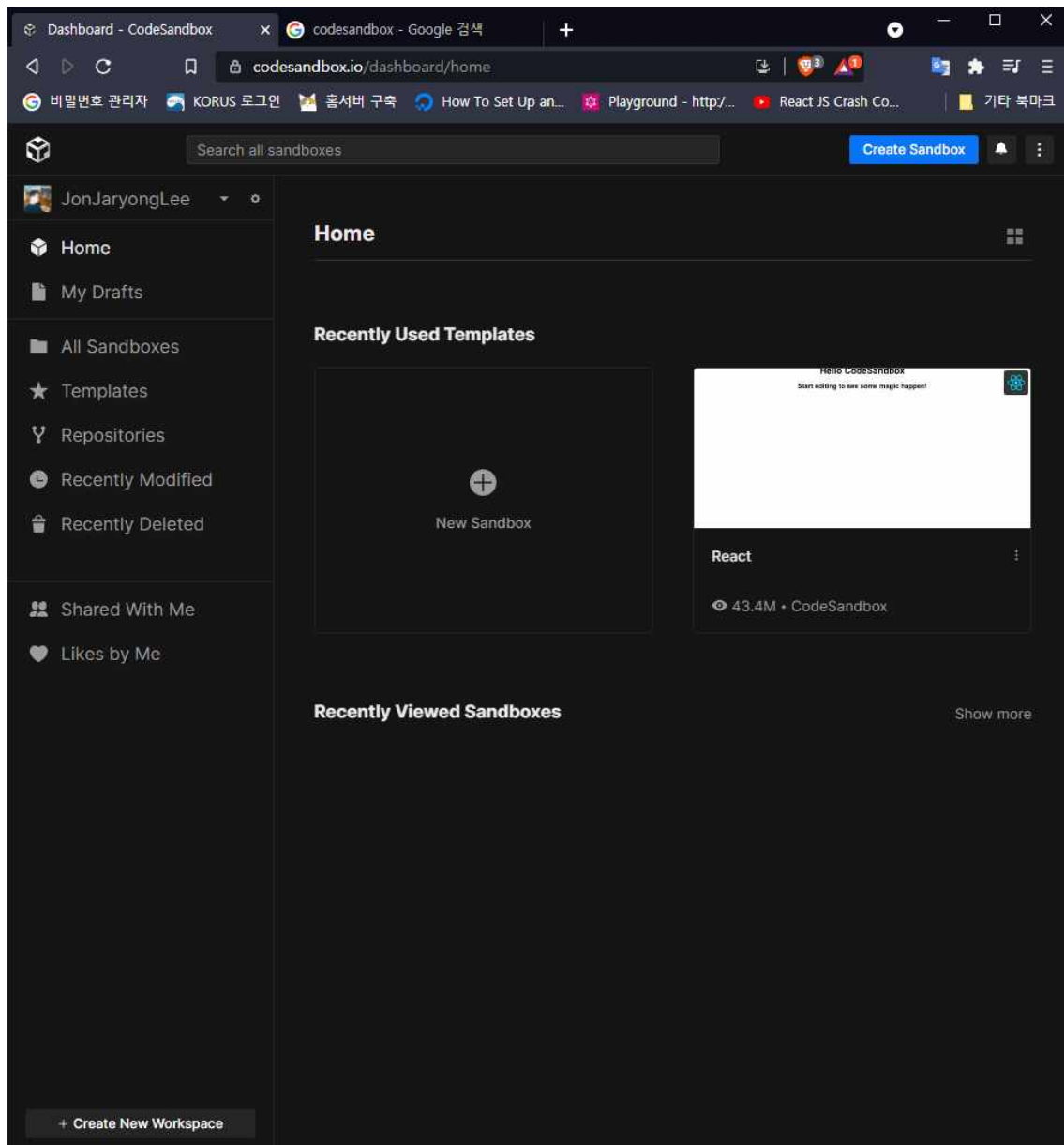
그럼 프론트엔드 기술로 왜 하필 React를 배우는가? 네 가지 이유가 있다.

1. 점유율이 높다. 경쟁자 Vue.js 에 비해 월등하다. 신규 프로젝트는 거의 React로 만들어진다.
2. 페이스북이 만들었고, 직접 관리한다. 대기업의 지원을 받는다는 건 어마어마한 강점이다.
3. 앱을 컴포넌트(component)로 나누어 개발한다.
4. 함수형 프로그래밍 방식인 Hook으로 개발하기 때문에 Vue.js 보다 쉽다.

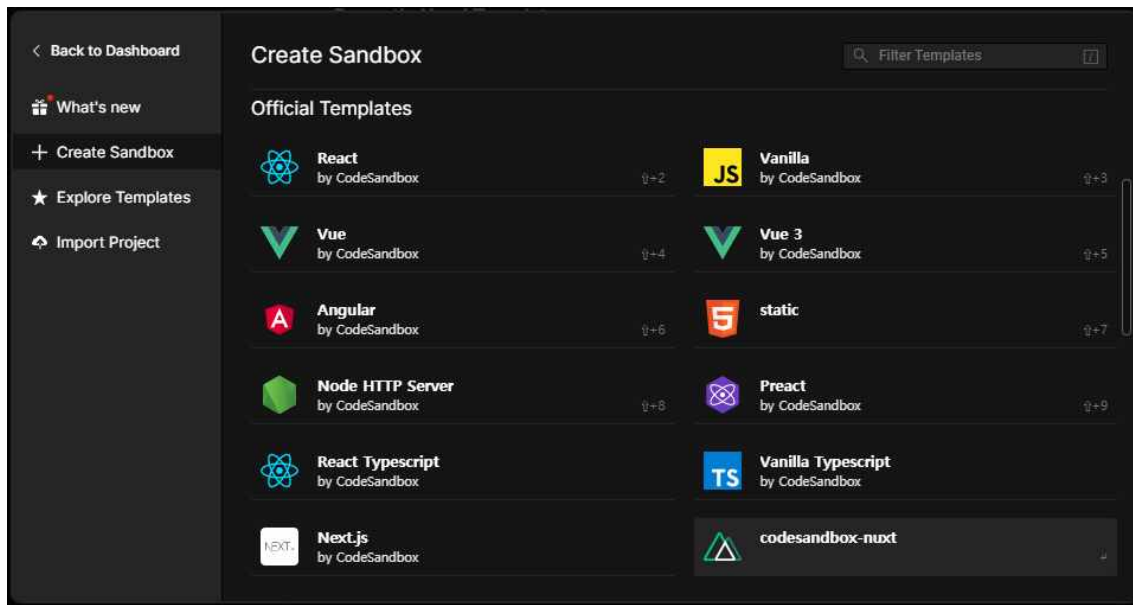
3번의 컴포넌트는 3장에서 바로 설명할것이다. 4번인 Hook은 지금 당장 무슨말인지 모르니깐 넘어가자.

이제 당장 Hello를 찍어보자. TypeScript 강좌와 마찬가지로, 따로 설치하는 작업 없이 CodeSandbox를 통해 작업할 것이다. 따라서, 크롬만 깔려있는 컴퓨터라도 당장 React를 배울 수 있다.

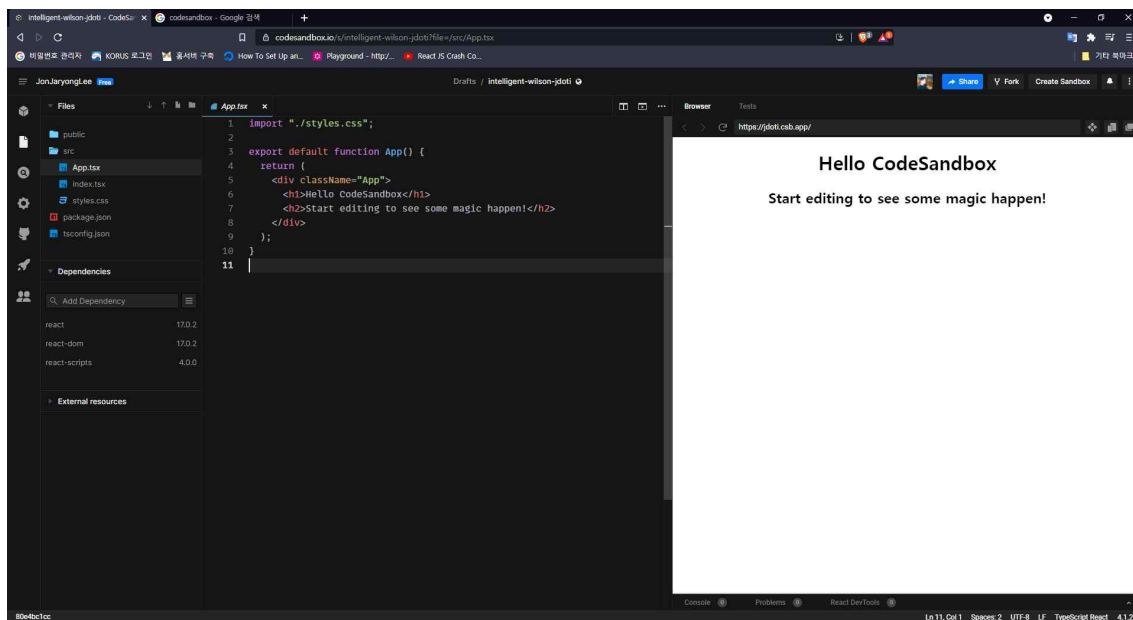
* sandbox는 사전에선 어린이들이 장난치는 모래놀이통을 의미한다. sandbox가 아닌, 거실에서 모래로 장난을 치면 집이 난장판이 될 것이다! 코딩에서 sandbox는 외부의 환경에서 격리된, 우리의 코드를 테스트하는 공간을 의미한다.



로그인하고 CodeSandbox 에 접속하면 다음과 같은 화면이 나온다. New Sandbox 를 클릭해보자.



원하는 환경을 선택할 수 있다. 우리가 찾는 건 React Typescript 다. 선택해주자.



다음과 같이, 당장 코딩을 시작할 수 있는 환경이 구축되었다. Hello가 찍히는 것을 볼 수 있다.

React Version: 17.0.2

만약 React의 버전이 17에서 18이상으로 바뀐다면, 이 강의 전체를 바꿔야할수도 있다.

* version에 대해 잠깐 설명하고 넘어가자. 앞으로 여러분이 개인적으로 공부하다보면, 버전이 바뀌었기 때문에 옛날의 지식들로는 더이상 작동하지 않는 상황이 벌어질수도 있다. 버전은 보통 세 부분으로 나뉜다.

17.0.2를 예로 들어보면,

첫번째 숫자(17)는 아주 큰 변화를 말한다. **함부로 업그레이드하면 에러날 가능성이 매우 크다.**

두번째 숫자(0)는 새 기능 추가다. 업그레이드를 하더라도 여러분의 코드는 정상작동할것이다. 새로 추가된 기능을 써도 되고 안 써도 된다.

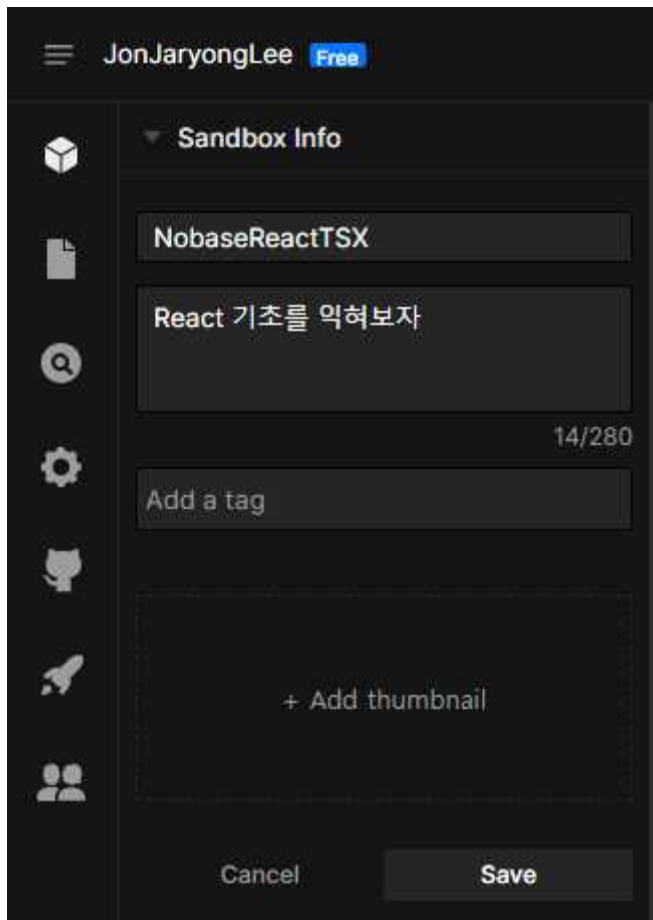
세번째숫자(2)는 버그 수정이다. 사용하는 입장에선 좋은 변화다.

추가로, 1.0.1 과 1.0.10 중 뭐가 더 최신버전일까? 정답은 1.0.10이다. 각 숫자는 소수가 아니라 자연수다. 1과 10의 비교라고 생각해야한다.

