



고급 소프트웨어 실습(CSE4152)


7주차 React – To do List

목차

- 웹 프로그래밍 소개
- React 소개
- NPM 명령어
- React 프로젝트 생성법
- JSX 문법
- JavaScript 문법
- 실습 설명
- 과제 설명

웹 프로그래밍

- 웹 프로그래밍은 웹 사이트/ 페이지를 접속했을 때 보이는 화면을 구성하는 것들을 실제로 만들어 내는 작업을 뜻한다.
- 최근 유무선 통신망에서 다양한 종류의 웹 기반 인터넷 서비스가 널리 보급됨에 따라 웹 프로그래밍 기술이 유용하게 사용된다.
- 웹 프로그래밍은 프론트엔드 프로그래밍, 백엔드 프로그래밍으로 나누어진다.

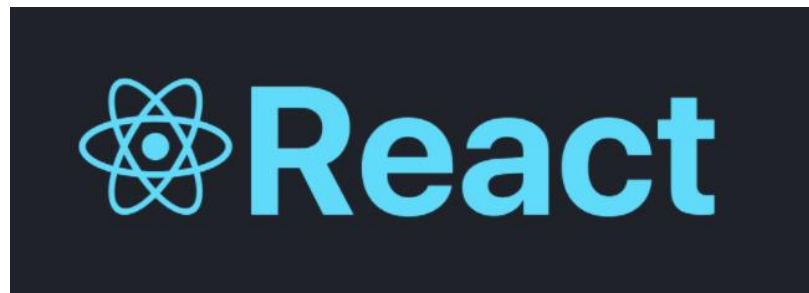
 웹 프로그래밍을 통해서 프로그램의 작동을 확인할 수 있으며, 배포를 통해 실제로 다른 사용자들이 구현한 서비스를 사용할 수 있다.

Front-end & Back-end Programming

- 프론트엔드 프로그래밍(Front-end Programming): 브라우저 단에서의 동작을 구현하는 것, 눈에 보이는 것들의 담당, 클라이언트 측 프로그래밍
 - * 사용 언어: HTML (페이지의 구조, 뼈대), CSS (HTML 요소들에 디자인 요소 적용), Javascript (페이지에서의 동적인 작동 구현)
- 백엔드 프로그래밍(Back-end Programming): 웹 서버 단에서의 동작을 구현하는 것, 데이터 저장 및 삭제 등의 웹 페이지의 보이지 않는 실제 기능들을 구현

React(React.js)란?

- 페이스북에서 제공하는 자바스크립트 라이브러리
- UI(User Interface) 개발에 특화되어 웹/앱 어플리케이션 개발에 최근 활발히 사용됨
- 어플리케이션의 동적 기능 구현을 효과적으로 할 수 있음
- 프론트엔드 + 백엔드 프로그래밍을 한번에 구현 가능



JavaScript & JavaScript Library

- JavaScript: HTML, CSS로 만들어진 정적인 웹 페이지에 동적인 기능들을 구현
 - 정적인 웹 페이지: 화면의 구성요소가 변경되지 않음
 - 동적인 웹 페이지: 경고, 탭 인터페이스, 클릭 이벤트 등의 기능이 작동
- JavaScript Library: 자바스크립트에서 자주 사용하는 기능들을 모아 정리하여 웹 개발 시 필요한 기능을 가져와 편하게 재사용하도록 만들어 놓은 코드의 집합

 React는 JavaScript Library의 일종

React의 특징

- 단방향 데이터 흐름
 - 데이터의 흐름이 부모 -> 자식, 한 방향으로만 흐르는 단방향 데이터 흐름 (one-way data flow)을 가짐
- 컴포넌트(Component) 기반
 - 독립적인 단위의 소프트웨어 모듈인 컴포넌트(component)로 UI 구성
 - 리액트의 모든 구성요소는 컴포넌트(버튼, 폼, 레이아웃, 화면)
 - 원하는 컴포넌트만 수정 가능
 - 재사용성, 관리, 유지보수가 용이

React의 특징

- JSX 사용
 - 자바스크립트를 확장한 문법
 - Html과 유사
- 가상 돔(Virtual DOM)
 - DOM(Document Object Model)은 웹 어플리케이션 파일(html, xml, css ...)을 트리 구조로 인식, 데이터를 객체로서 관리
 - React는 이벤트 발생 시마다 가상 돔을 만들고, 실제 돔과 비교하여 최소한의 변경사항만 실제 돔에 반영

React의 특징

- Props, State 개념
 - Props: 부모 -> 자식 컴포넌트로 전달해주는 데이터, 전달받은 props는 자식 컴포넌트에서 변경 불가능, 전달해준 최상위 부모 컴포넌트만 변경 가능
 - State: 컴포넌트 내부에서 선언하여 내부에서 값 변경 가능, 사용자와의 상호작용을 반영하기 위한 동적 데이터 구현에 사용, 클래스형 컴포넌트에서만 사용 가능, 각 state는 독립적임

npm 명령어

- Node.js: 자바스크립트 런타임 환경, React.js 사용에 편리한 도구를 제공하는 오픈소스
- NPM은 Node.js 기반 모듈들을 모아 둔 패키지 관리자로, Node.js 설치 시 같이 설치됨
- NPM 명령어로 생성 및 업데이트 등의 프로젝트 관리를 할 수 있다.
- npm install: NPM 패키지 설치
- npm update: 설치한 패키지 업데이트
- npm help: 명령어 목록 확인
- npm start: package.json의 scripts에서 start 명령어 실행
(앱을 실행시킴, 실제 앱 프로젝트 작동 확인 시 사용)
- npm stop: npm start 했던 것을 멈춤
- npm restart: stop 후 재시작

