

React

강사 : 백현숙

React란

- 리액트란 사용자인터페이스를 만들기 위한 JavaScript 라이브러리입니다
- 특징
 - 선언형
 - React는 상호작용이 많은 UI를 만들 때 생기는 어려움을 줄여줍니다.
 - 애플리케이션의 각 상태에 대한 간단한 뷰만 설계하면 React는 데이터가 변경됨에 따라 적절한 컴포넌트만 효율적으로 갱신하고 렌더링합니다.
 - 속도가 빠릅니다. 전체가 아니라 데이터가 변경된 컴포넌트만 다시 화면에 보여줍니다.
 - 컴포넌트 기반
 - 스스로 상태 관리가 가능한 캡슐형 컴포넌트 입니다.
 - 기술 스택의 나머지 부분에는 관여하지 않기 때문에, 기존 코드를 다시 작성하지 않고도 React의 새로운 기능을 이용해 개발할 수 있습니다.

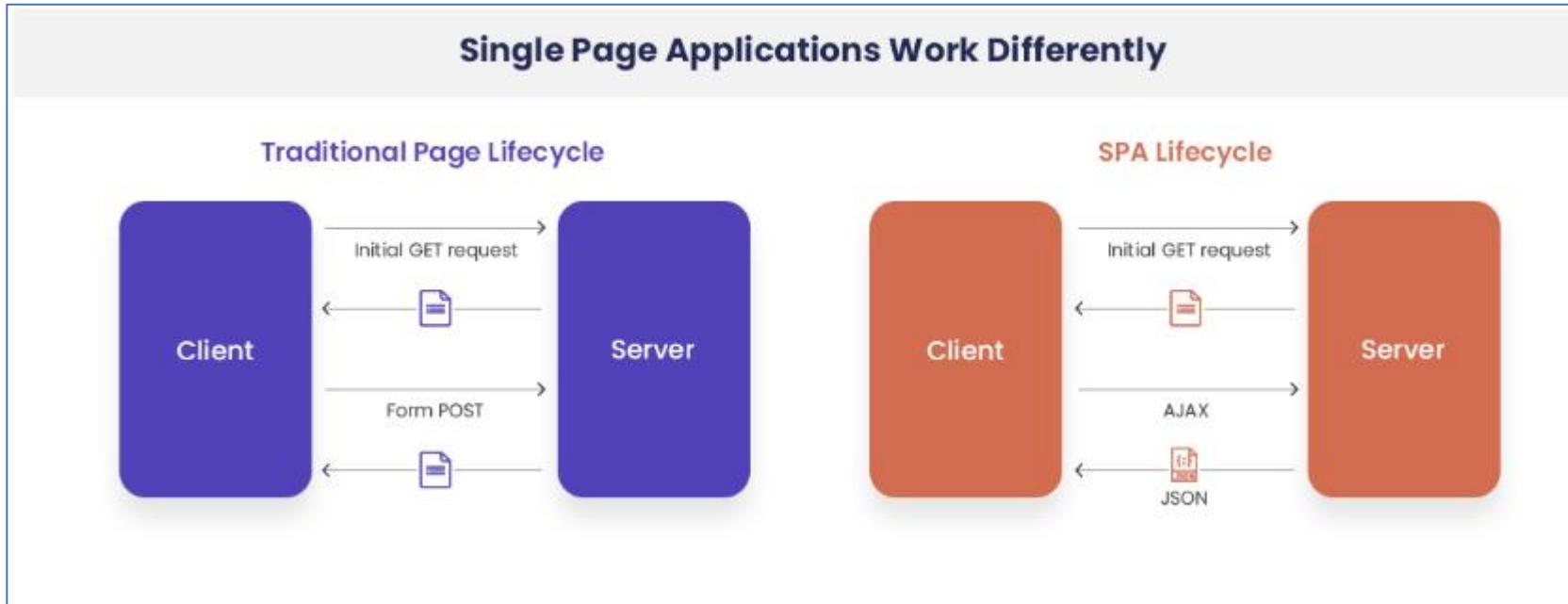
리액트 사용이유

- Naver Vibe, Flipkart, Instagram 이런 웹사이트 사이트들을 보면 페이지 전환시 새로고침 없이도 부드럽게 동작합니다. 이런 사이트들을 **Single Page Application** 이라고 합니다.
- **SPA(Single Page Application)**은 하나의 HTML 페이지로 구성된 애플리케이션을 말합니다. 전통적인 웹 애플리케이션에서는 새로운 페이지를 요청할 때마다 서버로부터 전체 HTML 페이지를 받아오고 브라우저가 이를 다시 렌더링합니다. 그러나 SPA에서는 처음에 한 번만 전체 페이지를 로드하고, 이후에는 사용자의 요청에 따라 필요한 데이터만 서버에서 불러와서 부드러운 동작을 하고 속도도 빠르게 진행됩니다.
- SPA를 지원하는 라이브러리는 리액트, 앵귤러, 뷰등이 있습니다. 우리나라에서는 현재 리액트와 뷰 두 가지의 프레임워크가 인기가 있습니다.
- 초기 로딩은 느리나 로딩 후의 처리 속도는 빠릅니다. Ajax기반으로 서버와 통신을 합니다. 예
- SEO(Search Engine Optimization) 에는 적합하지 않습니다. SEO란 검색 최적화를 말합니다.

SPA 특징

- 기존의 웹페이지는 client/web browser (프론트) 가 server(백엔드) 에게 html,javascript,css 등 화면에 보여줄 데이터를 요청하고 , 서버는 요청된 파일을 client에게 뿌려주는 형태이다. 즉 전체 문서가 서버로 부터 다시 불려온다.
- SPA는 Client (Browser) 가 필요한 부분만 자바스크립트(Ajax통신과 JSON객체)를 사용하여 필요시 업데이트를 해주는 방식이다.
- 단점
 - 앱의 규모가 커지면 자바스크립트 파일이 너무 커진다. (페이지 로딩시 실제 사용자가 방문하지 않을 수도 있는 페이지의 스크립트도 불러옴)
 - SEO 가 어렵다. 페이지 정보를 수집하지 못하여 페이지 검색이 힘들다. (구글,네이버,다음) 같은 대형 포털사이트에 검색 안될 경우도 있다.

SPA



출처 : <https://velog.io/@cyongchoi/Spa-%EB%9E%80>

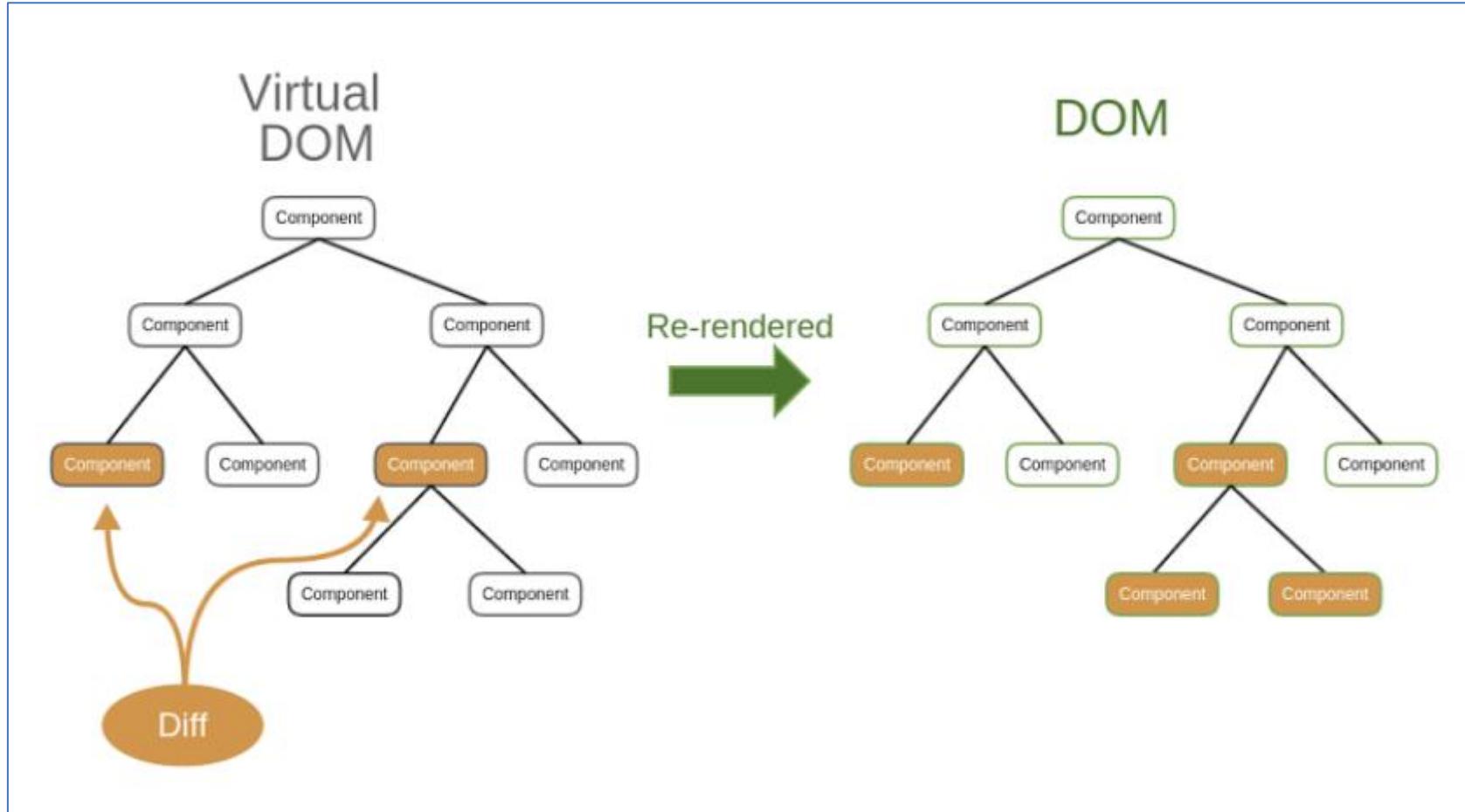
소개

- react란 사용자 인터페이스를 만들기위해 페이스북과 인스타그램에서 개발한 오픈소스 자바스크립트 라이브러리입니다.
- 사용자 인터페이스(User Interface)에 집중하며, Virtual DOM을 통해 속도와 편의를 높이고, **단방향** 데이터플로우를 지원하여 보일러플레이트 코드(꼭 필요하면서 간단한 기능인데, 많은 코드를 필요로 하는 코드, Dto 클래스 같은 경우 getter, setter이 너무 많다)를 감소시켰습니다.
- 많은 사람들이 React를 MVC의 V를 고려하여 선택합니다.

Virtual Dom

- virtual dom 이란 브라우저의 실제 DOM(Document Object Model)과는 별도로, 메모리 내에서 가상으로 존재하는 DOM의 사본입니다. 이는 주로 React와 같은 라이브러리에서 사용되며, 웹 애플리케이션의 성능을 최적화하는 데 중요한 역할을 합니다.
- Virtual DOM이 더 빠른 이유는 DOM 자체를 자주 수정하지 않고, 메모리 상에서 가상으로 변화된 상태를 먼저 처리한 뒤 최소한의 변경 사항만을 실제 DOM에 반영하기 때문입니다.