CodeSandbox 를 사용했을 때 장점은 다음과 같다.

1. 글자 하나를 치더라도 저장할 필요 없이 옆 테스트화면에서 즉각 반영된다. ctrl+s를 누르면 변경사항이 최종적으로 반영된다.
2. url을 통한 접속이 가능하다. 사이트 url은 sandbox 만들 때 랜덤으로 정해진다. 당장 휴대폰을 키고 테스트해보자.
3. 실습자 모두가 정해진 환경에서만 코드를 입력하기 때문에 결과가 같을 수밖에 없다.
4. sandbox 안에서만 놀기 때문에 내 컴퓨터와 아무런 관련이 없다.

우리 프로젝트 이름을 정해보자.



바꾼다고 해서 url이 바뀌는 건 아니기 때문에 필자와 똑같이 이름과 설명을 지어도 된다.

이제 본격적으로 React를 배워보기 전에, HTML 을 먼저 알아보자.

1. HTML

웹 세계에 성경이 있다면 이렇게 시작될 것이다.

창세기 1장 1절. 태초에 HTML 이 있었다.

1991년, 웹의 창조주 팀 버너스 리 (Sir Tim Berners-Lee) 가 처음 월드 와이드 웹(www)을 만들 땐 웹 세상엔 HTML 밖에 없었다. 이 사람은 이걸 학자들끼리 문서 교환하는 목적으로 만들었다. 그래서 .html 로 작성된 파일을 '문서(document)' 라고도 한다. 대인배였던 팀 버너스 리는 이 기술을 특허 등록없이 무료로 풀어버렸고, 덕분에 우리 인터넷이 지배하는 세상에 살게 되었다.

30년이 지난 지금까지도, HTML은 버전이 업그레이드되면서 그 문법은 변해왔더라도 여전히 브라우저에서 쓰이며, 대체제가 없다. 우리는 React 로 프론트엔드 개발을 할 것이지만, 결국 하나의 index.html 파일이 될 것이다. 즉, **React를 배우기 전에 반드시 HTML을 먼저 알아야 한다.** 그러나 다행스럽게도, HTML을 전부 다 배울 필요는 없고, 이 장에 있는 내용으로 React 를 배우기위한 준비를 하는 데엔 충분하다.

* index.html은 뭘 뜻하는걸까?

여러분이 나중에 서버에 여러분의 코드를 올리게 된다면, 주소를 쳤을 때 가장 먼저 보여지는 페이지가 바로 index.html 이다. 만약 파일명을 study.html 로 바꾸었다면, /study.html 로 쳐줘야 접속되지만, index.html 의 경우엔 그럴 필요가 없다. 즉, index.html 은 첫 페이지다.

HTML을 이루는 것은 태그(tag) 와, 태그 안의 속성(attribute) 이다. 많이 쓰이는 <a> 를 예로 들어보자.

```
<a href="https://조교행님.org">조교행님사이트로 가기!</a>
```

결과:

[조교행님사이트로 가기!](https://조교행님.org)

수업에서 < > 로 쓰였다면 HTML 태그를 의미한다.

<a> 는 링크를 나타내는 태그다. **태그는 열고 닫아야** 한다. 닫을때는 </태그이름> 으로 닫는다. 어떤 태그들은 안에 내용을 요구하지 않을수도 있는데, 그 경우엔 <태그이름 /> 식으로 하나만 쓴다.

속성은 태그를 조작한다. 여기서 쓰인 속성은 href="" 이며, 링크 사이트의 주소를 의미한다.

속성의 값(value)을 쓸 땐 큰따옴표("")를 사용한다.

클릭을 하면 속성 href 의 값에 해당하는 사이트로 접속한다.

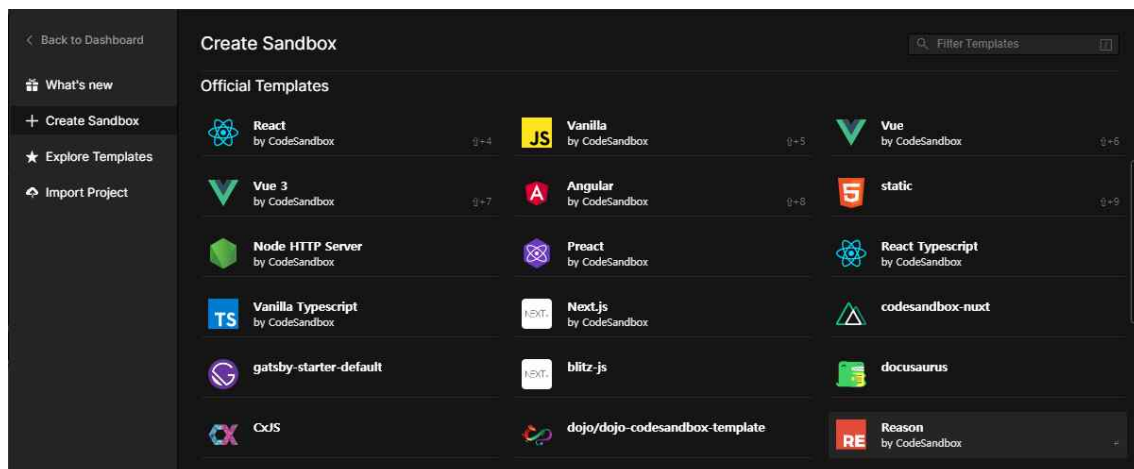
여러개의 속성을 쓸 때는 객체처럼 콤마(,) 를 쓰지 않고 띄워서 적는다.

각 태그마다 정해진 속성이 있으므로, 잘 모르겠다 싶을 때는 W3Schools, 또는 MDN 사이트에서 태그와 속성을 검색해보는것이 좋다.

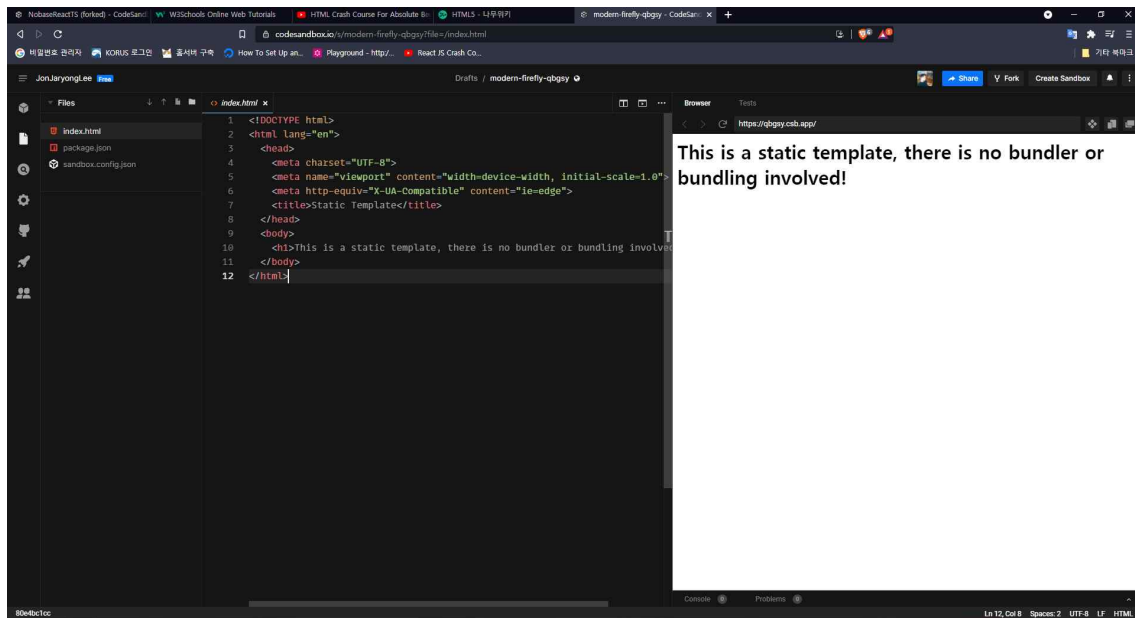
<https://www.w3schools.com/>

<https://developer.mozilla.org/ko/>

무슨말인지 모르겠으니깐, 실습을 통해 깨우쳐보기로 하자!



HTML 을 연습하기위해 sandbox 를 하나 만들것이다. static 클릭.



순수하게 HTML 만 연습해볼 수 있는 환경이다. 이걸 JavaScript 코드가 한 줄도 없기에, ctrl + s 를 눌러줘야 반영된다.

가장 기본적인 이 페이지를 분석해보자. 자세히보면 확장자가 .html 이다.



line1: HTML5 버전의 문서임을 알려준다. 5는 버전을 의미한다. 다른 버전은 알 필요 없다. 2014년 10월부터 2021년 6월인 지금까지 쪽 표준으로 쓰이고 있다.

line2 - 12: <html>. 안에 있는 것이 html 임을 뜻한다.

line3 - 8: <head>. 웹브라우저 본문에 표시되는 부분이 아니라면 <head> 에 들어간다. 인코딩방식, 표시영역, 설정, 제목 등등이 들어간다.

line4 - 6: <meta>. 해당 문서에 대한 정보 정의. <head> 안에 들어간다.

line7: <title>. **사이트의 제목**이다. 크롬 사이트 탭에 표시된다.

line9 - 11: <body>. 브라우저에 보여지는 내용이다.

line10: <h1>. 제목으로 쓰는 큰 글자다. <h1> <h2> <h3> ... <h6> 까지 있다.

결과:

Hello

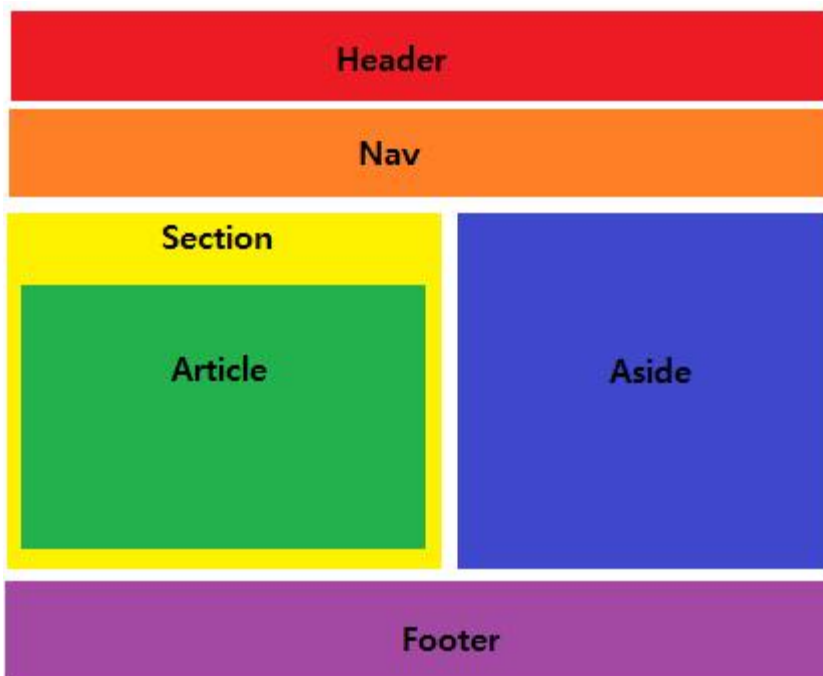
이제 간단한 연습들을 좀 해볼 것이다.

tip. 태그를 입력하려고 < 부터 입력할 필요 없이, h1을 치면 자동완성을 지원한다. 또한 매우 편리하게도, codesandbox 에선 ctrl + s 를 누르자마자 자동으로 줄맞춤을 예쁘게 해준다.

지금부터 나올 태그들은 모두 <body> 안에 해당된다.

1. 시맨틱 태그와 <div>

시맨틱 태그는 순전히 개발자를 위한 태그다. 컴퓨터와는 상관없고, 우리가 읽기 편하라고 쓰는 태그들이다.



header 화면의 상단에 위치하는, .html 파일의 전체적인 정보를 정의(head와 혼동 주의)

nav 문서의 네비게이션 항목을 정의. 웹사이트의 상단 탭, 좌측 탭 등에서 누르면 그 페이지로 가는 부분을 말한다.

section 문서의 구간을 나눌 때 사용.

article 본문

aside 광고와 같이 페이지의 내용과는 관계가 적은 내용들

footer 화면의 하단에 위치하는, 사이트나 문서의 전체적인 정보를 정의.

개인정보처리방침, 약관, 회사이름 등이 들어가는 부분.

코드를 통해 알아보자.

```
9      <body>
10      <header>
11          <h1>내 소개</h1>
12      </header>
13      <nav>
14          <ul>
15              <li>나는 누구인가</li>
16              <li>내 가족</li>
17              <li>내 직장</li>
18          </ul>
19      </nav>
20      <section>
21          <article>
22              나는 조교행님일세
23          </article>
24      </section>
25      <section>
26          <article>
27              나는 혼자 산다네
28          </article>
29      </section>
30      <section>
31          <article>
32              나는 조교라네
33          </article>
34      </section>
35      <aside>짜장면 단돈 3천원</aside>
36      <footer>all rights reserved</footer>
37  </body>
```

line14 - 18: 은 리스트형 정보를 나타낼 때 쓰인다. 내부에 각 리스트 항목을 로 표현한다.