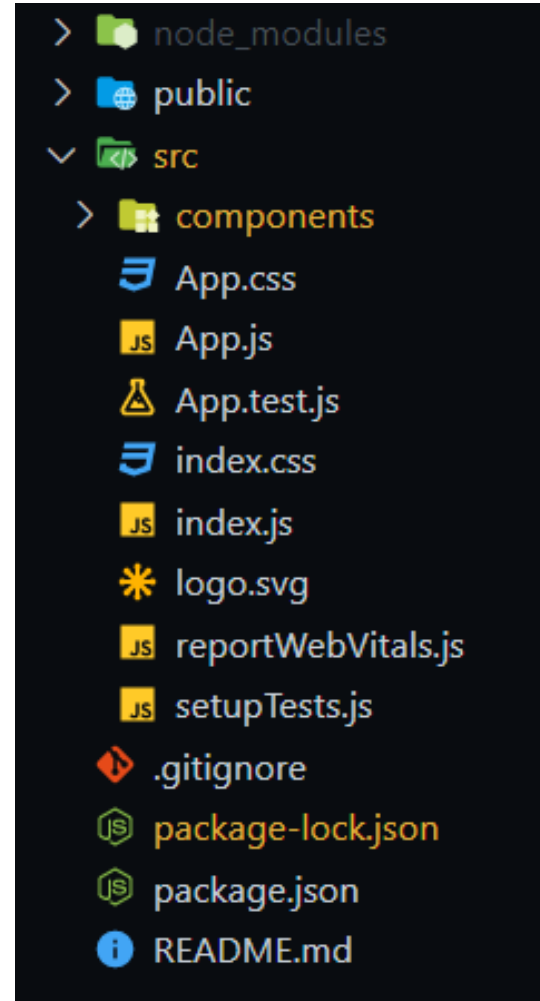
React 프로젝트 생성

- docker 실행 후 터미널/커맨드 창에서 다음과 같은 명령어 입력

```
npx create-react-app my-app // react 앱 생성
cd my-app                  // 생성된 앱 프로젝트 디렉토리로 이동
npm start                  // 생성된 앱 실행(http://localhost:3000/을 통해 확인 가능)
```

React 프로젝트 생성

- React 프로젝트 생성 시
기본으로 제공되는 파일 구조



React 프로젝트 구성

- node_modules: npm install 명령어 실행으로 설치된 모듈들을 담는 폴더
- public: static 파일(이미지 등)들을 담는 폴더
- src: React 프로젝트의 실질적 개발 작업이 이루어지는 폴더
- src/App.js: 웹 페이지에 실제로 보여지는 코드, 작성한 컴포넌트를 가져와서 구현
- src/components: 웹 페이지를 구성하는 컴포넌트들을 담는 폴더
- src/index.js: ReactDOM.render() 를 통해 작업한 App.js를 렌더링하는 최상위 메인 파일
- package.json: 프로젝트의 메타 정보를 담는 파일, start, build, test 등의 스크립트를 수정 가능

JSX란?

- 자바스크립트를 확장한 문법
- React에서 개발을 쉽게 하기 위해 HTML과 비슷한 문법으로 코드를 작성할 수 있도록 해 줌
- 실제 빌드 시 자바스크립트로 자동 변환

JSX 파일의 기본 구조

```
import React from "react";
import OtherComponent from "../OtherComponent";
```

필요한 기능 및 컴포넌트를

라이브러리/다른 파일에서 불러옴

(JSX를 사용하려면, React를 꼭 import 해야 함)

```
function ExampleComponent() {
  const curProps = 1;

  const exampleFunc = () => {
    // do the particular job
    curProps += 1;
  };

  return (
    <div>
      <OtherComponent props={curProps} func={exampleFunc} />
      <h1>This is example.</h1>
    </div>
  );
}
```

function 키워드로 함수형 컴포넌트를 생성,

return() 안에는 항상 최종적으로 하나의 컴포넌트

(여기선 <div></div> 태그)가 다른 컴포넌트들을 감싸는 형태가 되어야 함

컴포넌트 내 정의된 변수와 함수

return (

다른 컴포넌트에 props로 현재 컴포넌트 내 변수, 함수를 넘겨줄 수 있음

```
<div>
  <OtherComponent props={curProps} func={exampleFunc} />
  <h1>This is example.</h1>
</div>
```