Virtual DOM

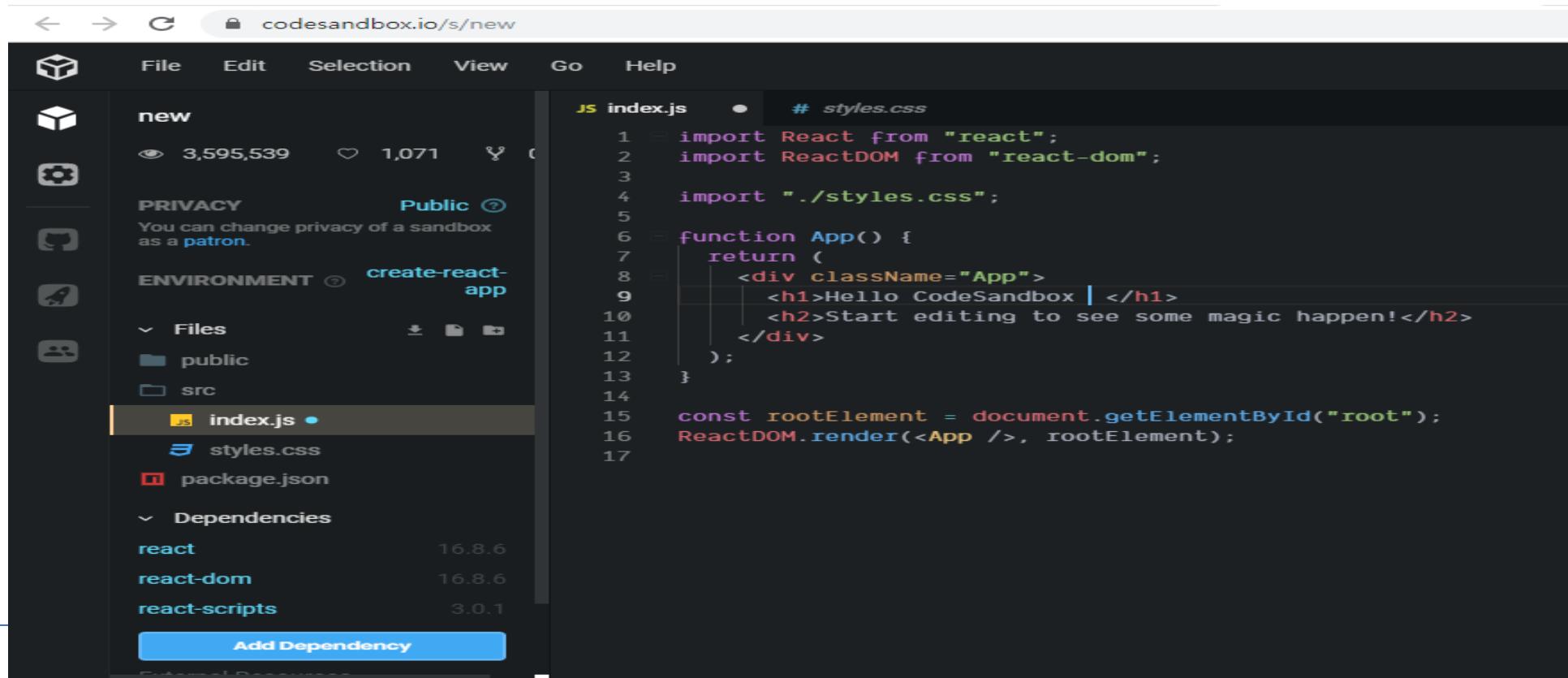


Virtual dom 과 dom 구조의 차이

- 일반 DOM
 - DOM의 변경이 발생하면 브라우저는 즉시 DOM을 업데이트하고, Reflow 및 Repaint를 진행합니다. 이 과정은 매우 비효율적일 수 있으며, 작은 변경이 있어도 전체 DOM을 재계산하게 됩니다.
- Virtual DOM
 - 상태가 변경되면 먼저 Virtual DOM이 업데이트됩니다. 그리고 Virtual DOM은 이전 상태와 변경된 상태를 비교(diffing)합니다. 이 비교 과정을 통해 바뀐 부분만을 찾아내고, 그 부분만을 실제 DOM에 반영합니다. 즉, DOM 전체를 수정하지 않고 필요한 부분만 최소한으로 수정합니다.

react 시도하기

- 온라인 플레이그라운드를 이용해 미리 react에 대해 살펴볼 수 있습니다
- 코드 샌드 박스 : <https://codesandbox.io/s/new>

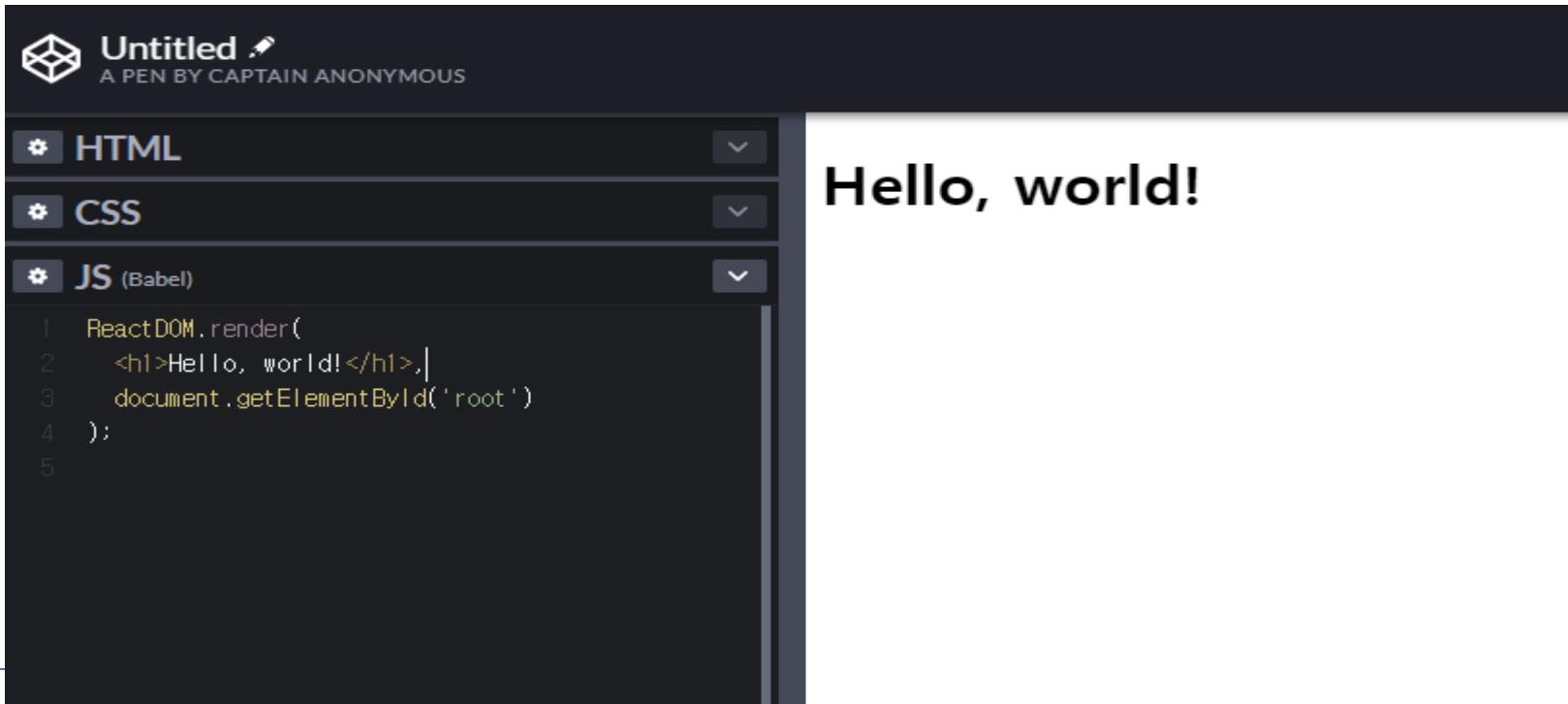


The screenshot shows the CodeSandbox web interface. On the left, there's a sidebar with icons for navigation, a search bar, and sections for 'new' projects, 'PRIVACY', 'ENVIRONMENT', 'Files' (containing 'public' and 'src' folders with 'index.js', 'styles.css', and 'package.json'), and 'Dependencies' (listing 'react', 'react-dom', and 'react-scripts'). At the bottom of the sidebar is a blue button labeled 'Add Dependency'. The main area is a code editor with dark-themed syntax highlighting. It displays the contents of 'index.js' as follows:

```
index.js  # styles.css
1 import React from "react";
2 import ReactDOM from "react-dom";
3
4 import "./styles.css";
5
6 function App() {
7   return (
8     <div className="App">
9       <h1>Hello CodeSandbox</h1>
10      <h2>Start editing to see some magic happen!</h2>
11    </div>
12  );
13}
14
15 const rootElement = document.getElementById("root");
16 ReactDOM.render(<App />, rootElement);
17
```

react 시도하기

- <https://codepen.io/pen?&editable=true&editors=0010>
- 회원가입 필요



The screenshot shows a dark-themed CodePen interface. At the top, it says "Untitled" with a pencil icon and "A PEN BY CAPTAIN ANONYMOUS". Below that are three dropdown menus: "HTML", "CSS", and "JS (Babel)". The "JS (Babel)" menu is open, displaying the following code:

```
1 ReactDOM.render(  
2   <h1>Hello, world!</h1>,|  
3   document.getElementById('root')  
4 );  
5
```

To the right of the code editor, the output area displays the text "Hello, world!" in a large, bold, black font.

카운터.html(전통적인 자바스크립트)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Parcel Sandbox</title>
    <meta charset="UTF-8" />
  </head>

  <body>
    <h2 id="number">0</h2>
    <div>
      <button id="increase">+1</button>
      <button id="decrease">-1</button>
    </div>
  </body>
</html>

<script>
  const number = document.getElementById("number");
  const increase = document.getElementById("increase");
  const decrease = document.getElementById("decrease");

  increase.onclick = () => {
    console.log("increase 가 클릭됨");
    number.innerHTML = parseInt(number.innerHTML)+1;
  };
  decrease.onclick = () => {
    console.log("decrease 가 클릭됨");
    number.innerHTML = parseInt(number.innerHTML)-1;
  };

</script>
```

6

+1 -1

카운터2.html(카운터가 여러개 필요할 경우)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Parcel Sandbox</title>
    <meta charset="UTF-8" />
  </head>

  <body>
    <div id="counter1">
      <h2 id="number">0</h2>
      <div>
        <button id="increase">+1</button>
        <button id="decrease">-1</button>
      </div>
    </div>

    <div id="counter2">
      <h2 id="number2">0</h2>
      <div>
        <button id="increase2">+1</button>
        <button id="decrease2">-1</button>
      </div>
    </div>
  </body>
</html>
```

```
<script>
  const number = document.getElementById("number");
  const increase = document.getElementById("increase");
  const decrease = document.getElementById("decrease");

  increase.onclick = () => {
    console.log("increase 가 클릭됨");
    number.innerHTML = parseInt(number.innerHTML)+1;
  };
  decrease.onclick = () => {
    console.log("decrease 가 클릭됨");
    number.innerHTML = parseInt(number.innerHTML)-1;
  };

  const number2 = document.getElementById("number2");
  const increase2 = document.getElementById("increase2");
  const decrease2 = document.getElementById("decrease2");

  increase2.onclick = () => {
    console.log("increase 가 클릭됨");
    number2.innerHTML = parseInt(number2.innerHTML)+1;
  };
  decrease2.onclick = () => {
    console.log("decrease 가 클릭됨");
    number2.innerHTML = parseInt(number2.innerHTML)-1;
  };
</script>
```

react

리액트 라이브러리를 간단한 CDN 서비스를 이용해 사용할 수 있습니다

아래는 필요한 최소한의 라이브러리 입니다.