결과:

내 소개

- 나는 누구인가
- 내 가족
- 내 직장

나는 조교행님일세
나는 혼자 산다네
나는 조교라네
짜장면 단돈 3천원
all rights reserved

엥? 뭔가 잘못된거같다. 분명 위의 그림처럼 예쁘게 나와야하지 않는가? 하지만 HTML은 디자인을 담당하지 않기 때문에 정상적으로 나온 게 맞다. 나중에 CSS라는 언어를 배워서 직접 스타일링을 하게 되면 위의 레이아웃대로 만들 수 있을 것이다.

다른 태그도 아니고 왜 시맨틱 태그를 맨 처음 소개했을까? 여러분이 사용하는 습관을 들여놓는것을 권장하기 때문이다. 이것은 쉽게 말하면 주석같은 것이다. 쓸 때는 귀찮아도 막상 써놓으면 나중에 편하고, 읽기도 쉽다.

가독성이 나빠지는 정반대의 경우를 들어보면, 오로지 <div> 만 사용하는 예가 있다. <div>는 React 개발하면서 여러분들이 가장 많이 쓰게 될 태그인데, 영어단어로는 division을 의미하고, 쉽게 말하면 '영역'이다. 주로 className 속성과 같이 쓰인다.

* className 은 태그의 이름표라고 생각하면 된다. <div> 뿐만 아니라 모든 태그에 적용 가능한 속성이다.

시맨틱 태그로 잘 정리된 내용을 <div className=""> 만으로 변경해보자.

```

9    <body>
10   <header>
11     <h1>내 소개</h1>
12   </header>
13   <nav>
14     <ul>
15       <li>나는 누구인가</li>
16       <li>내 가족</li>
17       <li>내 직장</li>
18     </ul>
19   </nav>
20   <section>
21     <article>
22       나는 조교형님일세
23     </article>
24   </section>
25   <section>
26     <article>
27       나는 혼자 산다네
28     </article>
29   </section>
30   <section>
31     <article>
32       나는 조교라네
33     </article>
34   </section>
35   <aside>짜장면 단돈 3천원</aside>
36   <footer>all rights reserved</footer>
37 </body>

```



```

9    <body>
10   <div className="header">
11     <h1>내 소개</h1>
12   </div>
13   <div className="nav">
14     <ul>
15       <li>나는 누구인가</li>
16       <li>내 가족</li>
17       <li>내 직장</li>
18     </ul>
19   </div>
20   <div className="section">
21     <div className="article">
22       나는 조교형님일세
23     </div>
24   </div>
25   <div className="section">
26     <div className="article">
27       나는 혼자 산다네
28     </div>
29   </div>
30   <div className="section">
31     <div className="article">
32       나는 조교라네
33     </div>
34   </div>
35   <div className="aside">짜장면 단돈 3천원</div>
36   <div className="footer">all rights reserved</div>
37 </body>

```

어떤가, 확실히 보기 어려워졌지 않는가? 따라서, 지금부터라도 시맨틱 태그를 쓰는 습관을 들이도록 하자.

2. React 강의에 쓰일 태그들

HTML엔 정말 많은 태그가 있지만, 배워도 까먹기 마련이다. 당장 강의에서 쓰는 태그들만 익혀두고, 다른 태그들이 필요하다면 구글링해서 찾아 쓰도록 하자.

a.

이미지를 삽입하는 태그이다.

```

10    

```

line11 - 12: height는 높이, width는 너비를 의미한다. 둘 다 지정해줄 경우, 보통은 그림이 늘어져서 보기 안 좋으므로 둘 중 하나만 쓴다.

line13: src. 이미지의 링크다. 지금은 url로 걸었는데, 프로젝트 디렉토리 내부 public 디렉토리에 저장하고 가져와 쓸 수도 있다.

line14: alt. 혹시나 서버에서 문제가 생겨 이미지를 못 가져올 경우, 이미지를 대체하는 글자다. url을 틀리게 적어보아 확인해보자.

line15: title. 마우스를 올렸을 때 뜨는 글자다.

결과:



b. <p>, <input>

사용자로부터 입력을 받을 때 쓰는 태그다.

```
17      <p>
18          <input type="text" onchange="" placeholder="입력하세요" />
19      </p>
```

line17 - 19: <p> 태그는 paragraph의 약자로서, 단락을 구분할 때 쓰인다.

line18:

type="text"	문자열 입력을 의미한다. text 말고도 checkbox, radio, number, password 가 주로 쓰인다.
onChange=""	입력된 정보가 바뀔때 작동하는 이벤트 핸들러다.
placeholder=""	아무것도 입력되지 않았을 때 뜨는 글자를 말한다.

* 이벤트(event)는 브라우저에서 일어나는 사용자의 행동을 말한다. 클릭, 드래그, 스크롤, 타이핑, 화면 사이즈 줄이기 등 여러분이 상상할 수 있는 모든 동작은 이벤트다.

* on- 이 붙으면 이벤트 핸들러(event handler)를 의미한다. 이벤트 발생 시 동작하는 함수를 말한다.

결과:

c. <button>

```
21 <button onclick="">클릭할거야?</button>
```

말 그대로 버튼이다. onClick은 클릭했을 때 실행될 이벤트 핸들러를 말하며, <button> 뿐만아니라 다른 태그에서도 쓸 수 있다.

결과:

클릭할거야?

d. <a>

링크를 의미한다.

```
23 <a href="https://github.com/GNUCSAssistants">조교실 깃허브 가기</a>
```

href는 아까 설명했듯이, 이동할 페이지의 주소이다.

결과:

[조교실 깃허브 가기](https://github.com/GNUCSAssistants)

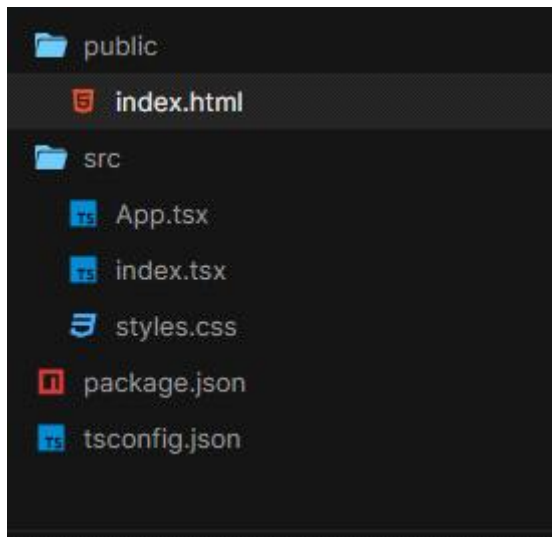
좋다. HTML은 이것으로 충분하다.

그럼 이런 질문이 들 것이다. 대체 왜 HTML을 배운것인가? 이 수업은 React 수업이지않은가.

맞다. 하지만 React 전에 HTML 을 알아야 할 두가지 중요한 이유가 있다.

1. React 에선 JSX라는, JavaScript 와 HTML을 결합한 문법을 사용한다.
2. React는 index.html 파일의 한 부분일 뿐이다.

첫번째는 3장인 component 에서 바로 배우게 될 것이다. 두번째는 직접 코드로 확인해보자. 맨 처음에 만들어두었던 React TypeScript SandBox 로 가보자.



왼쪽 파일 네비게이션에서 `public - index.html` 로 들어가보자.


```

1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8" />
5     <meta
6       name="viewport"
7       content="width=device-width, initial-scale=1, shrink-to-fit=no"
8     />
9     <meta name="theme-color" content="#000000" />
10    <!--
11      manifest.json provides metadata used when your web app is added to the
12      homescreen on Android. See https://developers.google.com/web/fundamentals/
13    -->
14    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
15    <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico" />
16    <!--
17      Notice the use of %PUBLIC_URL% in the tags above.
18      It will be replaced with the URL of the 'public' folder during the build.
19      Only files inside the 'public' folder can be referenced from the HTML.
20
21      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
22      work correctly both with client-side routing and a non-root public URL.
23      Learn how to configure a non-root public URL by running 'npm run build'.
24    -->
25    <title>React App</title>
26  </head>
27
28  <body>
29    <noscript>
30      You need to enable JavaScript to run this app.
31    </noscript>
32    <div id="root"></div>
33    <!--
34      This HTML file is a template.
35      If you open it directly in the browser, you will see an empty page.
36
37      You can add webfonts, meta tags, or analytics to this file.
38      The build step will place the bundled scripts into the <body> tag.
39
40      To begin the development, run 'npm start' or 'yarn start'.
41      To create a production bundle, use 'npm run build' or 'yarn build'.
42    -->
43  </body>
44 </html>

```

우리의 React 프로젝트는 최종적으로 이 코드로 보여지게 된다. 이 하나의 html 파일이 우리 눈에 보이는 것이다. title을 "노베이스 React 웹개발" 같은 것으로 한번 바꿔보자.

* 아래 내용은 여러분에게 이제까지 가르치지 않은 부분이 등장하기에, 우리의 React 코드가 index.html 과 무슨 관련이 있는지 궁금해서 미치는 학생들을 위한 부분이다. 잘 작동하는것으로 만족하는 학생이라면 넘어가도 좋다.

그런데 여기엔 React 코드가 하나도 없다. 우리가 눈여겨봐야할 부분은 line32, `<div id="root">` 라고 적힌 부분이다. 이게 대체 뭘 의미하는걸까?

다시 파일 네비게이션에서, src - index.tsx 파일을 열어보자. .tsx 는 TypeScript 를 사용하는 React 파일을 의미한다.

```
index.tsx
1  import { render } from "react-dom";
2
3  import App from "./App";
4
5  const rootElement = document.getElementById("root");
6  render(<App />, rootElement);
```

원말인지 모를 코드는 뒤로하고, line5에 root 가 보이지 않는가? 이 문서에서 id가 root인 놈의 정보를 rootElement라는 변수에 담는다. 그 변수는 line6에서 `<App>` 이라는 태그와 함께 `render()` 안으로 들어가게 된다. `<App>` 의 정체는 line3에 나온다. App.tsx 파일이다.

우리가 코딩하게 될 파일은 index.tsx 도, index.html 도 아니다. index.tsx 는 건드릴 일이 없을 것이며, index.html 에선 꼭해봤자 사이트 아이콘이나 제목 바꾸는 정도의 작업만 할 것이다. 우리는 src 안의 App.tsx 에서 코딩을 시작할 것이다.

다시 처음으로 돌아와서, 우리가 코딩하는 부분은 index.html에서 어디인가?

```
28  <body>
29    <noscript>
30      You need to enable JavaScript to run this app.
31    </noscript>
32    <div id="root"></div>
33    <!--
34      This HTML file is a template.
35      If you open it directly in the browser, you will see an empty page.
36
37      You can add webfonts, meta tags, or analytics to this file.
38      The build step will place the bundled scripts into the <body> tag.
39
40      To begin the development, run `npm start` or `yarn start`.
41      To create a production bundle, use `npm run build` or `yarn build`.
42    -->
43  </body>
```

바로 line32번, `<div id="root"></div>` 부분이다. 여러분들이 몇천줄, 몇만줄의 React 코드를 작성해도, 이 한줄 안에 있다.

핵심정리

React를 배우기 전에 반드시 HTML을 먼저 알아야 한다.

HTML 은 태그와 속성으로 이루어져 있다.

태그는 열고 닫아야 한다.

속성은 태그를 조작한다.

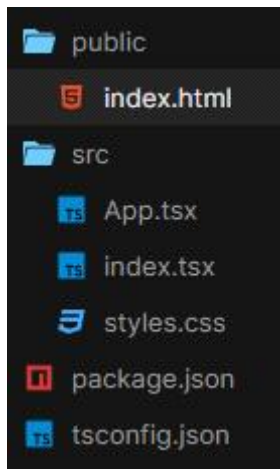
React에선 JSX라는, JavaScript 와 HTML을 결합한 문법을 사용한다.

React는 index.html 파일의 한 부분일 뿐이다.

2. React 프로젝트 디렉토리 구조

본격적인 React 수업에 앞서, 우리 프로젝트 디렉토리 구조를 파악해보자.

* 이 부분은 기술에 대한 설명이 많이 나와서 좀 어려울 수 있다. 우리 프로젝트가 어떤 식으로 돌아가는지 궁금한 학생들이 읽으면 좋을 것 같다.



public: 여러분이 개발하면서 쓸 이미지, 파비콘(로고) 같은 것들을 저장해두는 곳이다.
그리고 이 안에 index.html 이 있다.

src: source code 의 줄임말이다. 여러분이 코딩할 파일들이 들어간다.

App.tsx: 본격적인 React 개발의 시작이 될 파일이다.

index.tsx: App.tsx와 index.html 을 연결해주는 파일이다.

styles.css: 웹을 디자인할때 쓰는 css 파일이다.

css 는 이번 수업에서 다루지 않는다.

package.json: 우리 프로젝트에 대한 정보를 담은 파일이다.

tsconfig.json: TypeScript 설정파일이다. 이 파일이 있어야 TypeScript 가 JavaScript 로 컴파일된다.