실제
jsx 코드

);

```
export default ExampleComponent;
```

다른 파일에서 사용할 수 있도록 작성한 component를 export해 줌

OtherComponent.jsx 파일

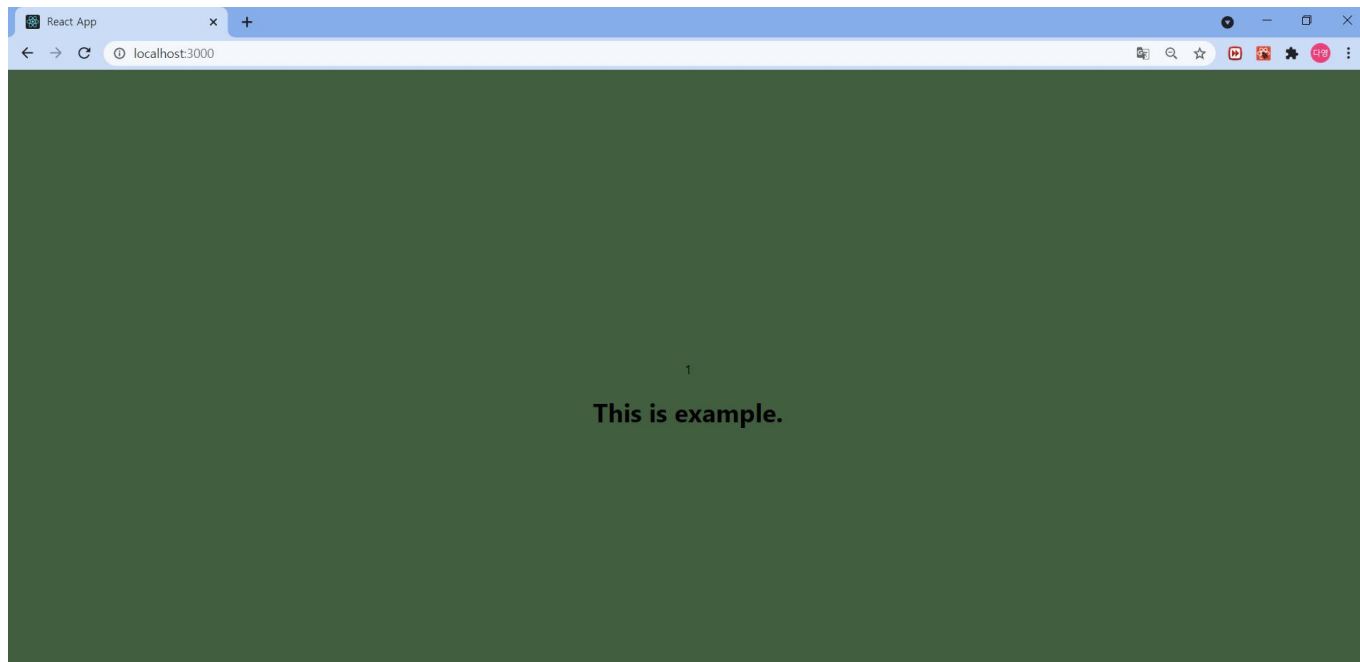
```
import React from "react";

function OtherComponent({props, func}) {

  return (
    <div>
      {props}
    </div>
  );
}

export default OtherComponent;
```


예제 파일 실행 결과



JSX 문법

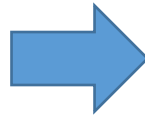
- 두 개 이상의 element(HTML 태그 / 컴포넌트)는 무조건 하나의 element로 감싸져야 함

```
import React, { Component } from 'react';

class App extends Component {
  render() {
    return (
      <div>
        Hello
      </div>
      <div>
        Bye
      </div>
    );
  }
}

export default App;
```

<컴파일 오류 발생>



```
import React, { Component } from 'react';

class App extends Component {
  render() {
    return (
      <div>
        <div>
          Hello
        </div>
        <div>
          Bye
        </div>
      </div>
    );
  }
}

export default App;
```

<div 태그로 두 element를 감싸 줌>

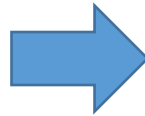
JSX 문법

- 모든 태그는 반드시 `</>`를 사용하여 닫혀 있어야 한다.

```
import React, { Component } from 'react';

class App extends Component {
  render() {
    return (
      <div>
        <input type="text">
      </div>
    );
  }
}

export default App;
```



```
import React, { Component } from 'react';

class App extends Component {
  render() {
    return (
      <div>
        <input type="text"/>
      </div>
    );
  }
}

export default App;
```

<컴파일 오류 발생>

<input 태그를 닫아 줌>

JSX 문법

- 자바스크립트를 사용할 때 {}안에 변수명, 자바스크립트 계산식, 값 등을 넣어 사용 (ex: {console.log(this.props)})

```
import React, { Component } from 'react';

class App extends Component {
  render() {
    const name = 'react';
    return (
      <div>
        hello {name}!
      </div>
    );
  }
}

export default App;
```

변수 선언:
var, let, const

<자바스크립트 문법으로 선언한 변수를 jsx 코드 내에서 사용>

JSX 문법

- 조건을 사용하여 render를 할 때는 if 문을 사용할 수 없음
- 삼항 연산자 혹은 AND 연산자(&&) 사용

```
import React, { Component } from 'react';

class App extends Component {
  render() {
    return (
      <div>
        {
          1 + 1 === 2
            ? (<div>맞아요!</div>)
            : (<div>틀려요!</div>)
        }
      </div>
    );
  }
}

export default App;
```

<삼항 연산자 사용하여 조건부 렌더링 구현>

JavaScript - 변수

- let을 통해서 변수 선언
 - 변수는 값을 선언 후 바꿀 수 있음
 - let은 선언 후 똑같은 이름으로 선언 불가능
 - var 키워드는 최근 자바스크립트에서 잘 사용하지 않음

```
let val = 1;
console.log(val); // 1
val = 2;
console.log(val); // 2

let val2 = 3;
let val2 = 4; // syntax error
```

- const를 통해서 상수 선언
 - 상수는 선언 후 값을 바꿀 수 없음

```
const val3 = 1;
val3 = 2; // error: val3 is read-only
```

JavaScript – 데이터 형식

- 숫자(Number) : 숫자 값을 변수에 대입
- 문자열(String) : 텍스트를 ‘ ’ 혹은 “ ”로 감싸서 변수에 대입
- 참/거짓(Boolean) : true, false 두 가지 종류의 값만 사용
- null : 변수에 값이 없음을 나타내기 위해 사용
- undefined : 선언은 했지만, 아직 값이 설정되지 않은 변수
- 배열: []로 감싸 여러 가지 값을 저장하는 리스트
 - 새로운 항목 추가할 때에는 push 함수 사용
 - 배열의 크기는 length 함수 사용해서 확인 가능

JavaScript – 객체

- `{}` 안에 원하는 값을 넣어 객체 선언
- **키: 원하는 값** 의 형태를 사용해서 값을 넣음
 - 키에 해당하는 부분에 공백이 있다면 `''`로 감싸 문자열의 형태로 만들어 줌
- 함수도 값으로 넣을 수 있음
 - 객체의 함수 속 `this`는 자신이 속한 객체를 가리킴

```
const student = {  
  name: 'Alice',  
  id: '20199999',  
  greeting: function () {  
    console.log('안녕하세요, ${this.id} ${this.name} 입니다.');  }  
};  
  
student.greeting(); // 안녕하세요 20199999 Alice 입니다.
```