초창기에는 jsp 페이지나 html 페이지에서 사용했는데 최근에는 nodejs 기반의 프론트엔드를 따로 설치하고 그 위에 react 를 설치합니다

```
<script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
<script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
```

react 첫번째 예제(hello.html)

```
<script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
```

react 첫번째 예제(hello.html)

```
<body>

<div id="root"></div>

<script type="text/babel"> <!-- text/babel 꼭 있어야 한다 →

  ReactDOM.createRoot(rootNode).render( // createRoot : render에서 바뀜, react 18부터 지원

    <React.StrictMode> // React.StrictMode : 문제가 발생할경우 알려주기 위함이지 화면에 출력되지

      //않는다

      <h1>Hello React</h1>

    </React.StrictMode>,

  );

</script>

</body>
```

ReactDOM.createRoot 대 ReactDOM.render

- ReactDOM.createRoot는 React 18에서 추가됨
- Concurrent Mode 지원:
 - createRoot는 동시성 모드를 기본적으로 지원합니다. 이를 통해 React는 사용자 인터페이스를 비동기적으로 렌더링할 수 있어, 복잡한 UI의 응답성을 개선하고, 시간이 오래 걸리는 작업이 전체 UI를 차단하는 것을 방지할 수 있습니다.
- Automatic Batching:
 - 자동 배치 기능이 개선되었습니다. 여러 상태 업데이트가 동시에 발생할 때, 이 상태 변화들이 하나의 렌더링으로 모아지며 더 효율적인 렌더링을 진행합니다.
- SSR (서버 사이드 렌더링) 지원:
 - createRoot는 서버 사이드 렌더링(SSR)을 더 잘 지원하며, 클라이언트와 서버 간의 하이드레이션 성능을 개선합니다.

hello2.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>

  <script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

</head>
<body>
  <div id="root"></div>

  <script type="text/babel">
    let rootNode= document.getElementById("root");
    ReactDOM.createRoot(rootNode).render(
      <React.StrictMode>
        <h1 style={{"color": "red"}}>Hello React</h1>
        <p>리액트를 배워보자</p>
        <p>리액트는 재미있는 라이브러리 입니다. </p>
      </React.StrictMode>,
    );
  </script>
</body>
</html>
```

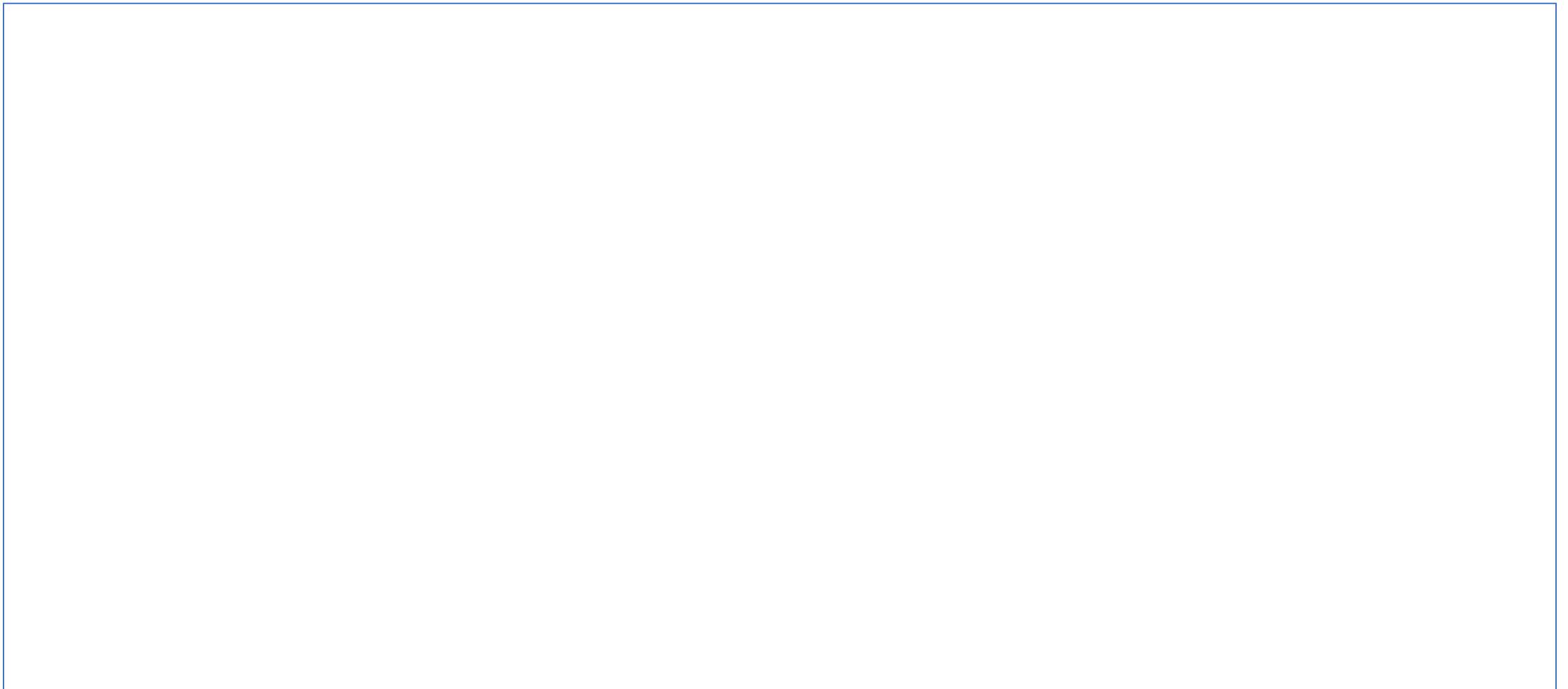
React Event

SyntheticEvent

- React가 기본 DOM 이벤트를 감싸는 Wrapper 객체로, React에서 사용하는 이벤트 시스템입니다.
- SyntheticEvent는 React의 가상 DOM에서 발생하는 이벤트를 처리하는 방식이며, 실제 브라우저의 이벤트와 동일한 인터페이스를 가집니다.
- 브라우저 이벤트와 다르게 React는 이벤트 풀링(event pooling, 재사용 패턴)을 사용하여 성능을 최적화합니다.
- 즉 SyntheticEvent는 이벤트가 끝나면 재사용됩니다.

React 의 이벤트 유형

- 마우스 이벤트: onClick, onMouseEnter, onMouseLeave, onMouseMove, onMouseDown, onMouseUp 등
- 키보드 이벤트: onKeyDown, onKeyUp, onKeyPress
- 폼 이벤트: onChange, onSubmit, onInput
- 포커스 이벤트: onFocus, onBlur
- 터치 이벤트: onTouchStart, onTouchMove, onTouchEnd



event1.html

```
• <!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>

  <script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

</head>
<body>
  <div id="root"></div>
  <script type="text/babel">
    let rootNode= document.getElementById("root");
    ReactDOM.createRoot(rootNode).render(
      <div>
        <h1>Event Processing</h1>
        <button type="button" onClick={alert('이벤트 테스트 111')}>클릭</button><br/>
        <button type="button" onClick={()=>{alert('이벤트 테스트 222')}}>클릭</button>
      </div>
    );
  </script>
</body>
</html>
```

onclick={ 람다로 전
달해줘야 한다 }

event2.html

```
<body>                                별도의 함수형 컴포넌트 만들기
  <div id="root"></div>
  <script type="text/babel">
    ReactDOM.render(
      <div>
        <App/>
      </div>,
      document.getElementById('root')
    );

    function App() {
      return(
        <div>
          <h1>Event Processing</h1>
          <button type="button" onClick={()=>{alert('Hello')}}>클릭</button>
        </div>
      )
    }
  </script>

</body>
```

event3.html

```
<div id="root"></div>
<script type="text/babel">
  ReactDOM.render(
    <div>
      <App/>
    </div>,
    document.getElementById('root')
  );

  function App() {
    const onclick = ()=>{
      alert("클릭");
    }
    return(
      <div>
        <h1>Event Processing</h1>
        <button type="button" onClick={onclick}>클릭 </button>
      </div>
    )
  }
</script>
```

별도의 함수형 컴포넌트 만들기

카운터2-react로 구축하기

```
<body>
  <div id="root"></div>
  <script type="text/babel">
    let rootNode= document.getElementById("root");
    ReactDOM.createRoot(rootNode).render(
      <React.StrictMode>
        <Counter/>
      </React.StrictMode>,
    );

    function Counter() {
      const [number, setNumber]=React.useState(0);
      const increase = ()=>{          setNumber(number+1);      }
      const decrease = ()=>{          setNumber(number-1);      }
      return(
        <div>
          <h1>{number}</h1>
          <button type="button" onClick={increase}>+1</button>
          <button type="button" onClick={decrease}>-1</button>
        </div>
      )
    }
  </script>

</body>
```

클래스형 기반 컴포넌트

```
<body>
  <div id="root"></div>
  <script type="text/babel">
    class AppClass extends React.Component
    {
      render(){
        return(
          <div>
            <h1>클래스기반 컴포넌트</h1>
          </div>
        )
      }
    }

    let rootNode= document.getElementById("root");
    ReactDOM.createRoot(rootNode).render(
      <React.StrictMode>
        <AppClass/>
      </React.StrictMode>
    );
  </script>
```

클래스 기반 컴포넌트(state)

```
<script type="text/babel">
  class AppClass extends React.Component
    constructor(props) { //생성자
      this.state = {
        name: "홍길동",
        age: 20,
      };
    }
    render() {
      const { name, age } = this.state;
      return (
        <div>
          <p>이름 : {name} 나이: {age}</p>
          <p>주소 : {this.props.address}</p>
        </div>
      );
    }
  }

let rootNode= document.getElementById("root");
ReactDOM.createRoot(rootNode).render(
  <React.StrictMode>
    <AppClass address="서울시"/>
  </React.StrictMode>
);
```

state : 컴포넌트내의 값을 저장한다

함수기반 컴포넌트

```
<div id="root"></div>
<script type="text/babel">

    function App() {
        return(
            <div>
                <h1>함수기반컴포넌트</h1>
            </div>
        )
    }

    let rootNode= document.getElementById("root");
    ReactDOM.createRoot(rootNode).render(
        <React.StrictMode>
            <App/>
        </React.StrictMode>
    );
}
```

함수기반컴포넌트(state)

```
<div id="root"></div>
<script type="text/babel">
  function App(props) {
    const [name, setName]=React.useState("홍길동");
    const [age, setAge]=React.useState(23);
    return(
      <div>
        <h1>함수기반컴포넌트</h1>
        <div>
          <p>이름 : {name} 나이: {age}</p>
          <p>주소 : {props.address}</p>
        </div>
      </div>
    )
  }

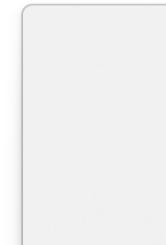
  let rootNode= document.getElementById("root");
  ReactDOM.createRoot(rootNode).render(
    <React.StrictMode>
      <App address ="서울시"/>
    </React.StrictMode>
  );
}
```

props

- 부모 컴포넌트에서 자식 컴포넌트에게 값을 보낼 때 사용한다.

```
<div id="root"></div>
<script type="text/babel">
  let rootNode= document.getElementById("root");
  ReactDOM.createRoot(rootNode).render(
    <React.StrictMode>
      <App title="props 학습" contents="props 는 부모 컴포넌트에서 자식 컴포넌트에 값을 전달하는 객체이다"/>
    </React.StrictMode>,
  );
}

function App(props) {
  const onclick = ()=>{
    alert("클릭");
  }
  return(
    <div>
      <h1>{props.title}</h1>
      <p>{props.contents}</p>
    </div>
  )
}
</script>
```



props2

- props 전달시 해체기능

```
<div id="root"></div>
<script type="text/babel">
  let rootNode= document.getElementById("root");
  ReactDOM.createRoot(rootNode).render(
    <React.StrictMode>
      <App title="props 학습" contents="props 는 부모 컴포넌트에서 자식 컴포넌트에 값을 전달하는 객체이다"/>
    </React.StrictMode>,
  );
//매개변수 전달시 const {title, content} = props; 해체가 일어난다
function App({title, contents}) {
  return(
    <div>
      <h1>{title}</h1>
      <p>{contents}</p>
    </div>
  )
}
</script>
```

props 를 이용해 부모컴포넌트 함수 전달하기

```
<script type="text/babel">
  let rootNode= document.getElementById("root");

  ReactDOM.createRoot(rootNode).render(
    <React.StrictMode>
      <App />
    </React.StrictMode>,
  );

  //매개변수 전달시 const {title, content} = props; 해체가 일어난다
  //데이터는 위에서 아래로
  //함수는 아래에서 위로 =
  function App({title, contents}) {
    const myfunc = ()=>{ alert('부모함수'); }
    return(
      <div>
        <Child parentFunc={myfunc} />
      </div>
    )
  }

  function Child({parentFunc}) {
    return(
      <div>
        <h1>자식객체입니다.</h1>
        <button type="button" onClick={parentFunc}>클릭</button>
      </div>
    )
  }
</script>
```

사칙연산기