* 컴파일(compile) 은 쉽게 말하면 번역이다. 브라우저는 TypeScript를 못 알아듣기때문에 JavaScript로 번역하는 과정이 꼭 필요하다. 또한 우리의 React 코드들도 결국

JavaScript로 컴파일될 것이다.

이 중에서 심도깊게 살펴봐야할 것이 있는데, package.json 파일이다. 이 파일 안에는 우리 프로젝트에 대한 정보가 담겨있다.

```
1 {
2   "name": "react-typescript",
3   "version": "1.0.0",
4   "description": "React and TypeScript example starter project",
5   "keywords": [
6     "typescript",
7     "react",
8     "starter"
9   ],
10  "main": "src/index.tsx",
11  "dependencies": {
12    "react": "17.0.2",
13    "react-dom": "17.0.2",
14    "react-scripts": "4.0.0"
15  },
16  "devDependencies": {
17    "@types/react": "17.0.0",
18    "@types/react-dom": "17.0.0",
19    "typescript": "4.1.3"
20  },
21  "scripts": {
22    "start": "react-scripts start",
23    "build": "react-scripts build",
24    "test": "react-scripts test --env=jsdom",
25    "eject": "react-scripts eject"
26  },
27  "browserslist": [
28    ">0.2%",
29    "not dead",
30    "not ie <= 11",
31    "not op_mini all"
32  ]
33 }
```

프로젝트 설명

패키지 dependencies

npm 명령어

npm 명령어 부분에서, start 와 build 는 나중에 VSCode 로 작업하게되면 많이 쓰게 될 것이다.

start: node.js 서버 실행

build: 빌드. 프로젝트를 html, css, javascript로 컴파일하는것을 말한다.

* node.js 는 브라우저 바깥에서 작동하는 JavaScript 다. 2009년 node.js 가 탄생되기 이전까지의 JavaScript는 오로지 브라우저에서만 작동했으나, node.js 이후 브라우저를 벗어나 엄청난 성장을 하게 되고, 메이저 언어의 위치를 차지한다.

* npm은 node package manager 의 약자로, 전 세계 JavaScript 패키지 저장소이다. 패키지는 라이브러리라고 생각하면 된다.

가장 중요한것은 dependency 항목이다. 한국어로 '의존성'이라고 번역된다. 대체 무엇을 말하는 것일까?

중국집을 떠올려보자. 중국집만 딱하니 있다면 장사가 가능할까? 납품 오는 채소업자, 장부를 관리해주는 회계사, 안에서 일하는 주방장, 근처에 있는 식자재마트, 오토바이 배달원 ... 동네 중국집만해도 엄청나게 많은 관계가 엉켜있다. 사장이 혼자 청경채 키우고, 세금 관리하고, 주방에서 칼질하고, 마트도 직접 운영하고, 손님 먹는동안 배달가고 하다보면 금방 망할 것이다! 우리 프로젝트도 마찬가지다. JavaScript만으로 개발하긴 정말 힘들다. 우리 다른 사람이 만들어둔 코드를 가져다 써야 한다.

dependency는, 이 프로젝트가 정상작동하기위해 필요한 npm 패키지 목록이다. 만약 다른 사람이 이 프로젝트 코드를 다운로드받으면, 적어도 이게 있어야 실행된다고 적어놓은 것이다.

이렇게 적어놓으면 뭐가 좋을까?

첫번째로, 이 프로젝트가 작동하려면 어떤 패키지가 필요한지 적어놓았으니 해당 패키지를 깔아야만 앱이 작동함을 알 수 있다. 중국집을 다른 사람에게 인수인계할때 새 주인이 알아야 할 정보들을 미리 적어둔 것이다.

두번째로, 다른 사람에게 내 코드를 보낼 때 모든 패키지를 다 보낼 필요가 전혀 없다. 지금 적혀있는 패키지는 여섯개밖에 안되지만, 저 여섯개를 설치한다고해서 딱 여섯개만 깔리진 않는다. react만 설치해도, 'react의 dependency' 까지 한꺼번에 깔리기때문에 용량이 어마어마해서 비효율적이다. 그러나, 패키지 이름만 적어놓으면 상대방이 명세서를 받고, npm install 명령어만 쳐서 필요한걸 나한테서가 아닌, npm 에서 받을 수 있다.

그러면 devDependencies 는 뭐가 다른걸까? 이것은 개발 및 테스트용 패키지이며, 실제 웹으로 배포(deploy)할때는 필요없는 패키지이다. 자세히보면 TypeScript 관련 패키지들인데, 개발중엔 당연히 TypeScript 를 번역하기위해 필요하지만, 배포할때는 이미 JavaScript이기 때문에 필요없다.

* package.json 을 개발할때마다 따로 작성해줘야될까? 그럴 필요 없다. 필요한 패키지를 npm 명령어로 install 하면, 자동으로 기록된다.

여기까지, 우리 프로젝트가 어떻게 생겼는지 알아보았다. 이제 본격적으로 React에 대해 배워보자.

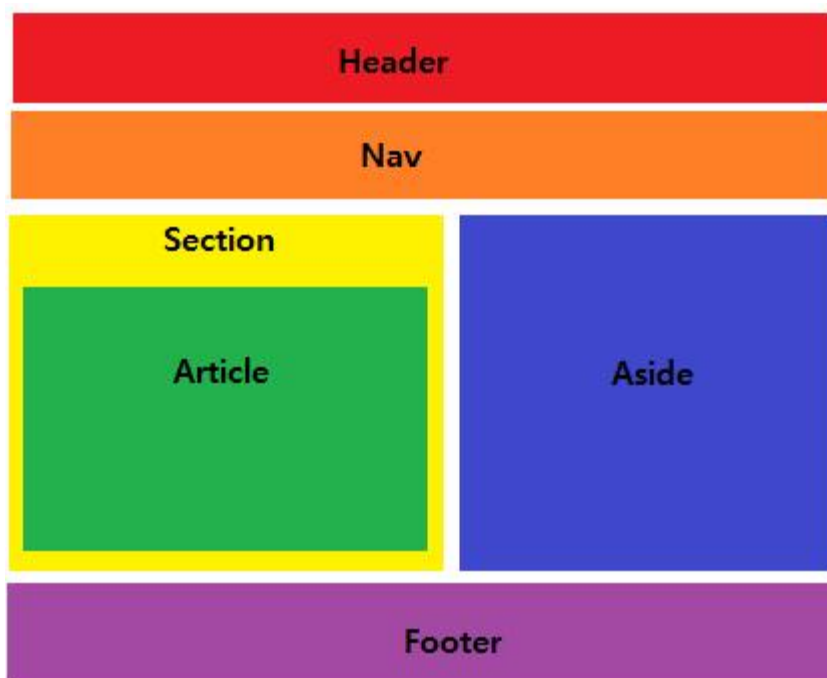
3. 컴포넌트(component)

React에서 주요한 기술은 딱 두가지로 요약된다.

1. component
2. hook

그 중 첫번째인 컴포넌트에 대해 알아보자. 컴포넌트란 무엇일까? 영어사전에선 구성요소, 부품이라고 번역된다.

그렇다. 컴포넌트는 부품이다. **웹페이지를 조갠 것이다.**



HTML 시간에 배웠던 그림을 그대로 가져와보았다. 시맨틱 태그를 설명했었는데, 각각을 컴포넌트로 만들 계획이라고 치자.

1. 네모는 모두 컴포넌트다.
2. 컴포넌트 안에 컴포넌트가 있을수도 있다. (Section 밑에 Article)
3. **컴포넌트는 함수다.**

Aside 부분을 예로 들어보자. 저 부분은 웹페이지에서 광고를 나타낼 영역이다. Aside 영역의 광고를 담당할 Advertisement 컴포넌트를 만들고, 광고 내용을 수정할 일이 있다면 **다른 파일 건드릴 필요 없이** Advertisement 컴포넌트를 수정하면 된다.

그리고 Advertisement 컴포넌트를 이미 만들어놓았기 때문에, 다른 곳에 쓸 일이

있을때마다 **재사용**할수있다. 광고부분을 매번 작성해야 할 필요가 없는 것이다.

이제 실제 코드를 살펴보자. 이론적인 부분이니, 천천히 봐야한다.

App.tsx

```
App.tsx
1  import "../styles.css";
2
3  export default function App() {
4    return (
5      <div className="App">
6        <h1>Hello</h1>
7      </div>
8    );
9  }
10
```

line1: import 는 App.tsx 파일에 "../styles.css" 파일을 집어넣겠다는 뜻이다. 이 파일 밖에 있는 코드를 쓰기 위해선 이렇게 import 해줘야된다. 여기서 디자인을 담당하는 css파일이 들어갔다.

line3: export 는 내보낸다는 뜻이며, import의 반댓말이다. App() 이라는 컴포넌트를 바깥으로 내보내겠다는 뜻이다. 옆에 default 가 있는데, 이것은 App.tsx 전체에서 내보낼 변수나 함수가 딱 이것밖에 없을 때 쓰인다.

* JavaScript 파일 각각을 모듈(module) 이라고 부른다.

다른 모듈을	쓰고싶으면	import 해야된다.
다른 모듈에서	쓰게 만들고 싶으면	export 해야한다.

line4 - 7: return 값이다. 그런데 좀 이상하지 않은가? JavaScript 안에 HTML 이 있다. 여러분이 만난 첫번째 React 문법이다. 이것은 JavaScript 도 아니고, TypeScript 도 아니고, HTML 도 아니다. JSX라는, React 고유의 문법이다. **JavaScript 안에 있는 HTML** 이라고 생각하면 될 것 같다.

여기서 컴포넌트에 대한 좀 더 정확한 정의를 알 수 있다. **컴포넌트는 HTML을 리턴하는 함수**이다.

이제까지 나온 컴포넌트에 대한 정의를 다시 정리해보자.

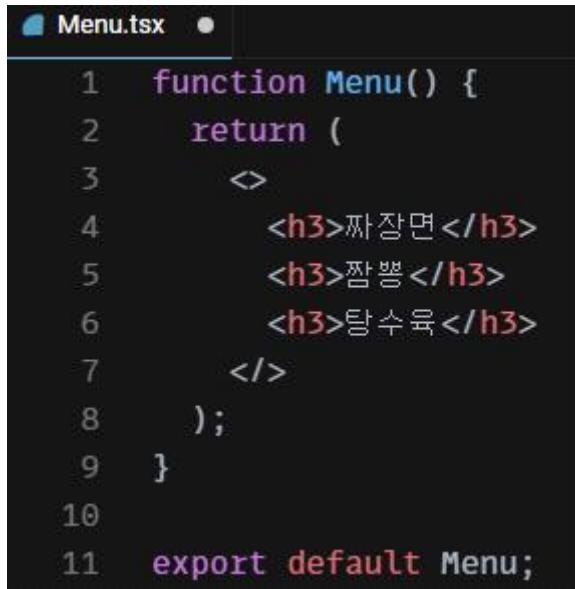
알아듣기 쉬운 설명: 웹페이지를 쏴낸 것

좀 더 정확한 정의: 웹사이트의 부분을 담당하는 함수

React에서 엄격한 정의: HTML을 리턴하는 함수

이제 우리의 컴포넌트를 만들어서 App.tsx 에 집어넣어보자. 여러분이 직접 작성해보는 첫 React 코드다. 파일명은 Menu.tsx 라고 짓고, src 디렉터리 안에 만들겠다.

Menu.tsx



```
1 function Menu() {
2   return (
3     <>
4       <h3>짜장면</h3>
5       <h3>짬뽕</h3>
6       <h3>탕수육</h3>
7     </>
8   );
9 }
10
11 export default Menu;
```

line1: export default 로 쓰일 컴포넌트는 파일명과 동일하게 함수로 선언한다.

line2 - 8: 여러줄 리턴할때는 소괄호() 로 감싸준다.

line3 - 7: return에서 여러 줄의 태그를 쓸 때는 다음과 같이 빈 태그 <> </> 를 넣어줘야 에러가 나지 않는다.

line11: export default 는 이와 같이 맨 아랫줄에 쓸 수도 있다. line1에 써도 상관없이 잘 동작한다.

이제 App.tsx 에서 Menu.tsx 를 import 해보자.

tip. 대부분의 에디터는 화면분할을 지원한다. CodeSandbox 도 마찬가지다.


```
App.tsx
1 import Menu from "./Menu";
2 import "./styles.css";
3
4 export default function App() {
5   return (
6     <div className="App">
7       <h1>메뉴</h1>
8       <Menu />
9     </div>
10   );
11 }
12

Menu.tsx
1 function Menu() {
2   return (
3     <>
4       <h3>짜장면</h3>
5       <h3>짬뽕</h3>
6       <h3>탕수육</h3>
7     </>
8   );
9 }
10
11 export default Menu;
```

line1: 우리의 모듈을 import 한다.

모듈을 import 할 때는 확장자를 떼고 적는다.