JavaScript – 조건문

- 조건문은 아래와 같은 형태로 사용

```
if (조건1) {  
    조건 1 만족 시 동작 코드;  
} else if (조건2) {  
    조건 2 만족 시 동작 코드;  
} else {  
    나머지 경우 동작 코드;  
}
```

- switch/case 문도 C언어와 같은 형태로 사용 가능

JavaScript – 조건문

- 조건의 비교에서 '=='와 '!='를 사용하면 피연산자의 형 변환 후 비교
- 대신 '===' , '!== '를 사용하면 형 변환을 하지 않고 비교

```
1 254 == '254'           // return true
2 true == 1              // return true
3 undefined == null      // return true
4 'abc' == new String('abc') // return true
5 null == false          // return false
6 'true' == true         // return false
7 true == 2              // return false
```

<형 변환 O>

```
1 254 === '254'          // return false
2 true === 1             // return false
3 undefined === null     // return false
4 'abc' === new String('abc') // return false
```

<형 변환 X>

JavaScript – 반복문

- For 문은 아래와 같이 c언어와 같은 형태로 사용

```
for (초기 구문; 조건 구문; 변화 구문) {  
    반복 코드;  
}
```

- while 문도 c언어와 같은 형태로 사용 가능
- for...in 반복문을 사용하여 객체가 가진 값을 반복할 수 있음
- break, continue를 사용하여 반복문 제어 가능

JavaScript – 나머지 매개변수 ...

- 나머지 매개변수 '...'
 - 함수 매개 변수 목록의 끝에서 인수 목록의 나머지를 배열로 모아 저장

```
function showName(fruit1, fruit2, ...rest) {  
    alert( fruit1 + ' ' + fruit2 ); // Apple Banana  
  
    // 나머지 인수들은 배열 rest에 할당  
    // rest = ["Grapes", "Orange"]  
    alert( titles[0] ); // Grapes  
    alert( titles[1] ); // Orange  
    alert( titles.length ); // 2  
}  
  
showName( "Apple", "Banana", "Grapes", "Orange");
```

JavaScript – 화살표 함수

- function 키워드를 사용하지 않고 화살표('=>')를 사용하여 간단하게 함수를 표현
- 함수 표현식을 통해 나타내며, 항상 익명 함수로 사용
- Call back 함수로 사용 가능
- 화살표 함수의 기본 문법:

```
// 일반 함수
var foo = function () { console.log("foo") }; // foo

// 매개변수가 없는 화살표 함수
var foo = () => console.log("foo"); // foo

// 매개변수가 2개인 화살표 함수
var foo = (a, b) => {
  var c = 3;
  return a + b + c;
} // a+b+c를 return
```

JavaScript – map 함수

- 배열 내 각각의 요소에 대해 주어진 함수를 수행하고 그 결과를 모아 새로운 배열 생성
- 형태: `array.map(callback (currentValue, index, array), thisArg)`
 - `currentValue` : 배열 내 현재의 값
 - `Index` : 배열 내 현재 값의 인덱스
 - `array`: 현재 배열
 - `thisArg`: callback 함수에서 `this`로 사용될 값

```
const arr = [1, 2, 3];  
const marr = arr.map((curVal) => curVal + 1);  
const marr2 = arr.map(function addOne(curVal){  
  return curVal + 1;  
})
```

```
console.log(marr); // [2, 3, 4]  
console.log(marr2); // [2, 3, 4]
```

React – useState

- state: 사용자와의 상호작용으로 인해 데이터 값이 바뀌는 컴포넌트의 동적인 값으로, 웹에서는 state를 관리해주어야 한다.
- React 패키지의 useState() 함수를 사용해서 state를 관리 할 수 있다.
- 선언 형태: `const ['state변수이름', 'state의set함수이름'] = useState('state초기화값');`
- state에는 숫자, 문자열, 배열, 객체, 객체 배열 등 다양한 값이 쓰일 수 있다.
- state의 set 함수로 state에 원하는 값을 저장하며, 다른 함수에서 사용 가능하다.
- useState() 함수 사용 예:

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

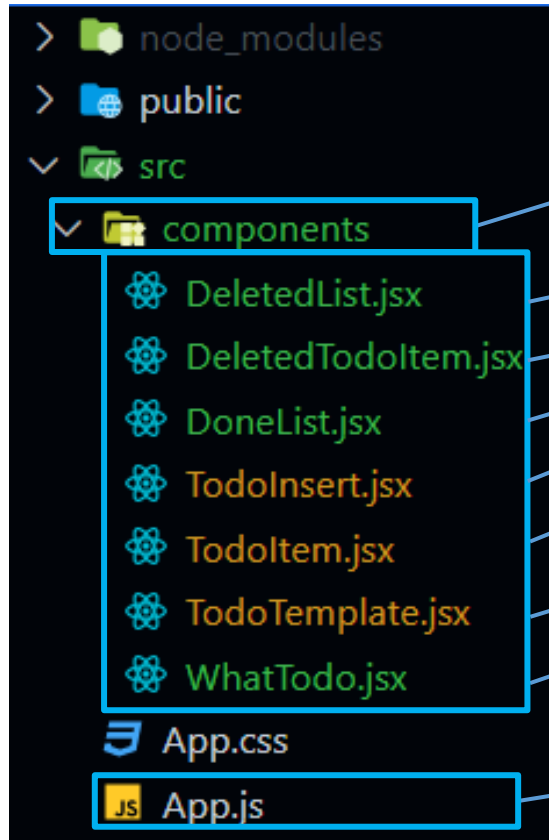
function Counter() {
  // number를 0으로 초기화
  const [number, setNumber] = useState(0);