```
<script type="text/babel">
  let rootNode= document.getElementById("root");
  ReactDOM.createRoot(rootNode).render(
    <React.StrictMode>
      <Counter/>
    </React.StrictMode>,
  );

  function Counter() {
    const [x, setX]=React.useState(0);
    const [y, setY]=React.useState(0);
    const [z, setZ]=React.useState(0);

    function xChange (e) { // <- input값으로 text 변경 함수
      setX(e.target.value);
    };
    function yChange (e){ // <- input값으로 text 변경 함수
      setY(e.target.value);
    };
    const add = ()=>{      setZ(parseInt(x)+parseInt(y));      }
    const sub = ()=>{      setZ(parseInt(x)-parseInt(y));      }

    return(
      <div>
        x : <input type="text" onChange={xChange}>/> <br/>
        y : <input type="text" onChange={yChange}>/> <br/>
        <h1>{z}</h1>
        <button type="button" onClick={add}>+</button>
        <button type="button" onClick={sub}>-</button>
      </div>
    )
  }
</script>
```

props, state, input태그

```
<div id="root"></div>
<script type="text/babel">
  let rootNode= document.getElementById("root");
  ReactDOM.createRoot(rootNode).render(
    <React.StrictMode>
      <Score name="홍길동" kor="90" eng="80" mat="60"/>
    </React.StrictMode>,
  );
  function Score(props) {
    console.log( props );
    const [name, setName]=React.useState(props.name);
    const [kor, setKor]=React.useState(props.kor);
    const [eng, setEng]=React.useState(props.eng);
    const [mat, setMat]=React.useState(props.mat);
    const [result, setResult]=React.useState(0);

    function nameChange (e) { // <- input값으로 text 변경 함수
      setName(e.target.value);
    };
    function korChange (e){ // <- input값으로 text 변경 함수
      setKor(e.target.value);
    };
    function engChange(e){ // <- input값으로 text 변경 함수
      setEng(e.target.value);
    };
    function matChange (e){ // <- input값으로 text 변경 함수
      setMat(e.target.value);
    };
  };
</script>
```

```
const confirm = ()=>{ setResult(parseInt(kor)+parseInt(eng)+parseInt(mat));
const reset = ()=>{ setName(""); setKor(0); setEng(0); setMat(0); }

return(
  <div>
    <h1>{props.name}</h1>
    name : <input type="text" onChange={nameChange} value={name} /> <br/>
    kor : <input type="text" onChange={korChange} value={kor}/> <br/>
    eng : <input type="text" onChange={engChange} value={eng}/> <br/>
    mat : <input type="text" onChange={matChange} value={mat}/> <br/>
    <h1>{result}</h1>
    <button type="button" onClick={confirm}>확인</button>
    <button type="button" onClick={reset}>리셋</button>
  </div>
)
</script>
```

앞의 예제 결과화면

홍길동

name : 임꺽정

kor : 90

eng : 80

mat : 80

250

확인

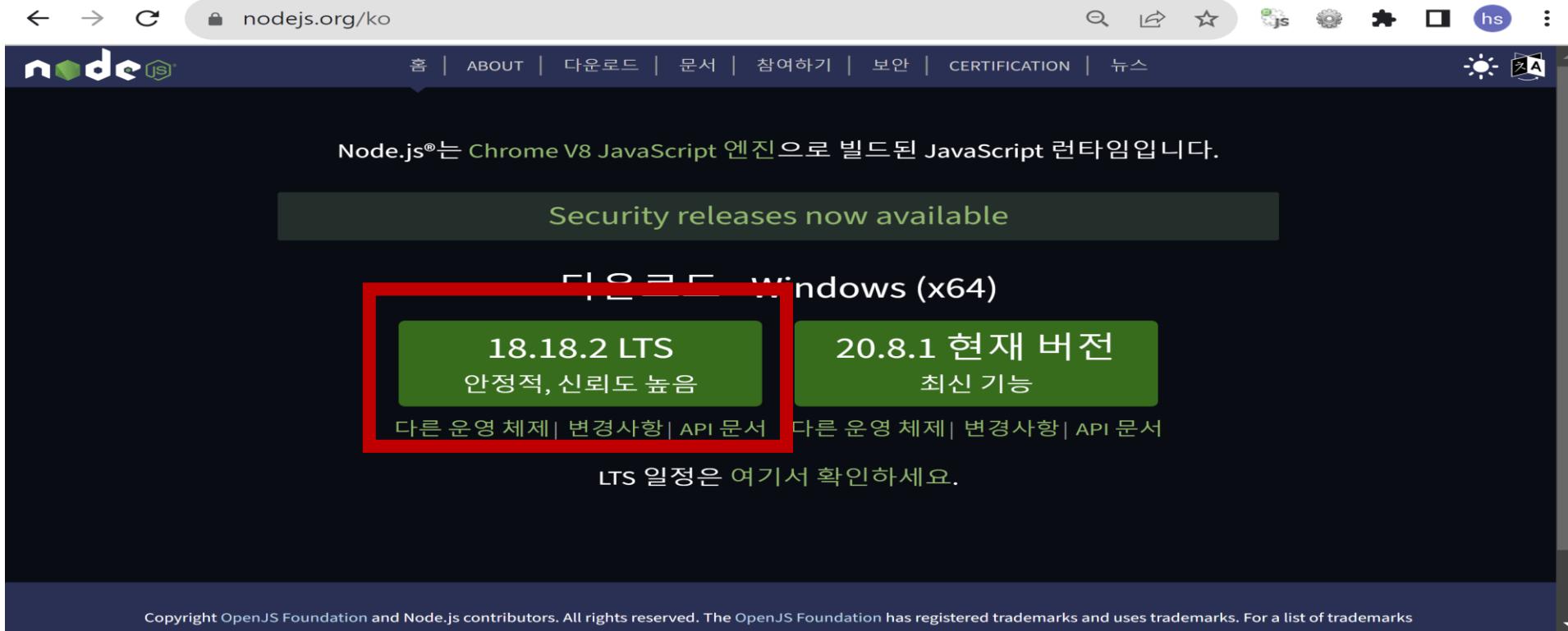
리셋

설치해야 할 것들

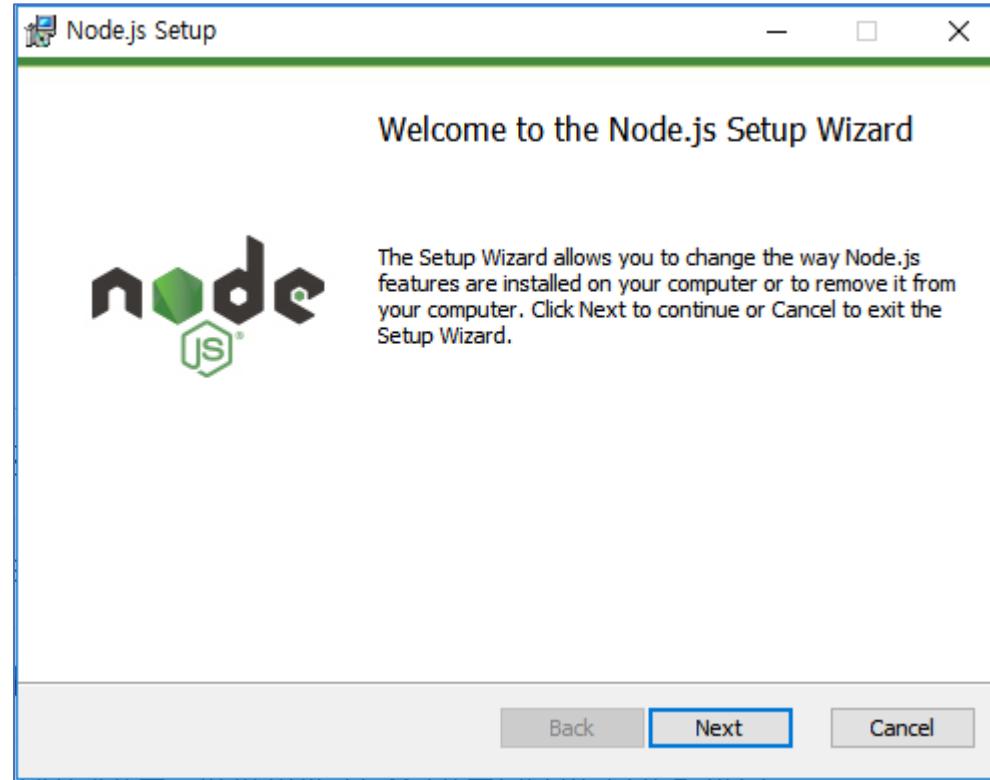
- **Node.js**: Webpack 과 Babel 같은 도구들이 자바스크립트 런타임인 Node.js 를 기반으로 만들어져 있습니다. React를 구동시키기 위해 필요합니다
- **Yarn**: Yarn 은 조금 개선된 버전의 npm 이라고 생각하시면 됩니다. npm 은 Node.js 를 설치하게 될 때 같이 딸려오는 패키지 매니저 도구입니다. 프로젝트에서 사용되는 라이브러리를 설치하고 해당 라이브러리들의 버전 관리를 하게 될 때 사용하죠. 우리가 Yarn 을 사용하는 이유는, [더 나은 속도, 더 나은 캐싱 시스템](#)을 사용하기 위함입니다.(맥사용자)
- **코드 에디터**: 그리고, 코드 에디터를 준비하세요. 여러분이 좋아하는 에디터가 있다면, 따로 새로 설치하지 않고 기존에 사용하시던걸 사용하셔도 됩니다. 저는 주로 VSCode 를 사용합니다. 이 외에도, Atom, WebStorm, Sublime 같은 훌륭한 선택지가 있습니다.
- 윈도우의 경우, [Git for Windows](#) 를 설치해서 앞으로 터미널에 무엇을 입력하라는 내용이 있으면 함께 설치되는 Git Bash 를 사용하세요.

nodejs 설치하기

- <https://nodejs.org/ko/> 안정화버전을 다운받자



nodejs 설치



yarn 설치

- cmd
- npm install yarn -g

react 프로젝트 만들기

visual studio code에서 유용한 확장팩

- ESLint 자바스크립트 문법 체크
- Relative Path 상대 경로에 있는 파일 경로를 편하게 작성할 수 있는 도구
- HTML5
- Guides 들여쓰기 가이드라인을 잡아준다
- Reactjs code snippets(리액트 관련 스니펫 모음, 제작자가 Charalampos Karypidis 인걸로 설치)

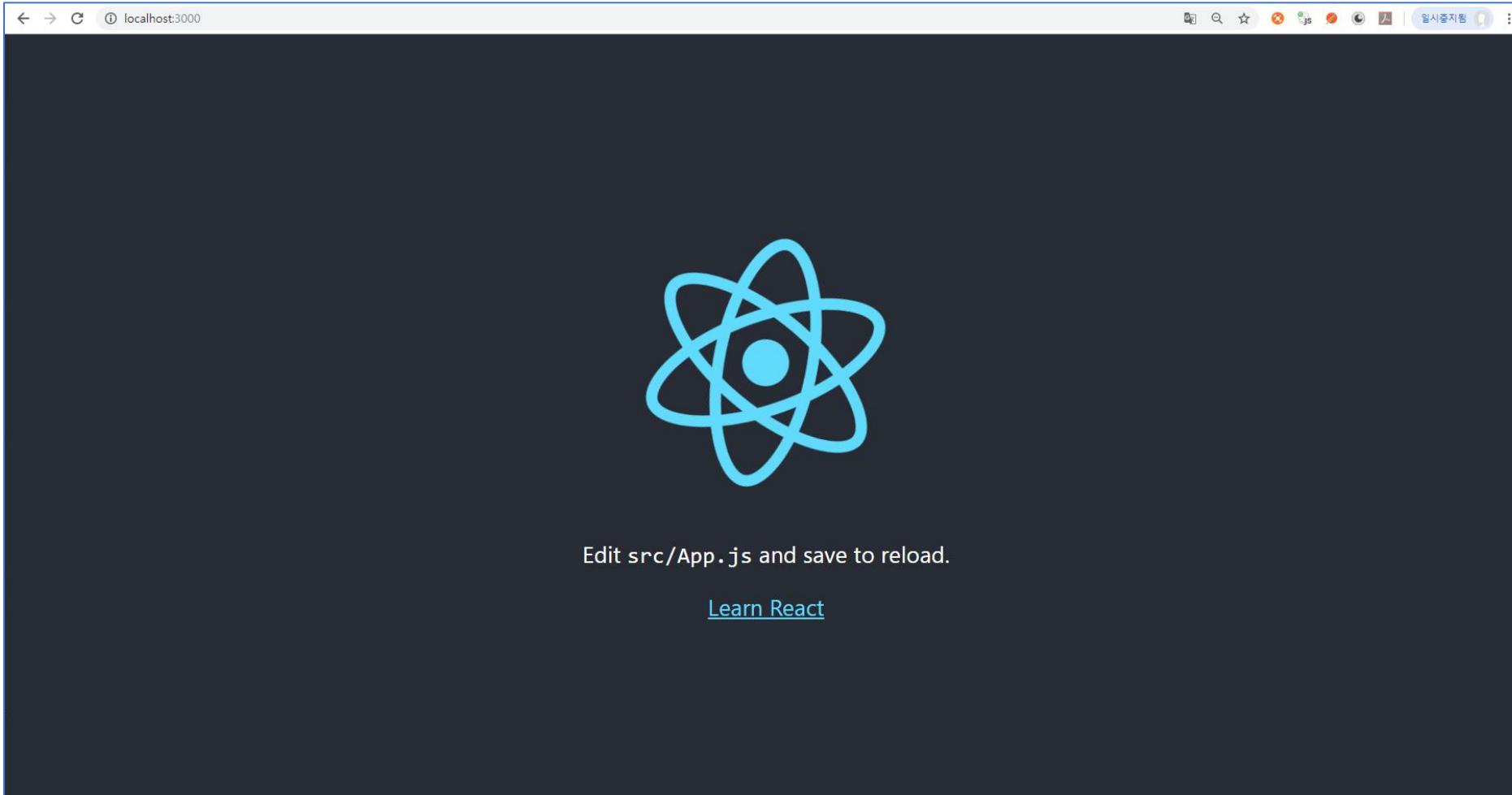
create-react-app 도구 설치하기

- **yarn** 이용시
- **yarn global add create-react-app** <= **global** 은 전역에서 쓰겠다는 의미임
- **npm install -g create-react-app** <= **-g** 은 전역에서 쓰겠다는 의미임

프로젝트 생성 절차

- 적절한 폴더에서 프로젝트를 생성한다. 여기서는 `c:/react_workspace` 경로에서 작성한다
- `cd /`
- `mkdir react_workspace`
- `cd react_workspace`
- `npx create-react-app hello-react` 또는
- `create-react-app hello-react` 오류발생시
- `cd hello-react`
- `yarn start`
- <http://localhost:3000>

첫 어플 실행화면



JSX 소개