'.'은 현재 파일인 App.tsx의 위치를 뜻한다. Menu.tsx 는 어디있는가? App.tsx와 같은 디렉터리인 src 에 위치한다. 풀어서 썼을 때 모듈의 경로는 src/Menu 이다.

line8: **컴포넌트 Menu를 HTML 태그처럼 사용했다.** 컴포넌트는 태그처럼 < /> 로 작성되는데, 첫 글자 대문자로 쓰인다.

* 파란색 배경으로 칠한 부분을 보자. 언뜻 보면 Menu.tsx 의 Menu() 컴포넌트를 가져온 것처럼 보이지만, 사실 파란색 배경으로 칠한 Menu는 우리가 지은 이름일 뿐이다. 궁금하면 파란색 배경 두 부분의 Menu를 Tomato 로 바꿔보자. 잘 작동한다. 왜일까? export default 를 사용해 단 하나의 변수 또는 함수만 export 하는 모듈은 import 하는 순간엔 이름이 없기 때문이다. 이 상황이 아니라, 만약 여러 개 변수 또는 함수를 export 한다면, import 시에 해당 변수 또는 함수의 이름을 정확하게 써줘야 한다.

결과:

메뉴

짜장면

짬뽕

탕수육

다음과 같은 결과가 나왔다. 아래 그림을 보면서, 확실히 이해했는지 정리해보자.

```
<div className="App">
  <h1>메뉴</h1>
  <Menu />
</div>
```



```
<div className="App">
  <h1>메뉴</h1>
  <h3>짜장면</h3>
  <h3>짬뽕</h3>
  <h3>탕수육</h3>
</div>
```



이제 컴포넌트를 다른 방식으로 만들어서 똑같은 결과를 내어볼 것이다. Menu.tsx 라는 파일을 따로 만들지 않고, App.tsx 에 직접 컴포넌트를 작성한다.

```

App.tsx
1  import "../styles.css";
2
3  function Menu() {
4    return (
5      <>
6        <h3>짜장면</h3>
7        <h3>짬뽕</h3>
8        <h3>탕수육</h3>
9      </>
10   );
11 }
12
13 export default function App() {
14   return (
15     <div className="App">
16       <h1>메뉴</h1>
17       <Menu />
18     </div>
19   );
20 }

```

두 가지 방식 중에 적절한 방식은 상황에 따라 다르다. React 개발자는 개발하면서 하나의 컴포넌트를 같은 파일에서 분리하거나, 컴포넌트 안에 컴포넌트를 두거나, 아예 다른 모듈을 만들어 import 해서 쓰기도 한다. 개발을 하게 되면서 자연스럽게 적절한 상황을 깨우치게 되겠지만, 하나만 기억하고 가자. **하나의 컴포넌트는 하나의 역할만 해야 하며, 이름에서 그 역할을 알 수 있어야 한다.**

지금 당장 컴포넌트의 힘을 느끼긴 어렵다. 수천줄, 수만줄을 개발할 때, 컴포넌트 없이 개발을 한다면 굉장히 코드가 길어질 것이다. 반복적인 부분을 반복해서 써줘야하기 때문이다.

핵심정리

컴포넌트란?

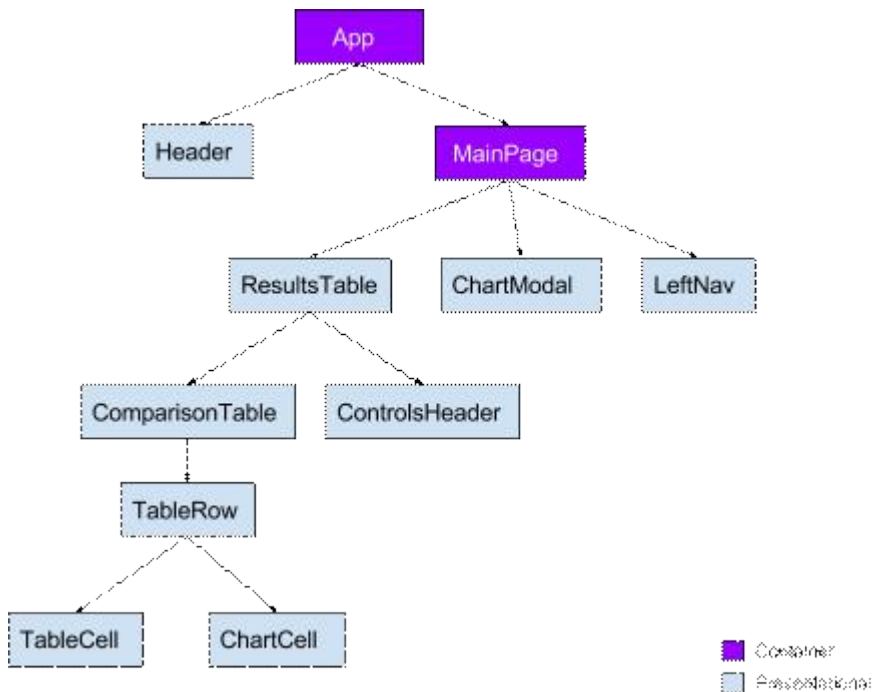
알아듣기 쉬운 설명: 웹페이지를 쪼갠 것

좀 더 정확한 정의: 웹페이지의 부분을 담당하는 함수

React에서 엄격한 정의: html을 리턴하는 함수
 컴포넌트를 사용했을 때 장점:
 코드 수정이 필요할 때 해당 컴포넌트만 수정할 수 있다.
 만들어둔 컴포넌트를 재사용할 수 있다.
 JavaScript 파일 각각을 모듈(module) 이라고 부른다.
 다른 모듈을 쓰고 싶으면 import 해야 된다.
 다른 모듈에서 쓰게 만들고 싶으면 export 해야 한다.
 JSX는 React 고유의 문법이며, JavaScript 안에 있는 HTML 이다.
 import 된 컴포넌트는 HTML 태그처럼 사용한다.
 하나의 컴포넌트는 하나의 역할만 해야 하며, 이름에서 그 역할을 알 수 있어야 한다.

4. props

컴포넌트가 딱 하나 있는 웹앱을 만들 생각이라면 굳이 React 를 쓸 이유가 없다. 하나의 웹앱에는 적게는 여러개, 많게는 수십 수백개의 컴포넌트가 들어갈 수 있다.



컴포넌트 구성의 예를 하나 가져와보았다. 이런 형태를 나무처럼 생겼다고해서 트리(tree)라고 한다. 하나의 부모 컴포넌트는 여러개의 자식 컴포넌트를 가질 수 있다.
 예제 이미지를 분석해보면, 크게 헤더와 메인페이지로 나뉘고, 메인페이지는 다시 결과테이블, 차트모델, 왼쪽 네비게이션으로 나뉜다. 결과테이블은 다시 다른

자식컴포넌트를 가진다. 여러분들도 여러분만의 웹앱을 개발하게 되면, 코딩하기전에 UI를 미리 만들어놓고, 다음과 같은 컴포넌트 트리를 미리 그려놓고 개발하게 될 것이다. 중요한 건, 컴포넌트간 데이터의 흐름이다. 여러분은 백엔드에 원하는 데이터를 요청하게 될 것인데, 그 데이터는 어디서 관리하는게 좋을까? **데이터는 최상단 컴포넌트인 App 에서 관리해야 한다.** 그렇게 안하면 작동 안되는것이 아니라, React 개발에서 권장되는 방식이다. 즉, 모든 데이터는 App 에서 관리해야하며, 받은 데이터를 자식에게 넘겨줘서 App의 데이터와 자손의 데이터가 일치하도록 만들어야한다.

이것을 구현하기위해 props 를 사용해볼 것이다.

우선 자식 컴포넌트가 필요하다. src 디렉터리 밑에 Student 컴포넌트를 하나 만들자. 확장자는 .tsx

Student.tsx

```
1  function Student() {
2    return (
3      <>
4        <h1>Student</h1>
5      </>
6    );
7  }
8
9  export default Student;
```

다음과 같은 컴포넌트를 만들고,

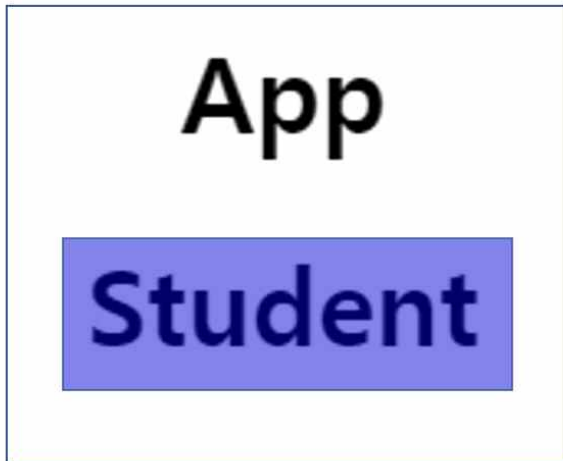
App.tsx

```
1  import Student from "./Student";
2  import "./styles.css";
3
4  export default function App() {
5    return (
6      <div className="App">
7        <h1>App</h1>
8        <Student />
9      </div>
10   );
11 }
```

line1 에서 임포트하고

line8 에서 불린다.

결과:



App 컴포넌트 안에 Student 컴포넌트가 생겼다.

우리의 목표는 App 의 데이터를 자식인 Student 로 내려보내는 것이다.

```
App.tsx
1  import Student from "../Student";
2  import "../styles.css";
3
4  export default function App() {
5    return (
6      <div className="App">
7        <h1>App</h1>
8        <Student
9          name="조교행님"
10         age={18}
11         isGirlfriend={true}
12         foo={[1, 2, 3, 4]}
13       />
14     </div>
15   );
16 }
```

line8 - 12 를 잘 보자. 이제 데이터를 내려보낼 것이다. 총 다섯 종류를 내려보낸다. 이름, 나이, 여자친구 유무, 그리고 넘버 어레이. string의 경우엔 쌍따옴표를 써주면

되고, 나머지 자료형엔 중괄호로 감싸주면 된다.

지금은 Student 태그 안에 데이터를 바로 써줬는데, 이미 있는 데이터를 쓰는 경우라면?

```
App.tsx x
1  import Student from "./Student";
2  import "./styles.css";
3
4  const myInfo = {
5    name: "조교행님",
6    age: 18,
7    isGirlfriend: true,
8    foo: [1, 2, 3, 4]
9  };
10
11 export default function App() {
12   return (
13     <div className="App">
14       <h1>App</h1>
15       <Student
16         name={myInfo.name}
17         age={myInfo.age}
18         isGirlfriend={myInfo.isGirlfriend}
19         foo={myInfo.foo}
20       />
21     </div>
22   );
23 }
```

이런 식으로 쓰면 된다.


```

Student.tsx x
1  interface StudentInfo {
2      name: string;
3      age: number;
4      isGirlfriend: boolean;
5      foo: number[];
6  }
7
8  function Student(props: StudentInfo) {
9      console.log(props);
10     return (
11         <>
12             <h1>Student</h1>
13         </>
14     );
15 }
16
17 export default Student;

```

line8 에서와 같이, 받을때는 한꺼번에 props로 받는다. prop 은 property의 줄임말이며, s를 써서 복수이다. 여기서 props는, 하나의 객체라고 이해해야한다. 그래서 인터페이스를 꼭 써줘야한다.

타입스크립트 강좌때 인터페이스의 각 프로퍼티를 구분하는것을 콤마로 썼는데, 세미콜론을 찍어도 잘 동작한다. 둘다 잘 동작하는것으로 확인되는데, 공식문서 읽어보니깐 세미콜론이 표준인것으로 확인된다.

line9에서 콘솔로그를 찍어보면 다음과 같은 결과가 나온다.

```

▼ {name: "조교행님", age: 18, isGirlfriend: true, foo: Array(4)}
  name: "조교행님"
  age: 18
  isGirlfriend: true
  ▼ foo: Array(4)
    0: 1
    1: 2
    2: 3
    3: 4

```

만약, props 로 전체를 받지 않고 프로퍼티 중 하나만 받으려면 중괄호를 이용해 다음과

같이 한다.

```
Student.tsx •
1  interface StudentInfo {
2      name: string;
3      age: number;
4      isGirlfriend: boolean;
5      foo: number[];
6  }
7
8  function Student({ age }: StudentInfo) {
9      console.log(age);
10     return (
11         <>
12             <h1>Student</h1>
13         </>
14     );
15 }
16
17 export default Student;
```

중괄호 안에 age를 써서 age만 받는 모습. 옆에 인터페이스 타입은 그대로 둔다.

콘솔로그 결과:

18

이제 실제로 화면에 붙여보자. props를 age만 받지 말고, 원래대로 props로 돌려놓자.

```

1 interface StudentInfo {
2   name: string;
3   age: number;
4   isGirlfriend: boolean;
5   foo: number[];
6 }
7
8 function Student(props: StudentInfo) {
9   console.log(props);
10  return (
11    <>
12      <h1>Student</h1>
13      <div>이름: {props.name}</div>
14      <div>나이: {props.age}</div>
15      <div>{props.isGirlfriend ? "여자친구있음!" : "여친없음"}</div>
16      <div>데이터들: {props.foo}</div>
17    </>
18  );
19 }
20
21 export default Student;

```

화면에 붙일때는 변수를 중괄호로 감싸주면 된다.

line15는 삼항연산자를 사용해서, true일때는 여자친구 있음, false 일때는 여친없음을 화면에 출력시킨다. 삼항연산자는 화면에 붙일 때 자주 쓸 것이다.