  const onIncrease = () => {
    setNumber(num => num + 1);
  }

  return (
    <div>
      <h1>{number}</h1>
      { /* 버튼 클릭 시 number가 1씩 증가 */ }
      <button onClick={onIncrease}>+1</button>
    </div>
  );
}

export default Counter;
```

실습 파일 구조



- 7주차에서 실제 구현한 컴포넌트를 저장하는 디렉토리 (프로젝트 최초 생성 시 존재 x, 직접 추가)
- 삭제된 항목들을 보여주는 컴포넌트
- 삭제된 항목 하나를 표현하는 컴포넌트
- 체크된 항목들을 보여주는 컴포넌트
- 항목 입력 창과 Enter 버튼에 대한 컴포넌트
- 할 일 항목 하나(체크된, 체크되지 않은 항목)에 대한 컴포넌트
- 각 컴포넌트를 감싸는 전체 컴포넌트
- 체크되지 않은 항목에 대한 컴포넌트
- 실제 react 코드를 생성하는 부분 (실습에서는 가장 바깥에 위치한 TodoTemplate 컴포넌트를 불러온다.)

실습: To do List 프로그램 구현

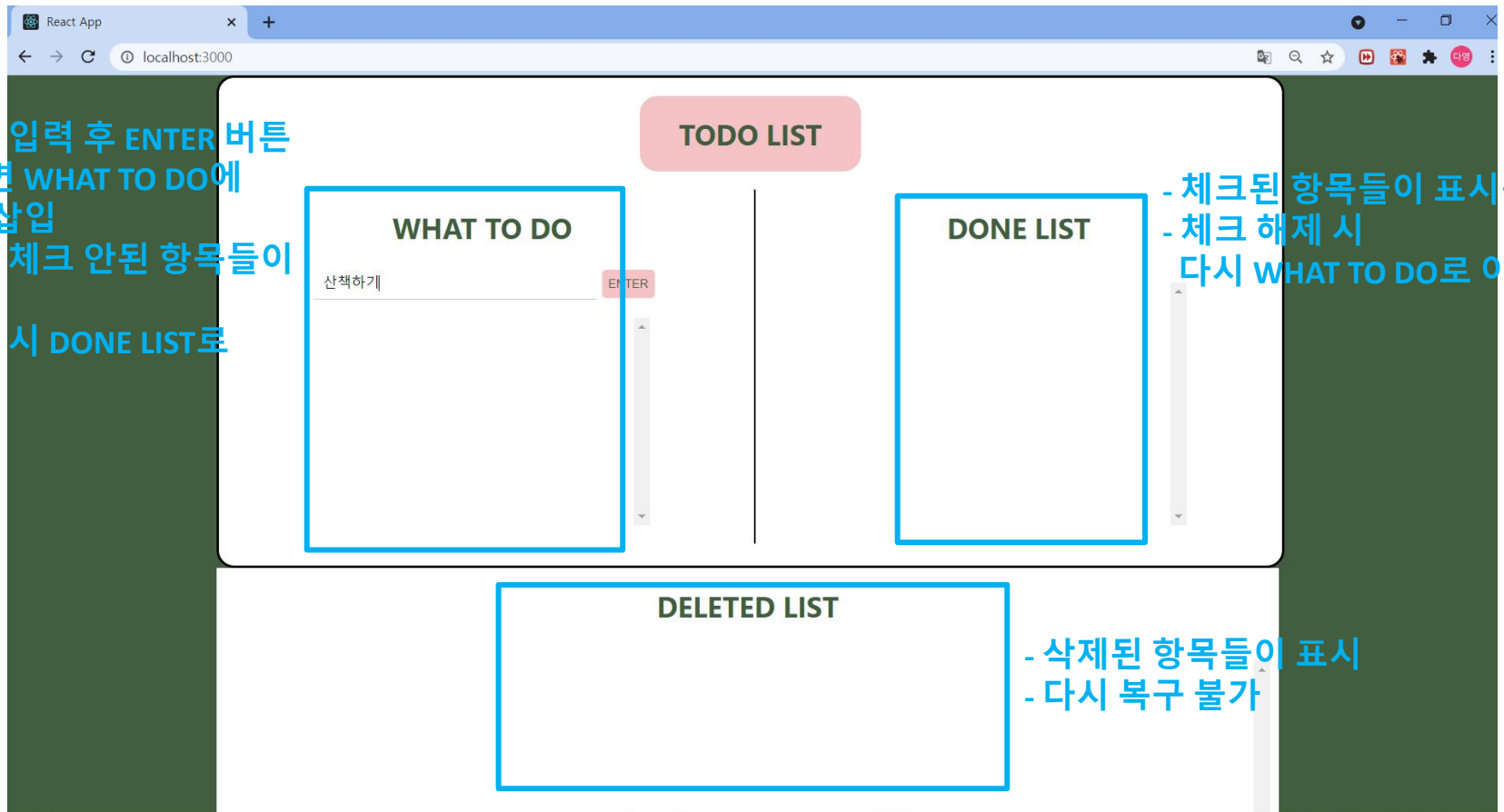
• 실습 완성 모습

- 새로 고침 시 그 동안 입력한 항목들 초기화

- 항목 입력 후 ENTER 버튼
누르면 WHAT TO DO에
항목 삽입
- 아직 체크 안된 항목들이
표시
- 체크 시 DONE LIST로
이동

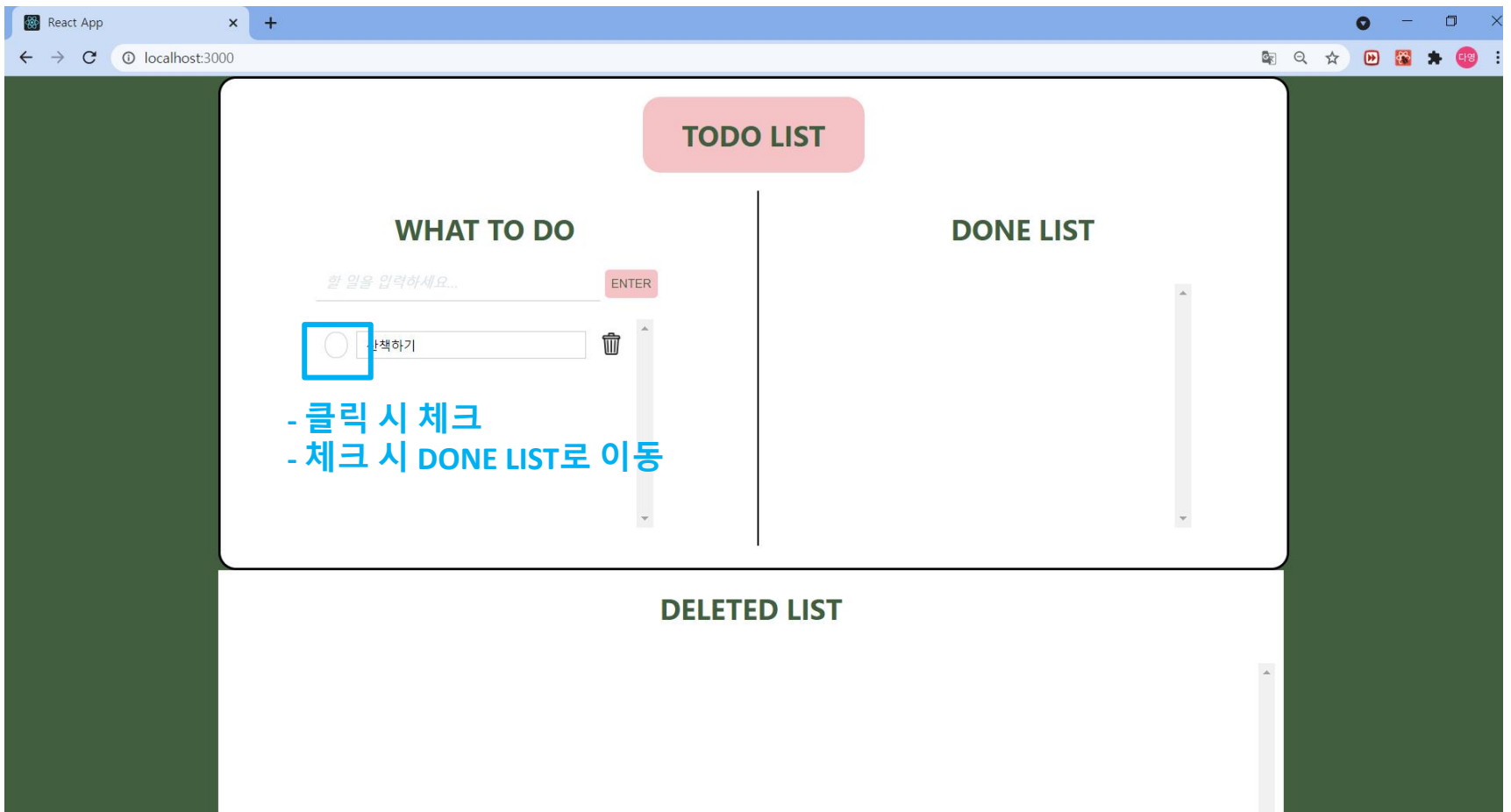
- 체크된 항목들이 표시됨
- 체크 해제 시
다시 WHAT TO DO로 이동

- 삭제된 항목들이 표시
- 다시 복구 불가



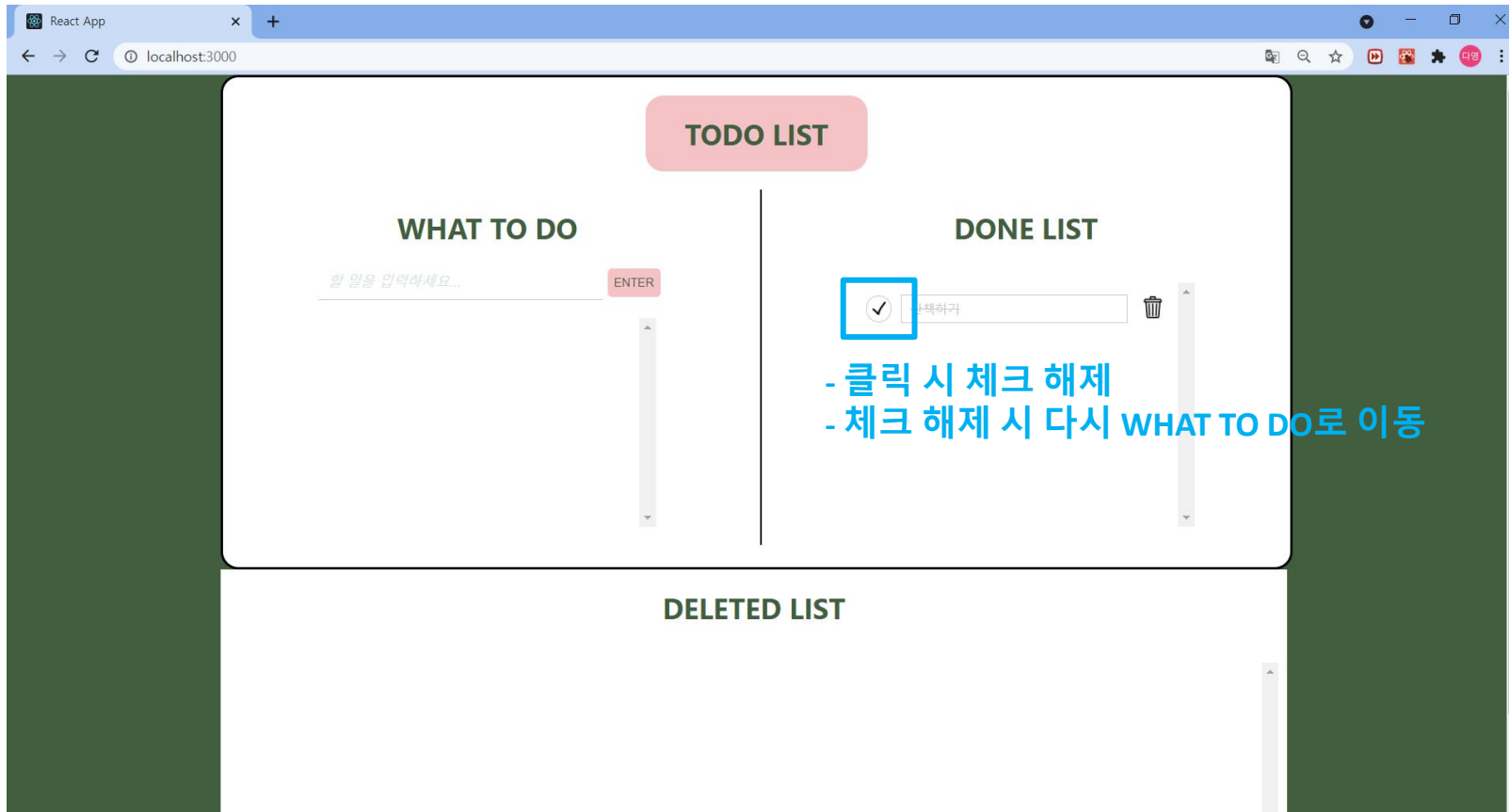
실습: To do List 프로그램 구현

- 실습 완성 모습



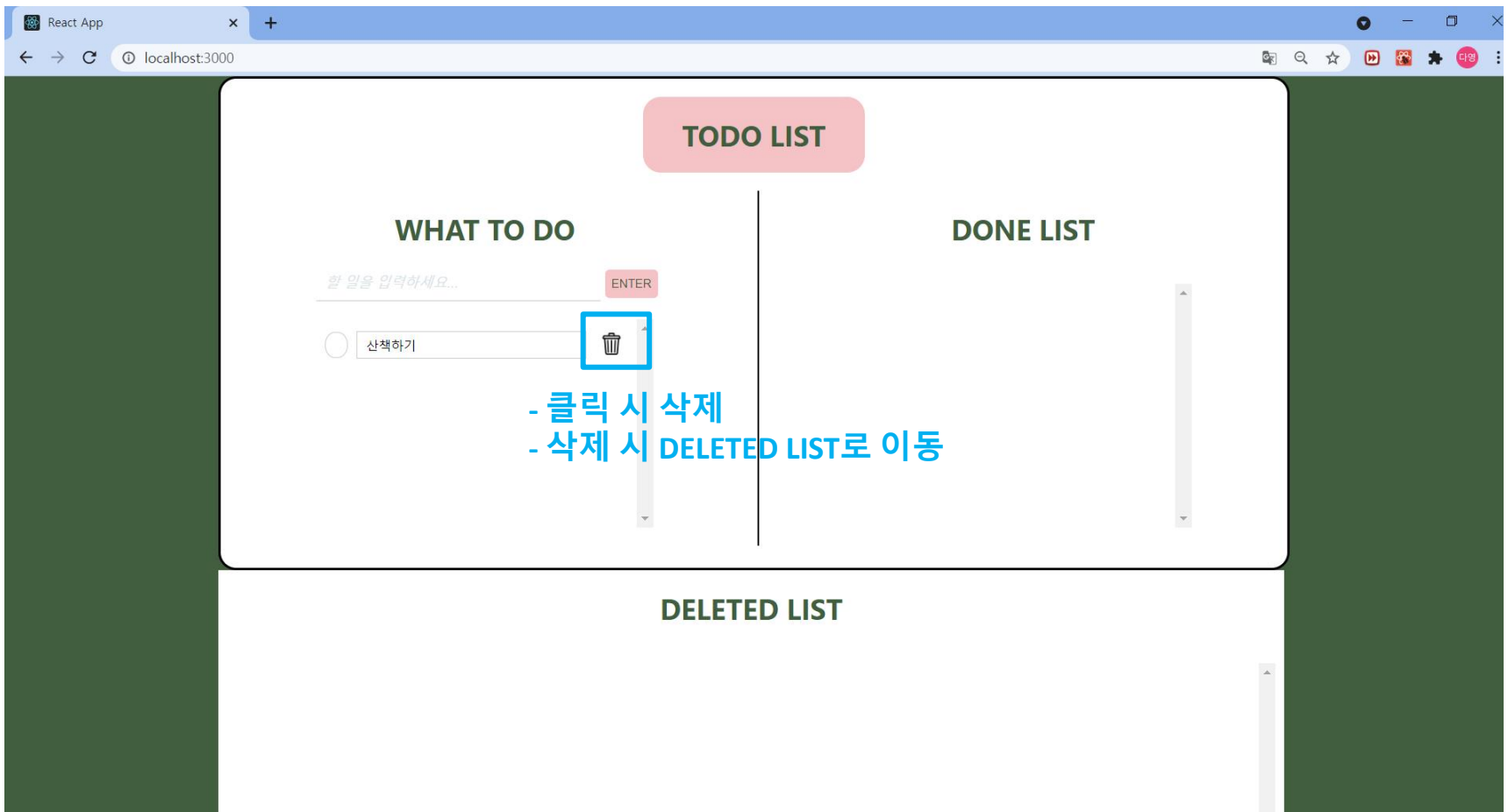
실습: To do List 프로그램 구현

- 실습 완성 모습



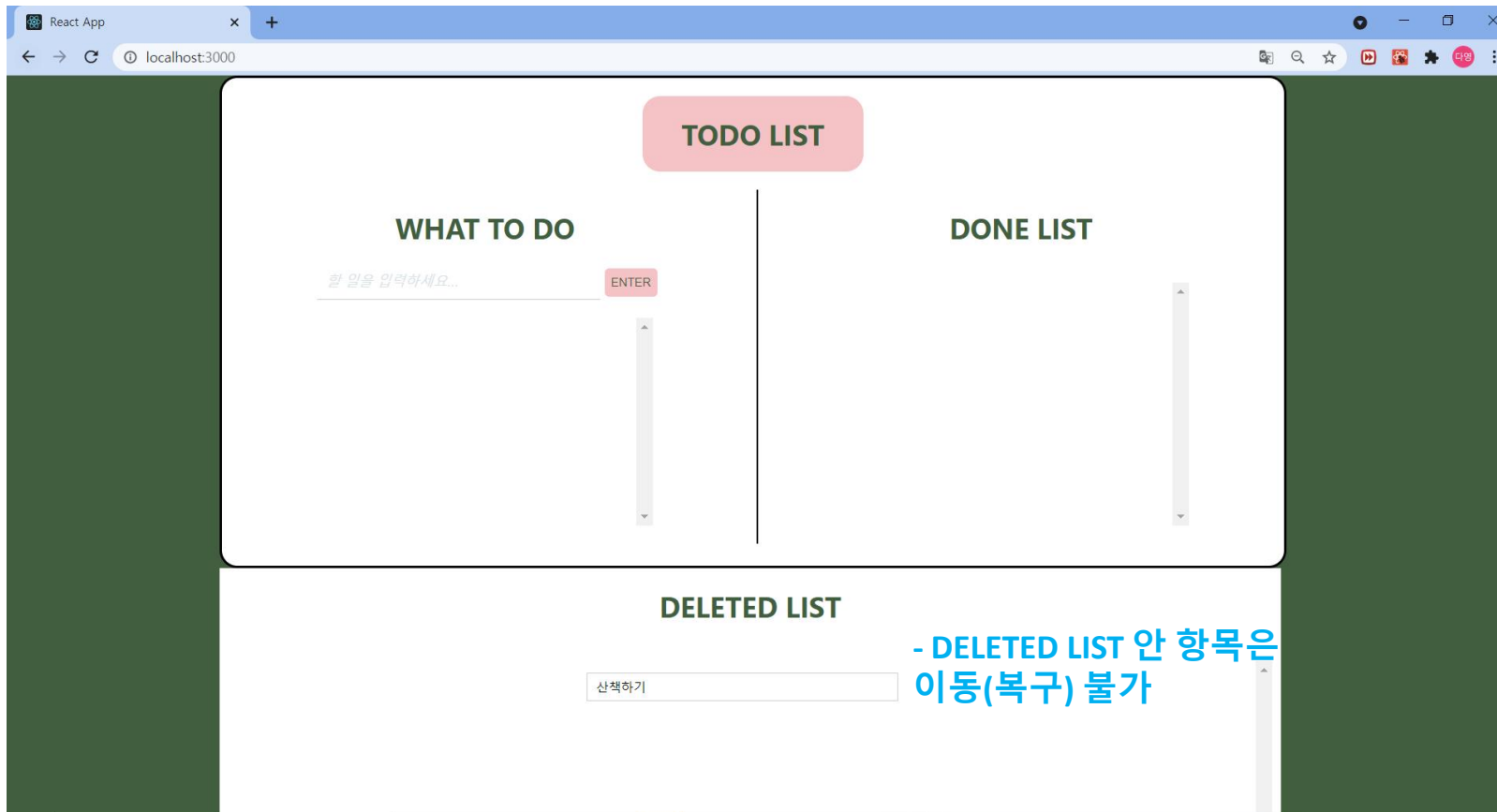
실습: To do List 프로그램 구현

- 실습 완성 모습