```
import React from 'react';
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link" href="https://reactjs.org" target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

App.js

```
import React from 'react';          //외부에 있는 라이브러리를 읽어온다 'react'라는 라이브러리를 React 이름으로
import logo from './logo.svg';      //상대경로, logo.svg 를 logo라는 이름으로
import './App.css';                //css 로드

function App() {
  return (
    <div className= " App " >          //랜더링할 태그들 태그는 반드시 root 태그가 하나 있어야 한다
      <header className= " App-header " >
        <h1>Hello React</h1>
      </header>
    </div>
  );
}

export default App; //외부에서 접근 가능하도록 노출(exports)를 해야 만 한다.
```

포트번호 변경

- package.json 파일 수정하기

```
▷ Debug
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
"scripts": {
  "start": "set port=4000 && react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
```

Virtual DOM

- HTML 둘은 HTML document의 구조와 같은 트리 구조로 이루어져 있다. 트리 구조는 탐색하기 쉬우므로 사용성이 좋다. 하지만 빠르지 않다
- 가상 둘은 HTML 둘의 추상화 개념이다. 이것은 가볍고, 브라우저 스펙의 구현체와는 분리되어있다. 사실, 둘은 이미 추상화 개념이기 때문에 가상 둘은 추상화를 또 추상화한 개념이다.
- 기존에는 리소스가 큰 DOM 객체를 지속적으로 업데이트 해주어야 했지만, 그 전 단계로 Virtual DOM이 버퍼 역할을 해줌으로써 어떤 부분이 바뀔 지 미리 계산, 수정 후에 해당 수정된 부분만 업데이트 해주기 위한 기술이다.
- 성능관리를 자동으로 해줌
- 서버에서 DOM 을 미리 랜더링

import와 export

- import 문은 다른 곳에 정의된 코드를 문서내로 불러오는 기능을 수행합니다.
- export 는 모듈을 외부로 내보내기 위해 사용합니다.
- export default
 - 일반적으로 해당 모듈엔 **하나의 개체(변수, 클래스, 함수 등)**만 있다는 의미로 받아들여진다.
 - 따라서 해당 모듈의 전체 개체를 export 한다는 의미를 갖는다.
 - 원하는 이름으로 import가 가능하다.(ex. import ThisIsSample from "경로")
- export
 - 복수의 개체가 있는 라이브러리 형태의 모듈에서 가져오기 할 때 사용된다.
 - 특정 개체만 가져오는게 가능하다.
 - 원하는 이름으로 import가 불가능하다.
 - 불러올때 중괄호가 필요하다

JSX란

JSX란

- JSX란 React에서 사용되는 JavaScript 확장 문법으로, HTML과 유사한 구문을 JavaScript 코드 내에서 사용할 수 있게 해줍니다. JSX는 React 컴포넌트에서 UI를 표현할 때 사용되며, JavaScript의 표현식으로 컴파일되어 실행됩니다
- HTML과 유사한 구문을 사용한다.
- JSX는 JavaScript 코드 안에 HTML 태그와 매우 유사한 구문을 포함할 수 있습니다. 이로 인해 사용자 인터페이스를 정의하는 것이 직관적이고 가독성이 높아집니다.
- 브라우저는 JSX를 이해하지 못하기 때문에 Babel과 같은 도구를 사용하여 JSX를 JavaScript로 변환한 후 실행됩니다. JSX는 내부적으로 `React.createElement()` 함수로 변환됩니다.
- JSX 안에서 JavaScript 표현식을 사용할 수 있습니다. 이를 통해 동적 데이터나 함수 호출 결과를 렌더링할 수 있습니다.

jsx1.html

- 중괄호를 통해서 JavaScript 표현식을 JSX에 삽입할 수 있습니다.

```
<div id="root"></div>
<script type="text/babel">
  let rootNode= document.getElementById("root");
  //중괄호를 통해서 JavaScript 표현식을 JSX에 삽입할 때 사용됩니다.

  //변수 선언
  let name = "홍길동";
  let age= 23;

  ReactDOM.createRoot(rootNode).render(
    <React.StrictMode>
      <div>
        <h1> 이름 : {name} </h1>
        <h1> 나이 : {age} </h1>
      </div>
    </React.StrictMode>,
  );
</script>
```

JSX 2.html

- css :jsx에서는

```
<body>
  <div id="root"></div>
  <script type="text/babel">
    let rootNode= document.getElementById("root");

    let mystyle={color:"blue", fontSize:"23pt"};
    //json 형태임, font-size 형태인 snake 표기법은 사용 못함, fontSize 형태의 카멜 표기법을 사용해야 한다
    //변수 선언
    ReactDOM.createRoot(rootNode).render(
      <React.StrictMode>
        <h1 style={mystyle}>스타일 적용하기</h1>
      </React.StrictMode>,
    );
  </script>
</body>
```

javascript 형태로 객체를 만들어서 react에게 전달해야 한다.

jsx3.html

- 함수도 {} 를 통해 react에 전달해야 한다

```
<body>
  <div id="root"></div>
  <script type="text/babel">
    let rootNode= document.getElementById("root");

    const myFunction = ()=>{
      alert("함수호출");
    }

    ReactDOM.createRoot(rootNode).render(
      <React.StrictMode>
        <h1 onClick={myFunction}>함수호출</h1>
      </React.StrictMode>,
    );
  </script>

</body>
```

함수도 {} 를 통해 전달해야 한다. 함수를 {} 없이하면 렌더링 될때 함수가 호출된다.

App1.js

```
//클래스 컴포넌트 - 이전 버전
class App1 extends React.Component {
  render(){
    return (
      <div className="App">
        <h1>클래스를 이용하여 컴포넌트 만들기</h1>
        <h2>첫번째 태그가 트리구조 하나만 있어야 한다</h2>
      </div>
    );
  }
}

export default App1;
```

클래스컴포넌트는 React.Component클래스를 상속받아야 하고 render()함수를 반드시 오버라이딩 해야 한다

App2.js

```
//클래스 컴포넌트 - 이전 버전
class App2 extends React.Component {

    sayHey(){    //내부 함수 선언
        alert("hey");
    }

    render(){
        //상수
        const Data={
            headTitle:"Welcome react",
            contentTitle:"react.js is beautiful",
            contentBody:"It's front end librry"
        };

        return (
            <div className="App">
                <h3 style={{backgroundColor:"darkRed",padding:"10px 10px 10px 10px", color:"white"}}>{Data.headTitle}</h3>
                <h3>{Data.contentTitle}</h3>
                <h3>{Data.contentBody}</h3>
            </div>
        );
    }
}

export default App2;
```

index.js

```
ReactDOM.render(  
  <React.StrictMode>  
    <App/>  
    <App1/>  
    <App2 />  
    <App3 name="홍길동" age="20" phone="010-0000-0000"/>  
  </React.StrictMode>,  
  document.getElementById('root')  
);  
  
serviceWorker.unregister();
```

app3.js

```
//클래스 컴포넌트 - 이전 버전
class App3 extends React.Component {

  render(){

    return (
      <div className="App">
        <h3>{this.props.name}</h3>
        <h3>{this.props.age}</h3>
        <h3>{this.props.phone}</h3>
      </div>
    );
  }
}

export default App3;
```

event_demo(App1.js)

```
import React from 'react';
import logo from './logo.svg';
import './App.css';

class App1 extends React.Component {
    //생성자 - react에서 생성자를 만들때 주의할점
    //생성자의 매개변수로 반드시 props가 와야 한다
    //부모생성자를 호출해야 한다 props를 매개변수로 전달한다
    constructor(props){
        super(props);

        //state객체에 count 변수하나를 저장한다
        this.state ={count:0, name:""};

        //함수를 만들고 , bind라는 과정을 거쳐줘야 setState 함수 호출 가능
        this.increaseCount = this.increaseCount.bind(this);
    }

    //카운트값을 증가시킨다
    increaseCount(){
        //일반함수의 경우 bind를 하지 않으면, 클래스 내부의 함수가 아니라서 this를 사용못한다
        //그래서 생성자에서 반드시 bind 작업을 해줘야 한다
        //원래있던값을 읽어와서 1 더한다음 state객체의 값을 바꾼다
        this.setState( {count: this.state.count+1} )
    }

    //lambda 를 이용할 함수를 사용할 경우
    decreaseCount = ()=>{
        this.setState( {count: this.state.count-1} )
    }
}
```

event_demo(App1.js)

```
render(){

  return (
    <div className="App">

      <h3>현재 카운트 : {this.state.count} </h3>
      <button onClick={this.increaseCount}>증가하기</button>
      <button onClick={this.decreaseCount}>감소하기</button>
    </div>
  );
}

export default App1;
```