앞에 props 붙은 게 싫다면, 따로따로 받아서 다음과 같이 붙이면 된다.

```

1 interface StudentInfo {
2   name: string;
3   age: number;
4   isGirlfriend: boolean;
5   foo: number[];
6 }
7
8 function Student({ name, age, isGirlfriend, foo }: StudentInfo) {
9   return (
10     <>
11       <h1>Student</h1>
12       <div>이름: {name}</div>
13       <div>나이: {age}</div>
14       <div>{isGirlfriend ? "여자친구있음!" : "여친없음"}</div>
15       <div>데이터들: {foo}</div>
16     </>
17   );
18 }
19
20 export default Student;

```

props 로 한번에 받지 않고, 프로퍼티 이름을 하나하나 받았다. 이 경우엔 화면에 붙일 때 props 를 앞에 안 붙여도 된다.

결과:

App

Student

이름: 조교행님
 나이: 18
 여자친구있음!
 데이터들: 1234

핵심정리

데이터는 최상위 컴포넌트인 App에서 관리해 자식으로 전달한다.

props 는 부모로부터 받은 객체이므로 인터페이스를 꼭 써줘야한다.

5. Map() 을 이용한 컴포넌트 반복

이제 많은 데이터를 받아서, 컴포넌트를 반복해보겠다.

먼저, 컴포넌트 파일에서 데이터를 작성해 쓰지 않고, 데이터가 있는 파일을 따로 만들도록 하겠다.



```
1  {
2    "data": {
3      "students": [
4        {
5          "id": 0,
6          "name": "조교행님",
7          "age": 18,
8          "isGirlfriend": true
9        },
10       {
11         "id": 1,
12         "name": "지수",
13         "age": 22,
14         "isGirlfriend": true
15       },
16       {
17         "id": 2,
18         "name": "펍수",
19         "age": 5,
20         "isGirlfriend": false
21       }
22     ]
23   }
24 }
```

data.json 이라는 파일을 만들었다. JSON 은 백엔드에서 프론트엔드로 데이터를 넘겨줄 때 가장 많이 사용되는 양식(form) 이다. 백엔드에서 데이터는 이런식으로 들어온다고 생각하면 된다. 그래서 프론트엔드 개발할때는, 백엔드 개발자가 코드를 만들때까지 기다리지 않고, 백엔드 코드와 연동하기전에 이런 샘플 데이터를 만들어놓은 다음에 화면을 만들어나간다.

구조를 자세히 보면, 자바스크립트 객체와 매우 비슷하게 생겼다. 다른 점으로는, 필드에도 쌍따옴표가 들어간다는 것 정도가 있는 것 같다.

참고로 JSON 은 매우 엄격한 문법을 사용하기 때문에 콤마 하나, 괄호 하나 틀리면 앱 전체가 안 돌아갈 수 있다.

이 데이터를 쓸 것이기 때문에 테스트로 인터페이스에서 써두었던 foo 부분은 지워버리겠다.

```
Student.tsx x
1 interface StudentInfo {
2   name: string;
3   age: number;
4   isGirlfriend: boolean;
5 }
6
7 function Student({ name, age, isGirlfriend }: StudentInfo) {
8   return (
9     <>
10      <h1>Student</h1>
11      <div>이름: {name}</div>
12      <div>나이: {age}</div>
13      <div>{isGirlfriend ? "여자친구있음!" : "여친없음"}</div>
14    </>
15  );
16 }
17
18 export default Student;
```

하나 이상한 건, 아까 JSON 데이터에는 id가 포함되어있었는데 이 인터페이스엔 id가 없다. 왜냐면 이 컴포넌트에선 id를 사용하지 않을 것이기 때문이다. 인터페이스엔 해당 컴포넌트에 사용할것만 쓰면 된다.

JSON을 import해서 쓰려면 TypeScript에서 한가지 설정을 해줘야된다. 보통은 백엔드로부터 바로 받기때문에 이런 설정을 할 필요 없이 잘 작동되는데, 우린 JSON 파일을 만들어두고 테스트를 할 것이기 때문이다. 파일 네비게이션을 보면 tsconfig.json 이라고 있다. 열어서, 다음을 추가해주자.

```

1  {
2    "include": [
3      "./src/**/*.ts"
4    ],
5    "compilerOptions": {
6      "strict": true,
7      "esModuleInterop": true,
8      "lib": [
9        "dom",
10       "es2015"
11     ],
12     "jsx": "react-jsx",
13     "resolveJsonModule": true
14   }
15 }

```

resolveJsonModule 을 추가하고 true 로 설정했다.

흔히 하는 실수중에 하나인데, 객체에서 이렇게 프로퍼티를 추가할 땐 앞줄에 콤마(,)를 넣어줘야 에러가 발생하지 않는다. 지금은 "react-jsx" 뒤에 콤마를 추가해줬다.

```

1  import Student from "./Student";
2  import "./styles.css";
3  import { data } from "./data.json";
4
5  console.log(data.students);
6
7  export default function App() {
8    return (
9      <div className="App">
10        <h1>App</h1>
11      </div>
12    );
13  }

```

앱 시작할 때 데이터가 잘 들어왔는지 콘솔로그로 찍어보겠다. 앞에 data 는 왜 들어갔는가? 나중에 백엔드 코드와 연동할때 굉장히 신경써야할 부분인데, 우리 데이터가

어떤 식으로 들어오는지 정확히 알고 있어야 한다.

이것은 개발할 때 매우 중요한 팁이다. 아까 JSON 파일을 다시 보면,

```
data.json x
1 {
2   "data": {
3     "students": [
4       {
5         "id": 0,
6         "name": "조교행님",
7         "age": 18,
8         "isGirlfriend": true
9       },
10      {
11        "id": 1,
12        "name": "지수",
13        "age": 22,
14        "isGirlfriend": true
15      },
16      {
17        "id": 2,
18        "name": "핑크수",
19        "age": 5,
20        "isGirlfriend": false
21      }
22    ]
23  }
24 }
```

맨 처음에 무엇으로 시작하는가?

data의 타입은 무엇인가?

그 안에 뭐가 있는가?

배열의 각 요소, 엘리먼트의 타입은 무엇인가?

그 객체는 무슨 프로퍼티로 구성되어 있는가?

각각의 자료형은?

data다.

중괄호로 쓰였으니깐 객체다.

students라는 배열이 있다.

객체다.

id, name, age, isGirlfriend 이다.

넘버, 스트링, 넘버, 불리언이다.

우리가 백엔드로부터 데이터를 받을때는, 이처럼 뭘 받는지 그 구조를 정확히 알고 있어야 한다. 이 데이터는 결국 프론트엔드에서 받기때문에, 백엔드 개발자가 정확히 뭘 넘겨줬으면 좋겠는지 문서화해서 백엔드개발자에게 '주문' 하는것이 제일 좋다.

어쨌든 콘솔로그로 찍어보면,

```
▼ (3) [Object, Object, Object]
  ▼ 0: Object
    id: 0
    name: "조교행님"
    age: 18
    isGirlfriend: true
  ▼ 1: Object
    id: 1
    name: "지수"
    age: 22
    isGirlfriend: true
  ▼ 2: Object
    id: 2
    name: "펑수"
    age: 5
    isGirlfriend: false
```

잘 들어온것을 알 수 있다.

이제 반복을 해볼 모든 준비가 끝났다. **map** 을 이용해 컴포넌트를 여러개 만들어볼 것이다. JavaScript 수업때 map을 설명했는데 까먹은 학생들을 위해 핵심만 잠깐 설명하자면, map 은 배열의 엘리먼트를 반복해서, 조작하여, 새로운 배열을 만드는 함수이다.

```

App.tsx  x
1  import Student from "./Student";
2  import "./styles.css";
3  import { data } from "./data.json";
4
5  console.log(data.students);
6
7  export default function App() {
8    return (
9      <div className="App">
10        <h1>App</h1>
11        {data.students.map((student) => (
12          <Student
13            key={student.id}
14            name={student.name}
15            age={student.age}
16            isGirlfriend={student.isGirlfriend}
17          />
18        ))}
19      </div>
20    );
21  }

```

data 객체의 students(복수) 배열의 각 요소를 student (단수) 라는 파라미터로 받고, 출력결과를 Student 컴포넌트로 리턴하는데, 각각의 Student에 props 로 name, age, isGirlfriend를 줄 것이다.

근데 하나를 더준다. line13의 key는 뭐냐면, 이것 안 써주면 React 에서 key 에러가 발생하기 때문에 써준것이다. 왜? **React에서 모든 컴포넌트는, 동일한 이름을 가졌다 하더라도 유일한 존재여야한다.** 그래서 key를 통해 구분해줘야한다. 보통 데이터의 id를 key로 쓴다. 왜? id는 각 데이터를 식별해주는, 유일한 요소이기 때문이다.

이 코드는 이렇게 짧게 줄일 수 있다.

```

App.tsx  x
1  import Student from "./Student";
2  import "./styles.css";
3  import { data } from "./data.json";
4
5  console.log(data.students);
6
7  export default function App() {
8    return (
9      <div className="App">
10        <h1>App</h1>
11        {data.students.map((student) => (
12          <Student key={student.id} {...student} />
13        ))}
14      </div>
15    );
16  }

```

props로 일일이 써줬던 것들을 이렇게 간단하게 줄일 수 있다. 이것을 property spread notation 이라고 하고, 2018년에 JavaScript 에서 정식 문법으로 채택되었다. [이걸 쓸지 말지는 개인 취향이다](#). 코드 짧게 쓰는게 좋은사람은 이렇게, 점 세개 붙여서 쓰면 되고, 정확하게 쓰고싶은사람은 첫번째 코드로 쓰면 된다.

그럼 어떤 일이 벌어졌을까?

결과:

App

Student

이름: 조교행님
나이: 18
여자친구있음!

Student

이름: 지수
나이: 22
여자친구있음!

Student

이름: 펑수
나이: 5
여친없음

`data.students` 배열의 갯수는 총 몇개였나? 3개였다. 그래서 `map` 함수에 의해 3개의 컴포넌트가 만들어졌고, 각자 다른 `props` 가 들어가서, 다른 결과를 보여준다.

물론, 맵함수의 파라미터를 따로 빼서 새로운 함수로 분리할수도 있다.

```

App.tsx x
1  import Student from "./Student";
2  import "./styles.css";
3  import { data } from "./data.json";
4
5  interface StudentInfo {
6      id: number;
7      name: string;
8      age: number;
9      isGirlfriend: boolean;
10 }
11
12 const renderStudent = (student: StudentInfo) => (
13     <Student key={student.id} {...student} />
14 );
15
16 export default function App() {
17     return (
18         <div className="App">
19             <h1>App</h1>
20             {data.students.map(renderStudent)}
21         </div>
22     );
23 }

```

이런식으로. 하지만 이 경우엔 student에 들어갈 interface를 따로 써줘야 작동한다. 함수를 따로 만들고 말고는 개발자가 개발 상황에 따라서 결정할 일이다. 지금같은 경우는 함수가 매우 짧기때문에 분리를 하는것보다 그냥 map 함수 안에서 작성하는게 더 낫다.

핵심정리

프론트엔드 개발자는 백엔드로부터 받을 데이터를 정확히 알고 있어야 한다.

컴포넌트를 반복할 땐 map() 을 사용하고, key 를 달아준다.

6. hook

다시 말하지만, React에서 주요한 기술은 딱 두가지다.

1. component
2. hook

즉, hook까지 배우고 나면 React의 핵심기술은 다 배운 셈이다.

프론트엔드 개발자는 화면을 만든다. 화면의 데이터 변경이 일어나면 화면은 더이상 최신정보를 담고 있지 않기 때문에 사용자는 새로운 화면이 필요하다. 새로운 화면을 가져오는 전통적인 방법은 무엇이었을까? 새로고침이었다. 그러나 **데이터가 변경되었을 때 새로고침 없이 즉각 화면을 다시 그릴 수 있다면** 멋지지 않을까?
hook을 사용하면 이것이 가능하다.

* 프론트엔드에서 화면을 그린다는 단어는 동사로 render, 명사로 rendering 이라고 한다.

* hook은 2019년 2월 React v16.8 부터 적용되었다. hook 이 없었다면 필자는 React 대신 라이벌 기술인 Vue.js 를 코스에 포함시켰을 것이다. hook 이전의 React 는 너무 어려웠기 때문이다. 그러나 hook 을 발표해버리면서 아예 새로운 개발방식을 제시했기 때문에 React의 진입장벽이 꽤 낮아졌다. 그리고 Vue.js 의 점유율이 React를 따라잡기가 힘들 정도로 낮기 때문에 React를 배우는것이 현 시점에선 가장 좋다.

hook은 대체 무엇일까? hook은 React 함수 컴포넌트에서만 쓰는 특별한 함수다. 함수이름 앞에 use 가 붙어있으면 hook이다.

hook은 크게 세가지로 구분된다.