실습: To do List 프로그램 구현

- 실습 완성 모습



실습: 구현 내용

- src > components>TodoTemplate.jsx 안 함수들 구현
- onRemoveTodo 함수를 참고하여 각 함수의 기능을 구현한다.
- 구현 함수
 - onRemoveDoneTodo: 주어진 Done List 안 항목(체크된 항목)의 id로 Done List에서 해당 항목을 찾고 Deleted List에 추가한 후에, 기존 Done List에서 제외
 - onToggleToDo: 주어진 What To Do 안 항목(체크되지 않은 항목)을 찾아 항목 객체의 속성(키)을 사용하여 체크한 후, Done List에 추가하고 What To Do에서 제외
 - onToggleDoneToDo: 주어진 Done List 안 항목을 찾아 항목 객체의 속성(키)을 사용하여 체크를 해제한 후, What To Do에 추가하고 Done List에서 제외

과제: 보고서 – 실습 프로젝트 정리

- 각 주요 컴포넌트들 간의 상호 관계 및 구조에 대해 설명
- 실습 프로젝트의 주요 함수들의 기능 및 코드 내용 설명
- 설명한 함수들 간의 상호 관계에 대해 설명

참고

- JavaScript 레퍼런스 (MDN): <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>
- JavaScript 레퍼런스 (MSDN): [https://docs.microsoft.com/ko-kr/previous-versions/visualstudio/visual-studio-2010/z688wt03\(v=vs.100\)?redirectedfrom=MSDN](https://docs.microsoft.com/ko-kr/previous-versions/visualstudio/visual-studio-2010/z688wt03(v=vs.100)?redirectedfrom=MSDN)
- React 공식 튜토리얼 : <https://ko.reactjs.org/tutorial/tutorial.html>
- React 프로젝트 생성 및 실행: <https://github.com/facebook/create-react-app>