함수컴포넌트 app2.js

```
import React, {useState} from 'react';
import logo from './logo.svg';
import './App.css';

function App2()
{
    const [count, setCount] = useState(0);

    //카운트값을 증가시킨다
    const increaseCount=()=>{  
        setCount( count+1 );  
    }

    //lambda 를 이용할 함수를 사용할 경우
    const decreaseCount = ()=>{  
        setCount( count-1 );  
    }

    return (  
        <div className="App">  

            <h3>현재 카운트 : {count} </h3>
            <button onClick={increaseCount}>증가하기</button>
            <button onClick={decreaseCount}>감소하기</button>
        </div>
    );
}

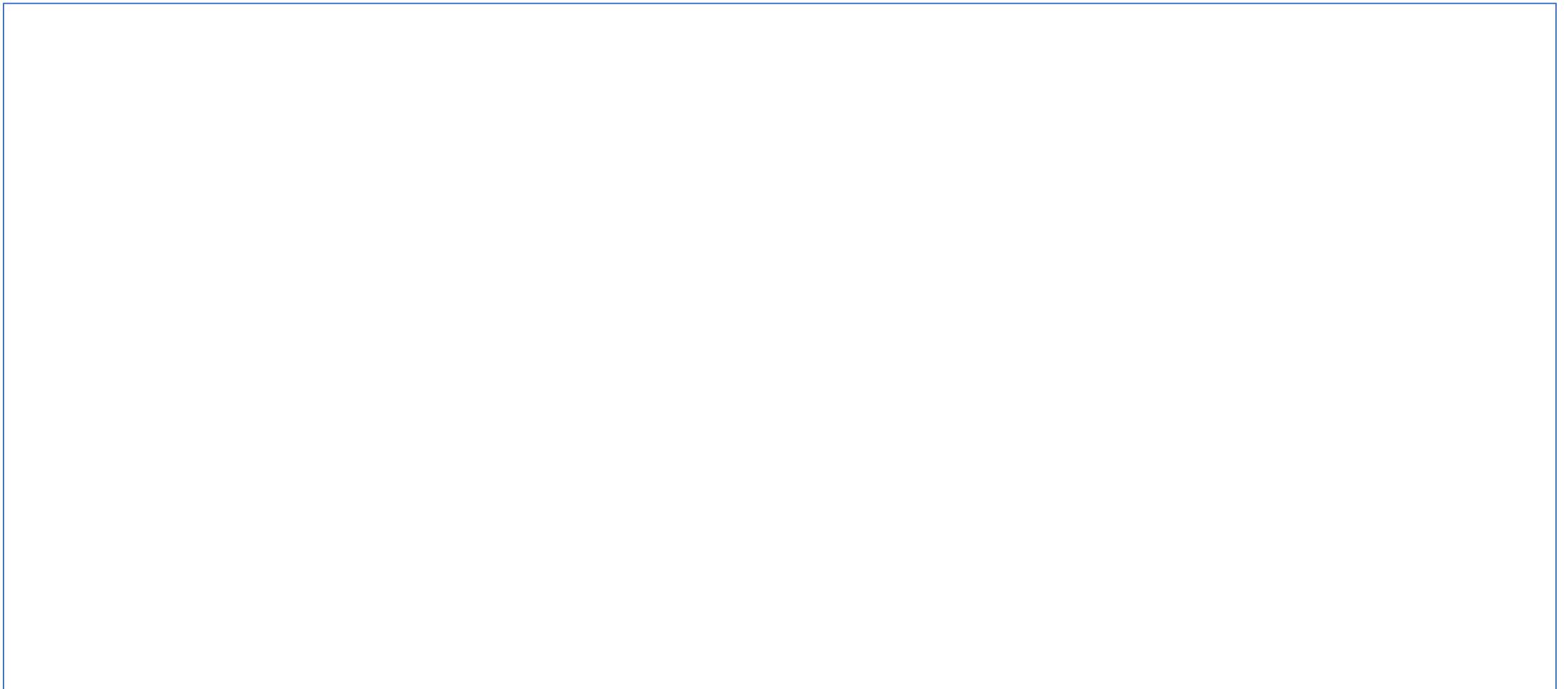
export default App2;
```

함수컴포넌트 app3.js

```
function App3()
{
  const [fruitList] = useState(["딸기", "사과", "배", "포도"]);

  return (
    <div className="App">
      <h3>과일 목록</h3>
      <table>
        {
          fruitList.map( (item)=>{
            return (
              <tr>
                <td>{item}</td>
              </tr>
            );
          })
        }
      </table>
    </div>
  );
}

export default App3;
```



JSX 사용하기(App1.js)

```
import React from 'react';          //모듈을 프로그램내로 loading한다

class App extends React.Component { //React.Component 상속, class개념 추가

  render(){

    return (
      <h1>Hello Veloport</h1>
      //반드시 tree구조 root 태그가 있어야 한다 , 따옴표를 쓰지 않는다

    );
  }

}

export default App; //모듈을 외부로 export 시켜야 한다
```

JavaScript Expression

- JSX 안에서, JavaScript 표현을 사용하려면 { } 로 둘러싸야 한다

```
render(){  
  
    let text = "홍길동"; //변수 선언, render 안에만 선언가능  
  
    return (  
  
        <div>  
  
            <h1> Hello Velopert </h1>  
  
            <h2> Welcome to {text}</h2>  
  
        </div>  
  
    );  
  
}
```

jsx 함수사용

```
sayHey(){ //함수 선언
    alert("hey");
}

render(){
    let text = "Dev-Server"
    return (
        <div>
            <h1> Hello Velopert </h1>
            <h2> Welcome to {text}</h2>
            <button onClick={this.sayHey}>Click Me</button>
        </div>
    );
}
```

jsx 문법

- If-Else 문 사용 불가 condition ? true : false 로 대체
- <p>{1 == 1 ? 'True' : 'False'}</p>

```
render() {
    let text = "Dev-Server";

    let pStyle = {
        color: 'aqua',
        backgroundColor: 'black'
    };

    return (
        <div>
            <h1> Hello Velopert </h1>
            <h2> Welcome to {text}</h2>
            <button onClick= {this.sayHey}>Click Me</button>
            <p style = {pStyle}>{1 == 1 ? 'True' : 'False'}</p>
        </div>
    );
}
```

jsx 에서 css적용

- css 는 camelCase를 사용한다 background-color -> backgroundColor 을 사용한다
- 데이터는 json 형태로 전달한다

```
import React from 'react';
import ReactDOM from 'react-dom';

class MyHeader extends React.Component {
  render() {
    return (
      <div>
        <h1 style={{backgroundColor: "lightblue"}}>Hello Style!</h1>
        <p>Add a little style!.</p>
      </div>
    );
  }
}
```

jsx 에서 css적용

```
class MyHeader extends React.Component {
  render() {
    const mystyle = {
      color: "white",
      backgroundColor: "DodgerBlue",
      padding: "10px",
      fontFamily: "Arial"
    };
    return (
      <div>
        <h1 style={mystyle}>Hello Style!</h1>
        <p>Add a little style!</p>
      </div>
    );
  }
}
```

jsx 에서 css 적용

app.css

```
body {  
    background-color: #282c34;  
    color: white;  
    padding: 40px;  
    font-family: Arial;  
    text-align: center;  
}
```

```
import React from 'react';  
import ReactDOM from 'react-dom';  
import './App.css';  
  
class MyHeader extends React.Component {  
    render() {  
        return (  
            <div>  
                <h1>Hello Style!</h1>  
                <p>Add a little style!.</p>  
            </div>  
        );  
    }  
}
```

이벤트핸들러

```
import React from 'react';
import ReactDOM from 'react-dom';

function shoot() {
  alert("Great Shot!");
}

const myelement = (
  <button onClick={shoot}>Take the shot!</button>
);
```

함수는 {}로 부른다

이벤트핸들러

```
import React from 'react';
import ReactDOM from 'react-dom';

class Football extends React.Component {
  shoot() {
    alert("Great Shot!");
  }
  render() {
    return (
      <button onClick={this.shoot}>Take the shot!</button>
    );
  }
}
```

함수는 {}로 부른다

클래스 내부에 선언되는 것이 좋다

이벤트핸들러(binding)

- 클래스 구성 요소에서 this키워드는 기본적으로 정의되어 있지 않으므로 일반 함수를 사용하면 this키워드는 메서드를 호출 한개체를 나타내며 전역 창 개체, HTML 단추 등이 될 수 있습니다.
- 화살표 함수 대신 일반 함수를 사용해야 하는 경우에는 반드시 바인딩을 해야 합니다

```
import React from 'react';
import ReactDOM from 'react-dom';

class Football extends React.Component {
  constructor(props) {
    super(props)
    this.shoot = this.shoot.bind(this)
  }
  shoot() {
    alert(this);
  }
  render() {
    return (
      <button onClick={this.shoot}>Take the shot!</button>
    );
  }
}
```

<https://reactjs.org/docs/legacy-context.html>

이벤트핸들러

```
class Football extends React.Component {  
  shoot = (a, b) => {  
    alert(b.type);  
    /*  
     * 'b' represents the React event that triggered the function,  
     * in this case the 'click' event  
     */  
  }  
  render() {  
    return (  
      <button onClick={(ev) => this.shoot("Goal", ev)}>Take the shot!</button>  
    );  
  }  
}  
  
ReactDOM.render(<Football />, document.getElementById('root'));
```

이벤트 발생 객체를 직접 파라미터로 전달 할 수 있다

이벤트핸들러

```
class Football extends React.Component {  
  shoot = (a, b) => {  
    alert(b.type);  
    /*  
     * 'b' represents the React event that triggered the function,  
     * in this case the 'click' event  
     */  
  }  
  render() {  
    return (  
      <button onClick={this.shoot.bind(this, "Goal")}>Take the shot!</button>  
    );  
  }  
}
```

bind를 사용할때는 반드시 마지막에 객체가 전달된다

React Form

```
class MyForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = { username: '' };
  }
  myChangeHandler = (event) => {
    this.setState({username: event.target.value});
  }
  render() {
    return (
      <form>
        <h1>Hello {this.state.username}</h1>
        <p>Enter your name:</p>
        <input
          type='text'
          onChange={this.myChangeHandler}
        />
      </form>
    );
  }
}
```

이벤트 핸들러는 callback 함수 형태로 전달 기본값을 주려면 value 속성을 주면된다.

틱택토 게임만들기

- 기본소스
 - board.css
 - game.js
- 소스 설명
 - **Square - Square** 컴포넌트는 <button>을 렌더링합니다
 - **Board - Board**는 사각형 9개를 렌더링합니다
 - **Game - Game** 컴포넌트는 게임판을 렌더링합니다

콤포넌트간 값 전달

```
//Square 컴포넌트 - 버튼을 하나의 컴포넌트로 만들었음
class Square extends React.Component {
    render() {
        return (
            <button className="square">
                {this.props.value}
            </button>
        );
    }
}

//게임판
class Board extends React.Component {
    //함수
    renderSquare(i) {
        //i - 매개변수, 이 값을 Square객체에 전달한다
        return <Square value={i}/>; //Square객체 반환
    }
}
```

사용자와 상호작용하는 컴포넌트 만들기

```
//Square 컴포넌트 - 버튼을 하나의 컴포넌트로 만들었음
class Square extends React.Component {
  render() {
    return (
      //이벤트 핸들러 , onClick 이벤트
      /*
        <button className="square" onClick={function() { alert('click'); }}>
          {this.props.value}
        </button>
      */
      //람다식으로 만들자
      <button className="square" onClick={() => alert('click')}>
        {this.props.value}
      </button>
    );
  }
}
```

state(값을 기억하자)

```
class Square extends React.Component {
  //생성자
  constructor(props) { //props
    super(props); //부모 생성자 호출
    //state추가
    this.state = {
      value: null,
    };
  }

  render() {
    return (
      //이벤트 핸들러 , onClick 이벤트
      <button className="square" onClick={() => this.setState({value: 'X'})}>
        {this.state.value}
      </button>
    );
  }
}
```