1. useState()
2. useEffect()
3. 기타 hook. 직접 hook을 만들수도 있다.

useState

첫번째, useState부터 알아보자.

useState() 그대로 읽어보면, 사용한다. state를.

그럼 state는 무엇인가? 함수가 끝나도 사라지지 않는 변수가 state다. **state는 컴포넌트의 데이터 저장소**이며, state는 변한다. 한국어로 '상태'라고도 한다.

바로 코드로 가보자.

간단한 카운터를 만들어볼것.

일단 저번에 썼던 data.json 지워버리고,

import 빼버리고, Student 컴포넌트 삭제하고,

기본 코드로 돌아오자.

```
App.tsx
1  import "../styles.css";
2
3  export default function App() {
4    return (
5      <div className="App">
6        <h1>App</h1>
7      </div>
8    );
9  }
```

여기서, 다음 코드를 아무생각없이 그대로 써보자

```
App.tsx x
1  import { useState } from "react";
2  import "../styles.css";
3
4  export default function App() {
5    const [count, setCount] = useState(0);
6    return (
7      <div className="App">
8        <h1>App</h1>
9        {count}
10       <button onClick={() => setCount(count + 1)}>plus</button>
11     </div>
12   );
13 }
```

결과:

App

12 plus

누를때마다 숫자가 증가된다.

무슨 코드인가? 버튼을 누르면 데이터가 바뀐다.

그런데 새로고침이 발생했는가? 아니다. 새로고침 없이 자연스럽게 데이터가 증가한다.

다시, 순서대로 이해하면,
버튼 클릭 -> state 변경 -> App 컴포넌트만 다시 렌더링
이런 식으로 실행되는 것이다.

```
App.tsx x
1 import { useState } from "react";
2 import "./styles.css";
3
4 export default function App() {
5   const [count, setCount] = useState(0);
6   return (
7     <div className="App">
8       <h1>App</h1>
9       {count}
10      <button onClick={() => setCount(count + 1)}>plus</button>
11    </div>
12  );
13 }
```

button 옆에 onClick은 이벤트 핸들러다. 1장 HTML 에서 설명했듯이, 이벤트 핸들러는 React의 개념이 아니라, 원래 HTML에 있는 개념이지만 React에선 조금 다르게 쓴다.

- ```
const [count, setCount] = useState(0);
```

state      count 변경하는 함수      count의 초깃값

```
<button onClick={ () => setCount(count+1) }>plus</button>
```

누르면,      실행된다      변경 내용

- 48 -



그 옆에 있는 `setCount`는 `count`를 변경하는 '함수'이다. 관습적으로, `set` 이라는 글자 다음에, 담당하는 `state` 이름을 적어준다.

즉, `useState` 라는 hook은 두가지 요소(엘리먼트)로 구성된 하나의 배열을 리턴하는데, `state`와 `state`를 변경하는 함수이다.

그리고 `useState`의 파라미터로 사용된 `0`은 우리의 `state`인 `count`의 초기값이다.

`onClick` 이벤트핸들러는 파라미터를 받지 않고, 실행 시 리턴값으로 `setCount` 함수를 실행한다. `setCount` 함수의 파라미터로 `count` 변경 내용이 들어간다.

그리고 `react` 는 화면 전체가 아니라, 컴포넌트를 다시 렌더링한다.

이론적으로 어떻게 작동하는지를 설명했다. 전체 실행과정을 다시 생각해보면,

버튼 클릭 -> `state` 변경 -> App 컴포넌트만 다시 렌더링

즉, 새로고침 없이 부드럽게 데이터가 변경되는것을 확인할 수 있다. 이렇게 작동하는게 마치 설치된 프로그램(네이티브 앱)처럼 동작한다고 해서 웹앱이라고 부르는 것이다.

\* 웹앱 이전의 웹개발은 개발로 쳐주지 않는 분위기가 개발자들 사이에서 있었다. 하지만 웹앱 이후로, 돌아가는 환경만 브라우저일 뿐인 진짜 프로그램(앱)을 만들 수 있게 된 것이다.

`state`, `setState` 를 props 로 내려보내기

App 컴포넌트에서 `useState`를 통해 만든 `count` 와 `setCount` 를 자식컴포넌트에게 props 로 내려보내서, `state`를 연동하고 수정해보자.

자식 컴포넌트 하나를 만들자.

```
Child.tsx x
1 interface propTypes {
2 count: number;
3 setCount: (num: number) => void;
4 }
5
6 export default function Child(props: propTypes) {
7 return (
8 <>
9 <h1>Child</h1>
10 {props.count}
11 <button onClick={() => props.setCount(props.count + 1)}>
12 child plus
13 </button>
14 </>
15);
16 }
```

props의 인터페이스를 만들어줘야한다. setCount에 단순히 function 이라고 쓰면 TypeScript에서 에러를 일으킨다. 함수의 정확한 타입을 써줘야한다. 파라미터 타입과 리턴 타입을 정확히 써주었다. num은 우리가 지은 이름이라서, n으로 바꾸든 x로 바꾸든 잘 동작한다. 그리고 setCount는 리턴값이 없기때문에 타입은 void이다.

\* void 는 빈 공간이라는 뜻이다. 파라미터나 리턴이 없으면 void 타입이다.

```
App.tsx x
1 import { useState } from "react";
2 import Child from "../Child";
3 import "../styles.css";
4
5 export default function App() {
6 const [count, setCount] = useState(0);
7 return (
8 <div className="App">
9 <h1>App</h1>
10 {count}
11 <button onClick={() => setCount(count + 1)}>plus</button>
12 <Child count={count} setCount={setCount} />
13 </div>
14);
15 }
```

Child 컴포넌트를 등록하고 props 를 내려보냈다.

결과:

App

13 plus

Child

13 child plus

child plus 를 클릭하면 무엇이 실행되는가? App에서 props로 내려보낸 setCount가 실행된다. setCount는 자식 컴포넌트에 있는 것처럼 보이지만 실제로는 App 에 있는 setCount를 자식컴포넌트인 Child 에서 실행한 것일 뿐이다.

즉, 이런 방식으로 모든 state를 App에서 관리하고, 수정할 수 있다.

### useEffect

두번째 hook 인 useEffect에 대해 알아보자. useEffect는 컴포넌트에서 state가 변경되고

컴포넌트가 다시 렌더링된 이후에 매번 실행되는 hook이다.

1. useState로부터 만들어진 set~~~ 함수가 실행되어 담당하는 state를 변경한다.
2. 렌더링
3. useEffect 실행

코드로 확인해보자.

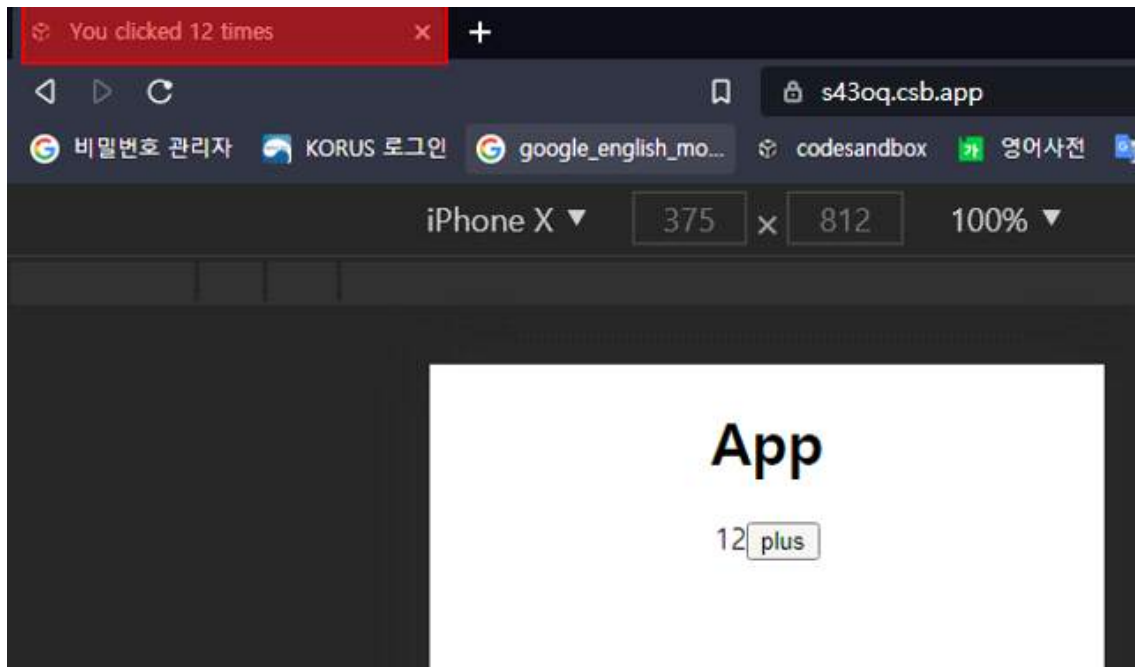
```
App.tsx
1 import { useState, useEffect } from "react";
2 import "./styles.css";
3
4 export default function App() {
5 const [count, setCount] = useState(0);
6 useEffect(() => {
7 // 브라우저 API를 이용하여 문서 타이틀을 업데이트합니다.
8 document.title = `You clicked ${count} times`;
9 });
10 return (
11 <div className="App">
12 <h1>App</h1>
13 {count}
14 <button onClick={() => setCount(count + 1)}>plus</button>
15 </div>
16);
17 }
```

line1: useEffect 를 import 해주었다.

line6 - 9: DOM 에 접근해 title 변경

\* React 에서 document 를 사용해 DOM 에 직접 접근하는것은 기피해야된다고 TypeScript 강좌때 이미 말했다. 그러나 단순한 테스트 예제이기 때문에 예외로 하겠다.

결과:



지금 테스트창에선 그 결과를 알기 어려우니, 테스트창에 있는 url 을 가져와서 크롬에 새창 띄우고 결과를 확인해보았다.

클릭할때마다 현재 페이지의 제목이 바뀌는 것을 알 수 있다.

useEffect 는 언제 써야할까? 렌더링이 이미 끝난 이후에, 즉 변경된 데이터가 화면에 부착된 다음이라야만 할 수 있는 작업들에 사용된다. 이런 작업들이 있다.

1. db 또는 앱 외부로부터 데이터 가져오기
2. DOM 요소 변경

마지막으로, hook의 규칙을 알아보자.

규칙을 따르지 않으면 linter(eslint) 에 의해 error 발생한다. 린터는 쉽게 말하면 코딩에서 쓰이는 맞춤법검사기다.

1. 최상위(at the Top Level)에서만 hook을 호출해야한다.  
반복문, 조건문, 중첩함수에서 hook을 호출하지 말라
2. 오직 React 컴포넌트 함수에서만 Hook 을 호출해야한다.  
일반적인 JavaScript 함수에서 호출하지 말라. 왜? React의 문법이니까.  
hook 안에서 hook을 호출하는것은 가능하다.
3. 이름지을땐 use를 앞에 쓰라.  
use가 앞에 쓰여야만 linter 가 위 두가지 규칙을 어기지 않았는지 검사할 수 있다

이번엔 핵심정리가 없다. 모든 부분이 다 중요하다. 이해가 안되면 다시 읽어봐서라도 확실히 익혀두자.

## 7. router

보통 우리가 만들 웹앱은 딱 하나의 모습만 보여주지 않는다. 로그인이 있고, 로그인 넘어가면 메인화면, 사용자버튼 클릭하면 현재 사용자 정보, 게시판 클릭하면 게시판화면 등등, 하나의 웹앱엔 수많은 화면이 존재한다.

지금까지 배운 기술만으로 이를 구현하려면 어떻게 해야할까? 각각의 mode를 따로 만들어서, login mode 일때는 다른 모든 컴포넌트를 꺼버리고 login mode 만 보이게 하면 될 것이다. 그러나 사실 이것은 매우 번거로운 일이다. 그 대신, 우리는 router 라는 것을 사용해서 이 작업을 좀 더 편리하게 해볼 것이다.

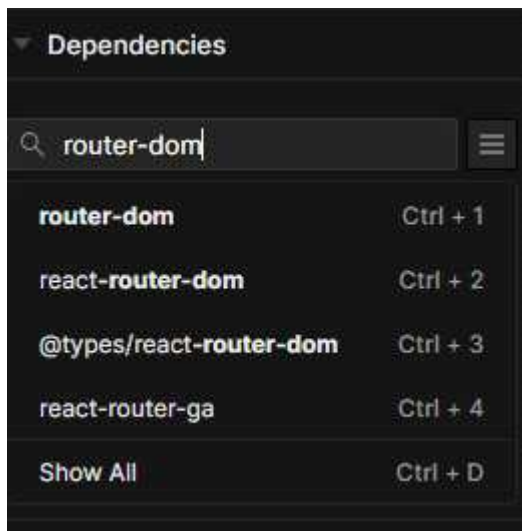
router는 url에 따라서 보이는 화면이 달라지게 만든다.