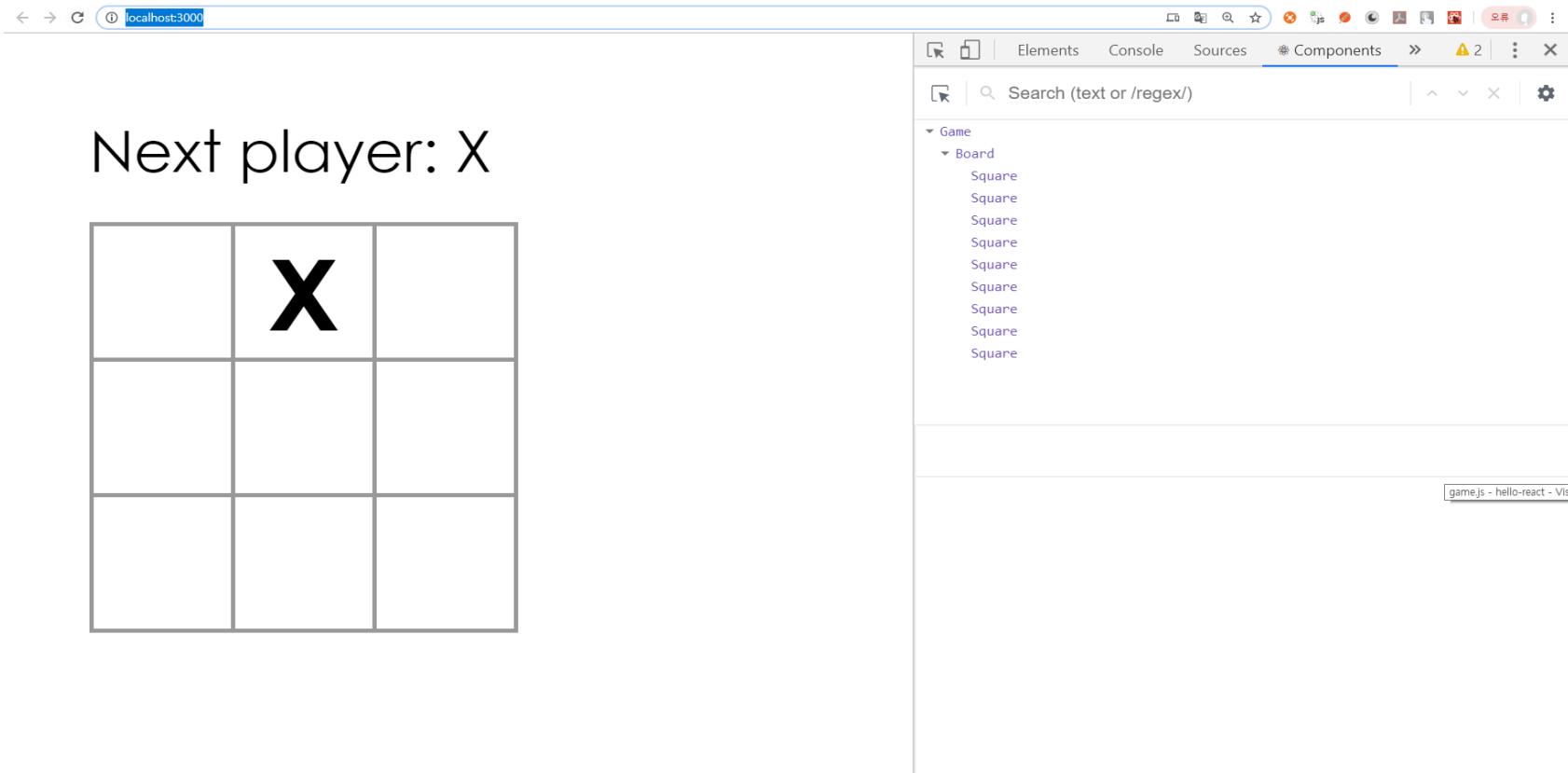
개발자 도구

- Chrome과 Firefox의 React Devtools 확장 프로그램을 사용하면 브라우저 개발자 도구에서 React 컴포넌트 트리를 검사할 수 있습니다
- 구글 웹스토어 검색
- <https://chrome.google.com/webstore/category/extensions?hl=ko>
- **React Devtools** 검색후 추가하기

```
<Game>
  <div className="game">
    <div className="game-board">
      <Board>
        <div>
          <div className="status">Next player: X</div>
          <div className="board-row">
            ><Square>...</Square>
            ><Square>...</Square>
            ><Square>...</Square>
          </div>
          <div className="board-row">
            ><Square>...</Square>
            ><Square>...</Square>
            ><Square>...</Square>
            ><Square>...</Square>
          </div>
          <div className="board-row">
            ><Square>...</Square>
            ><Square>...</Square>
            ><Square>...</Square>
            ><Square>...</Square>
          </div>
        </div>
      </Board>
    </div>
    <div className="game-info">
      <div/>
      <ol/>
    </div>
  </div>
</Game>
```

개발자도구

- F12 누르고 component 탭 선택



State 끌어올리기

- 현재 게임의 state를 각각의 Square 컴포넌트에서 유지하고 있습니다
- 각 Square가 아닌 부모 Board 컴포넌트에 게임의 상태를 저장하는 것이 가장 좋은 방법입니다.
- 각 Square에 숫자를 넘겨주었을 때와 같이 Board 컴포넌트는 각 Square에게 prop을 전달하는 것으로 무엇을 표시할 지 알려줍니다.
- 여러개의 자식으로부터 데이터를 모으거나 두 개의 자식 컴포넌트들이 서로 통신하게 하려면 부모 컴포넌트에 공유 state를 정의해야 합니다. 부모 컴포넌트는 props를 사용하여 자식 컴포넌트에 state를 다시 전달할 수 있습니다. 이것은 자식 컴포넌트들이 서로 또는 부모 컴포넌트와 동기화 하도록 만듭니다.

Board 클래스

```
class Board extends React.Component {
  constructor(props: any): Board 저장한다
  constructor(props) {
    super(props);
    this.state = {
      squares: Array(9).fill(null),
    };
  }

  //호출될 함수 추가하기
  handleClick(i) {
    const squares = this.state.squares.slice();
    squares[i] = 'X';
    this.setState({squares: squares});
  }

  renderSquare(i) {
    return (
      <Square
        value={this.state.squares[i]}
        onClick={() => this.handleClick(i)}
      />
    );
  }
}
```

불변성

- React에서는 객체 자체의 내용을 변경하기보다는 새로운 객체를 만들어서 복사된 객체를 이용해 값을 수정한다
- 특징
 - 복잡한 특징들을 단순하게 만듦
 - 변화를 감지함
 - 객체가 직접적으로 수정되기 때문에 복제가 가능한 객체에서 변화를 감지하는 것은 어렵습니다. 감지는 복제가 가능한 객체를 이전 사본과 비교하고 전체 객체 트리를 돌아야 합니다.
 - 불변 객체에서 변화를 감지하는 것은 상당히 쉽습니다. 참조하고 있는 불변 객체가 이전 객체와 다르다면 객체는 변한 것입니다.
 - React에서 다시 렌더링하는 시기를 결정함
 - 불변성의 가장 큰 장점은 React에서 순수 컴포넌트를 만드는데 도움을 준다는 것입니다. 변하지 않는 데이터는 변경이 이루어졌는지 쉽게 판단할 수 있으며 이를 바탕으로 컴포넌트가 다시 렌더링할지를 결정할 수 있습니다.

함수 컴포넌트

- React에서 함수 컴포넌트는 더 간단하게 컴포넌트를 작성하는 방법이며 state 없이 render 함수만을 가집니다. React.Component를 확장하는 클래스를 정의하는 대신 props를 입력받아서 렌더링할 대상을 반환하는 함수를 작성할 수 있습니다. 함수 컴포넌트는 클래스로 작성하는 것보다 빠르게 작성할 수 있으며 많은 컴포넌트를 함수 컴포넌트로 표현할 수 있습니다.

함수 컴포넌트

```
//Square 컴포넌트 - 버튼을 하나의 컴포넌트로 만들었음
function Square(props) {
  return (
    <button className="square" onClick={props.onClick}>
      {props.value}
    </button>
  );
}
```

순서 만들기(ox)교차로

```
constructor(props) {
  super(props);
  this.state = {
    squares: Array(9).fill(null),
    xIsNext: true,
  };
}

//호출될 함수 추가하기
handleClick(i) {
  //배열을 자른다
  const squares = this.state.squares.slice();
  //squares[i] = 'X'; //해당 배열의 값을 바꿔치기한다
  //this.setState({squares: squares});
  squares[i] = this.state.xIsNext ? 'X' : 'O';
  this.setState({
    squares: squares,
    xIsNext: !this.state.xIsNext,
  });
}

const status = 'Next player: ' + (this.state.xIsNext ? 'X' : 'O');
```

승자 결정하기

```
function calculateWinner(squares) {
  const lines = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
    [0, 3, 6],
    [1, 4, 7],
    [2, 5, 8],
    [0, 4, 8],
    [2, 4, 6],
  ];
  for (let i = 0; i < lines.length; i++) {
    const [a, b, c] = lines[i];
    if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c]) {
      return squares[a];
    }
  }
  return null;
}
```

승자 결정하기

```
renderSquare(i) {  
  
  const winner = calculateWinner(this.state.squares);  
  let status;  
  if (winner) {  
    status = 'Winner: ' + winner;  
  } else {  
    status = 'Next player: ' + (this.state.xIsNext ? 'X' : 'O');  
  }  
  
  return (  
    <Square  
      value={this.state.squares[i]}  
      onClick={() => this.handleClick(i)}  
    />  
  );  
}
```

승자 결정하기

```
//호출될 함수 추가하기
handleClick(i) {
  //배열을 자른다
  const squares = this.state.squares.slice();
  if (calculateWinner(squares) || squares[i]) {
    alert("이미 놓았음");
    return;
  }
  //const squares = this.state.squares.slice();

  squares[i] = this.state.xIsNext ? 'X' : 'O';
  this.setState({
    squares: squares,
    xIsNext: !this.state.xIsNext,
  });
}
```

시간 여행 추가하기

```
history = [
    // 첫 동작이 발생하기 전
    {
        squares: [
            null, null, null,
            null, null, null,
            null, null, null,
        ]
    },
    // 첫 동작이 발생한 이후
    {
        squares: [
            null, null, null,
            null, 'X', null,
            null, null, null,
        ]
    },
    // 두 번째 동작이 발생한 이후
    {
        squares: [
            null, null, null,
            null, 'X', null,
            null, null, 'O',
        ]
    },
    // ...
]
```

시간 여행 추가하기

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
import './Board.css';
function Square(props) {
  return (
    <button className="square" onClick={props.onClick}>
      {props.value}
    </button>
  );
}
```

시간 여행 추가하기

```
class Board extends React.Component {
  renderSquare(i) {
    return (
      <Square
        value={this.props.squares[i]}
        onClick={() => this.props.onClick(i)}
      />
    );
  }

  render() {
    return (
      <div>
        <div className="board-row">
          {this.renderSquare(0)}{this.renderSquare(1)}{this.renderSquare(2)}
        </div>
        <div className="board-row">
          {this.renderSquare(3)}{this.renderSquare(4)}{this.renderSquare(5)}
        </div>
        <div className="board-row">
          {this.renderSquare(6)}{this.renderSquare(7)}{this.renderSquare(8)}
        </div>
      </div>
    );
  }
}
```

시간 여행 추가하기

```
class Game extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      history: [
        {
          squares: Array(9).fill(null)
        }
      ],
      stepNumber: 0,
      xIsNext: true
    };
  }
}
```

시간 여행 추가하기

```
handleClick(i) {
  const history = this.state.history.slice(0, this.state.stepNumber + 1);
  const current = history[history.length - 1];
  const squares = current.squares.slice();
  if (calculateWinner(squares) || squares[i]) {
    return;
  }
  squares[i] = this.state.xIsNext ? "X" : "O";
  this.setState({
    history: history.concat([
      {
        squares: squares
      }
    ]),
    stepNumber: history.length,
    xIsNext: !this.state.xIsNext
  });
}
```

시간 여행 추가하기

```
jumpTo(step) {
    this.setState({
        stepNumber: step,
        xIsNext: (step % 2) === 0
    });
}
```

시간 여행 추가하기

```
render() {
  const history = this.state.history;
  const current = history[this.state.stepNumber];
  const winner = calculateWinner(current.squares);

  const moves = history.map((step, move) => {
    const desc = move ?
      'Go to move #' + move :
      'Go to game start';
    return (
      <li key={move}>
        <button onClick={() => this.jumpTo(move)}>{desc}</button>
      </li>
    );
  });

  let status;
  if (winner) {
    status = "Winner: " + winner;
  } else {
    status = "Next player: " + (this.state.xIsNext ? "X" : "O");
  }
}
```

시간 여행 추가하기