`https://react.com/login`  
로그인 화면

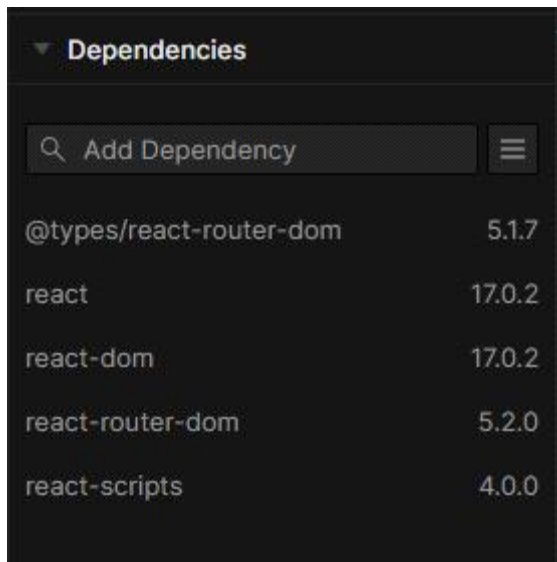
`https://react.com/main`  
메인화면

이런식이다. 뒤에 따라오는 url로 웹페이지를 구분한다.

일단 React는 Router를 제공하지 않는다. 이 기능을 쓰기 위해선 `react-router-dom` 이라는 것을 dependency 로 추가해줘야한다.



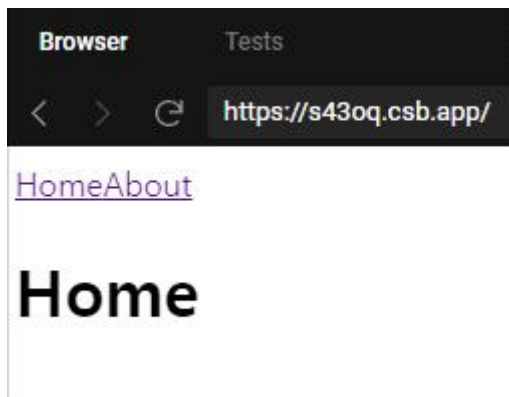
`react-router-dom` 을 추가하고, 추가로 우린 TypeScript를 사용하니깐, `@types/react-router-dom` 을 추가해주도록 하겠다.



react-router-dom v5.2.0

@types/react-router-dom v5.1.7

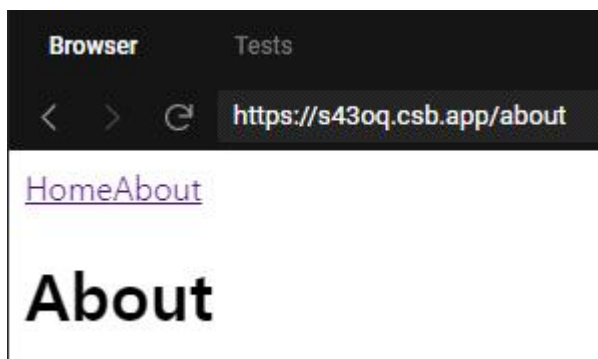
우리가 하고자 하는 것부터 확실하게 알고 가자.



이런 화면을 만들어놓고,

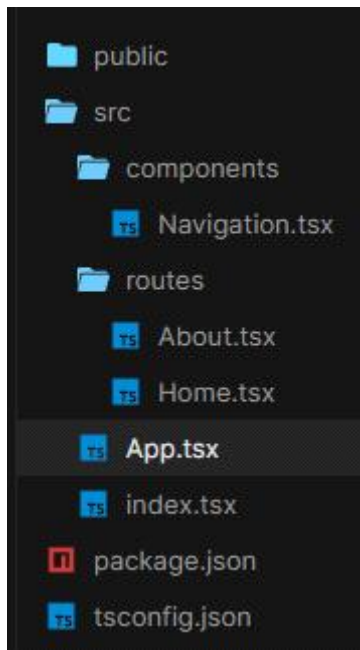
Home 링크를 클릭하면 Home 컴포넌트가 보이게 한다.

만약 About 링크를 클릭하면?



주소창을 잘 보자. /about 이 추가되었다. 이 화면에선 About 컴포넌트의 내용만 보인다.

이제 만들어보자. 가장 먼저, 디렉터리 구조를 바꿔줄 것이다. 기존에 있던 것을 지우고 아예 새로운 프로젝트를 시작한다고 생각해라.



components 디렉터리와 routes 디렉터리를 만들었다.

routes 는 각 화면의 최상위 컴포넌트라고 생각하면 된다.

components 는 쓸만한 컴포넌트를 모아둔 디렉터리다. Home과 About을 전환할때 쓰일

Navigation 이라는 컴포넌트를 넣어놨다.

App.tsx에선 지금부터 화면이 아니라, router와 state 관리만 담당한다.

styles.css 는 삭제했다.

가장 간단한 Home.tsx 와 About.tsx 다. 보드시피 그냥 컴포넌트이다.

```
Home.tsx x
1 export default function App() {
2 return (
3 <>
4 <h1>Home</h1>
5 </>
6);
7 }
```

```

About.tsx
1 export default function About() {
2 return (
3 <>
4 <h1>About</h1>
5 </>
6);
7 }

```

다음은 핵심인 App.tsx이다.

```

App.tsx
1 import { BrowserRouter, Route } from "react-router-dom";
2 import Home from "../routes/Home";
3 import About from "../routes/About";
4 import Navigation from "../components/Navigation";
5
6 export default function App() {
7 return (
8 <BrowserRouter>
9 <Navigation />
10 <Route path="/" exact={true} component={Home} />
11 <Route path="/about" component={About} />
12 </BrowserRouter>
13);
14 }

```

line1: BrowserRouter, Route 두가지를 react-router-dom 으로부터 import 한다.

line2 - 4: routes 디렉터리에 있는 Home, About 컴포넌트와, components 디렉터리에 있는 Navigation 컴포넌트를 import 한다.

line8 - 12: BrowserRouter 로 감싸준다.

line9: Navigation 컴포넌트이다. 이 컴포넌트엔 Home 과 About으로 갈 수 있는 링크가 있다. 이름이 꼭 Navigation 일 필요는 없다. LeftNavigation, TopNavigation 등, 네비게이션 컴포넌트는 상황에 따른 이름을 지어주면 된다. 단, 네비게이션은 반드시 BrowserRouter 안에 위치해야하는데, 왜냐면 BrowserRouter 안에 각각의 Route 의 path에 접근할 것이기 때문이다.

line10 - 11: Route 안에는 각 컴포넌트로 향하는 정보가 담겨있다.

두 가지 필수적인 프로퍼티가 있다. path 와 component이다. exact는 선택사항이다.

path는 url 경로를 말한다.

exact를 true로 지정하면 하위 경로를 포함해서 보여주지 않는다.



무슨 말이나면, 만약 exact 부분을 빼버릴 경우, / 주소로 향할 시엔 about까지 같이 보여주게 된다. about 앞에 /가 포함되어 있기 때문이다.  
(테스트 시에 차이점을 보여줄 것이다.)  
component는 해당 path에서 보여주고싶은 컴포넌트를 말한다.

다음은 Navigation 컴포넌트이다.

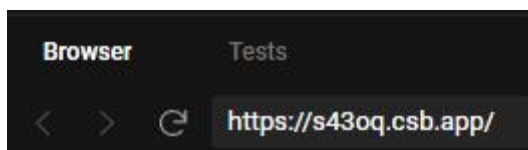
```
Navigation.tsx x
1 import { Link } from "react-router-dom";
2 export default function Navigation() {
3 return (
4 <>
5 <Link to="/">Home</Link>
6 <Link to="/about">About</Link>
7 </>
8);
9 }
```

line1: react-router-dom에서 제공하는 Link를 import 한다.

line5 - 6: Link를 두 개 만들 것이다. 이것은 <a> 태그와 똑같이 화면에 보여질 것이다.  
to 뒤에, 해당 링크를 클릭했을 때 향할 주소를 쓴다.  
이 주소는 App.tsx의 각 Route 의 path와 일치해야 한다.

결과:

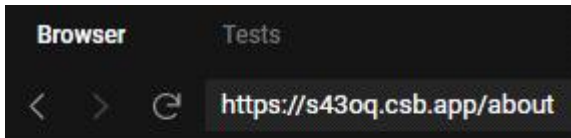
첫 화면. 주소는 무엇인가? 원래 주소 그대로다.



[HomeAbout](#)

# Home

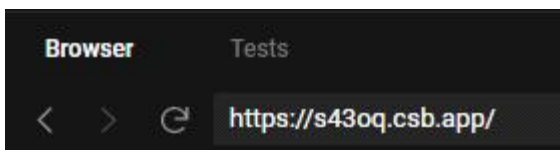
About을 클릭했을 경우 /about 에 연결된 About 컴포넌트를 보여준다.



[HomeAbout](#)

## About

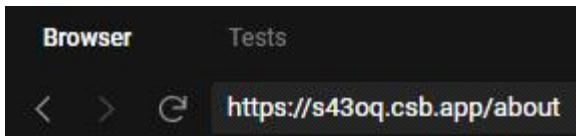
다시 home을 클릭했을 경우. 원래 주소로 돌아왔다.



[HomeAbout](#)

## Home

만약, App.tsx의 Home Route 에서 exact={true} 부분을 빼버리면 어떤 일이 발생할까?



[HomeAbout](#)

## Home

## About

이처럼, Home 아래에 About 이 렌더링된다. 왜? Home 컴포넌트의 path는 "/" 인데, About 컴포넌트의 path는 "/about"으로 "/" 를 포함하고 있기 때문이다. 개발자는 자기가 개발하고자 하는 웹앱의 형태에 따라서, Route 의 exact 를 true 로 지정할지 결정해야 한다.

Router는 결국 무엇인가? url에 따라서 보이는 화면을 달라지게 만드는 것이다.

## 8. route props

마지막 장이다. 각각의 route 에 props 을 보내주는 연습을 해볼 것이다. 이것은 필수적인데, App 컴포넌트에서 모든 state를 관리하기 때문이다.

클릭하면 단순히 페이지가 바뀌는것뿐만 아니라, 데이터를 전달해줄것이다.

```
Navigation.tsx x
1 import { Link } from "react-router-dom";
2 export default function Navigation() {
3 return (
4 <>
5 <Link to="/">Home</Link>
6 <Link to="/about">About</Link>
7 </>
8);
9 }
```

핵심은 Navigation 컴포넌트 내부의 Link 의 프로퍼티인 to 이다.

to는 path 뿐만 아니라, 객체를 보낼 수 있다.

```
<Link to="/">Home</Link>
<Link
 to={{
 pathname: "/about",
 state: { fromNavigation: true }
 }}
>
 About
</Link>
```

이런 식으로. 이 경우, 주소는 pathname 이라는 프로퍼티로 확실하게 써줘야한다. state가 바로 우리가 보낼 객체다. 이것을 About 컴포넌트로 보내버리자.

```

1 export default function About({ location: { state } }: any) {
2 console.log(state);
3 return (
4 <>
5 <h1>About</h1>
6 </>
7);
8 }

```

line1: 객체의 일부분만 가져올 땐, 이런 식으로 쓴다.

`{ location: { state } }` props 안에 있는 state만 가져올 것이다.

\* any라는 타입은 처음 볼 텐데, 아무 타입이나 허용하겠다는 뜻이다. 원래 객체를 파라미터로 받을 땐 interface를 써줘야 한다고 했다. 우린 간단한 테스트를 해볼 것이기 때문에 any를 써보았는데, **any는 테스트할때 말고는 쓰지 말아야 한다. any를 남용하면 TypeScript를 쓸 이유가 없기 때문이다.**

line2: 콘솔로그. 첫 시작 화면에서 about 링크를 클릭하면 다음 콘솔로그가 출력된다.

```

▶ {fromNavigation: true}

```

정상적으로 데이터가 들어온 것을 볼 수 있다.

react-router-dom 에는 이 밖에도 많은 기능이 있지만, 개발하면서 차차 알아볼 기회가 있을 것이다.

## 9. 마치며

벌써 마친단 말인가? **그렇다.**

React의 모든 것을 다루었는가? **아니다.**

React뿐만 아니라, React 생태계와 밀접하게 연관된 다른 기술들, Redux, Apollo, Webpack은 다루지 않았다. VSCode로 우리만의 개발환경도 구축해본적이 없다.

우린 React 의 핵심만 다루어봤을 뿐이다.

심지어 이 가이드엔 실전예제 하나 포함되어있지 않다. 그저 기술들만 나열했을 뿐이다.

그러나, 실전개발을 해보기 전에 다음 두 가지를 마쳐야 한다고 생각한다.

첫째. 스타일링. 프론트엔드 개발자가 기초적인 디자인을 할 줄 모르면 안된다.

둘째. 백엔드. 데이터베이스와 프론트엔드를 연결할 줄 알아야 한다.

이 코스에선 이후 두 가지 길을 제시한다. 둘 중 어떤 걸 먼저 수강하든지, 상관없다.

1. 스타일링: Tailwind CSS
2. 데이터베이스: SQLite

추가적으로 공부를 하고 싶은 학생은 React 에서 제공하는 공식 자습서를 공부해보는것을 권장한다. 한국어로 번역되어있다.

<https://ko.reactjs.org/tutorial/tutorial.html>

물론 선택사항이다. React에 대한 '핵심'은 다 배웠다.