```
let status;
if (winner) {
  status = "Winner: " + winner;
} else {
  status = "Next player: " + (this.state.xIsNext ? "X" : "O");
}

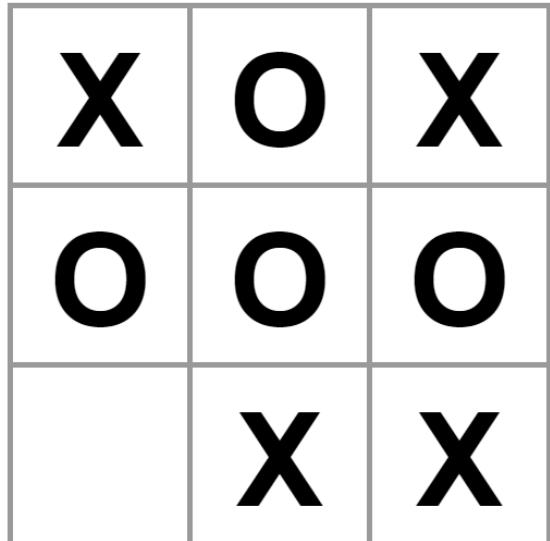
return (
  <div className="game">
    <div className="game-board">
      <Board
        squares={current.squares}
        onClick={i => this.handleClick(i)}
      />
    </div>
    <div className="game-info">
      <div>{status}</div>
      <ol>{moves}</ol>
    </div>
  );
}
```

시간 여행 추가하기

```
function calculateWinner(squares) {
  const lines = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
    [0, 3, 6],
    [1, 4, 7],
    [2, 5, 8],
    [0, 4, 8],
    [2, 4, 6]
  ];
  for (let i = 0; i < lines.length; i++) {
    const [a, b, c] = lines[i];
    if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c]) {
      return squares[a];
    }
  }
  return null;
}

export default Game
```

시간 여행 추가하기



Winner: O

1. Go to game start
2. Go to move #1
3. Go to move #2
4. Go to move #3
5. Go to move #4
6. Go to move #5

자동완성

- preferences - setting

함수형컴포넌트

함수형컴포넌트

Hook 기능이 React 16.8버전에 새로 추가되었습니다.

Hook은 클래스 컴포넌트를 작성하지 않아도 state와 같은 특징들을 사용할 수 있습니다.

- React 16.8 버전부터 함수형 컴포넌트에서 state와 라이프사이클을 사용할 수 있는 Hook이라는 개념이 등장했습니다. 때문에 기존에 동일시 되었던 Stateless Component와 Functional Component는 개념적으로 분리가 되었다고 볼 수 있습니다.
- **State hook**
- Hooks에는 state를 사용할 수 있는 useState함수가 추가되었습니다. 인자로 초기 state값을 넣어주면 state와 state를 업데이트 시켜주는 함수가 배열 형태로 반환됩니다.
- **Effect Hook**
- state hook외에도 비교적 간단한 라이프사이클을 사용하기 위해 useEffect라는 함수도 추가되었습니다. useEffect는 컴포넌트가 마운트 되었을 때(componentDidMount), 컴포넌트가 업데이트 되었을 때(componentDidUpdate), 컴포넌트가 마운트 해제될때(componentWillUnmount) 세가지 경우의 라이프 사이클을 사용할 수 있습니다

함수형 콤포넌트

함수형 콤포넌트

함수형 콤포넌트

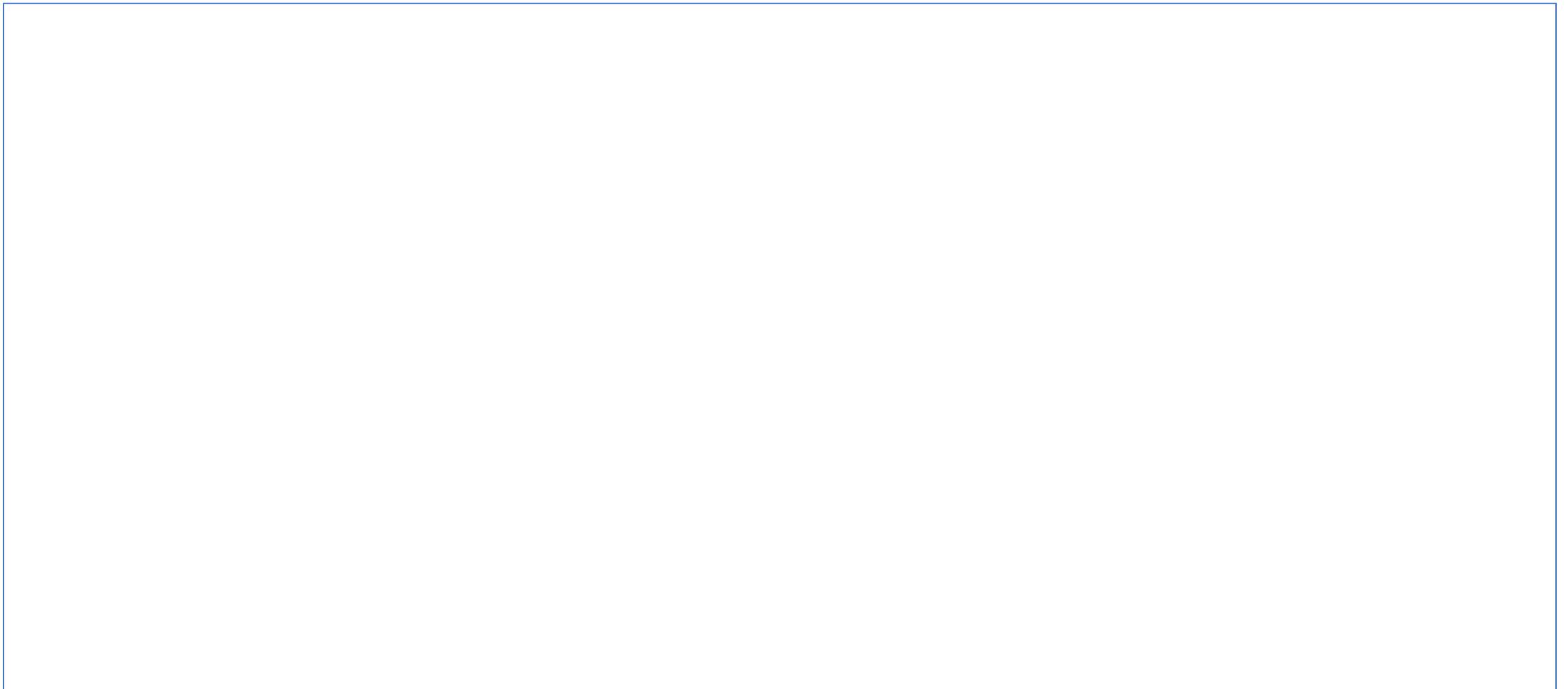
```
import React, { useState } from "react";

function About() {
  //컴포넌트가 저장할 데이터를 만들어 보자
  const [message, setMessage] = useState("어서오십쇼!");

  //이벤트 핸들러 만들기
  const onClickEnter = () => setMessage("주문 어떻게 해드릴까요?");

  return (
    <div>
      <h1> This is About {message} </h1>
      <button type="button" onClick={onClickEnter}>주문</button>
    </div>
  );
}

export default About;
```



라우터

라우터

- `cd /react_workspace`
- `npm install -g create-react-app`
- `create-react-app myapp-router`
- `cd myapp-router`
- `npm install react-router-dom`
- `yarn start`
- 포트번호 수정(`package.json`수정)

`"start": "set PORT=3000 & react-scripts start",`

/ : Home 컴포넌트를 보여줍니다.
/about : About 컴포넌트를 보여줍니다.
/posts : Post 컴포넌트를 보여줍니다.
/login :Login 컴포넌트를 보여줍니다.

`"start": "set PORT=3000 & react-scripts start",`

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { BrowserRouter } from "react-router-dom";
ReactDOM.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  document.getElementById('root')
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

/components/header.js

```
//header.js

import React, { Component } from 'react';
import { Link } from 'react-router-dom';
import './header.css';

class Header extends Component {
  render() {
    return (
      <div className="header">
        <Link className="item" to="/">홈</Link>
        <Link className="item" to="/about/TeamProject">소개</Link>
        <Link className="item" to="/Posts">포스트</Link>
        <Link className="item" to="/login">로그인</Link>

      </div>
    );
  }
}

export default Header;
```

/router/about.js

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

class About extends Component {
  render() {
    return (
      <div>
        {this.props.match.params.username} 의 소개
      </div>
    );
  }
}

export default About;
```

/router/about.js

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

class About extends Component {
  render() {
    return (
      <div>
        {this.props.match.params.username} 의 소개
      </div>
    );
  }
}

export default About;
```

/router/home.js

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

class Home extends Component {
  render() {
    return (
      <div>
        豪
      </div>
    );
  }
}

export default Home;
```

/router/about.js

```
import React, { Component } from 'react';
import { Redirect } from 'react-router-dom';

class Login extends Component {
  isLoggedIn = false;

  render() {
    return (
      <div>
        {
          !this.isLoggedIn && <Redirect to="/" />
        }
        로그인 되었습니다.
      </div>
    );
  }
}

export default Login;
```

/router/posts.js

```
import React, { Component } from 'react';
import { Route, Link } from 'react-router-dom';

class Post extends Component {
  render() {
    return (
      <div>
        <p>title: {this.props.match.params.title}</p>
        <p>id: {new URLSearchParams(this.props.location.search).get('id')}</p>
      </div>
    );
  }
}

class Posts extends Component {
  render() {
    return (
      <div>
        <p>포스트</p>
        <Link to="/posts/first?id=1">First</Link>
        <Link to="/posts/second?id=2">Second</Link>
        <Link to="/posts/third?id=3">Third</Link>
        <Route path="/posts/:title" component={Post} />
      </div>
    );
  }
}

export default Posts;
```

/router/nomatch.js

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

class NoMatch extends Component {
  render() {
    return (
      <div>
        404
      </div>
    );
  }
}

export default NoMatch;
```

app.js

```
import React, { Component } from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
//외부 js모듈들 불러오기
import Header from './components/header';
import Home from './routes/home';
import About from './routes/about';
import Posts from './routes/posts';
import Login from './routes/login';
import NoMatch from './routes/nomatch';
class App extends Component {
  render() {
    return (
      <Router>
        <div>
          <Header />
          <Switch>
            <Route exact path="/" component={Home}/>
            <Route path="/about/:username" component={About}/>
            <Route path="/posts" component={Posts}/>
            <Route path="/login" component={Login}/>
            <Route component={NoMatch}/>
          </Switch>
        </div>
      </Router>
    );
  }
}

export default App;
```

라우터 새버전(App.js)

```
import logo from './logo.svg';
import './App.css';
import * as React from "react";
import { Routes, Route, Outlet, Link } from "react-router-dom";

import About from './component/about';
import Home from './component/home';
import Dashboard from './component/dashboard';

function App() {
  return (
    <div>
      <h1>Basic Example</h1>

      /* Routes nest inside one another. Nested route paths build upon
         parent route paths, and nested route elements render inside
         parent route elements. See the note about <Outlet> below. */
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="about" element={<About />} />
          <Route path="dashboard" element={<Dashboard />} />

          /* Using path="***" means "match anything", so this route
             acts like a catch-all for URLs that we don't have explicit
             routes for. */
          <Route path="*" element={<NoMatch />} />
        </Route>
      </Routes>
    </div>
  );
}


```

라우터 새버전(App.js)

```
function Layout() {
  return (
    <div>
      {/* A "layout route" is a good place to put markup you want to
         share across all the pages on your site, like navigation. */}
      <nav>
        <ul>
          <li><Link to="/">Home</Link></li>
          <li>
            <Link to="/about">About</Link>
          </li>
          <li>
            <Link to="/dashboard">Dashboard</Link>
          </li>
          <li>
            <Link to="/nothing-here">Nothing Here</Link>
          </li>
        </ul>
      </nav>

      <hr />

      {/* An <Outlet> renders whatever child route is currently active,
         so you can think about this <Outlet> as a placeholder for
         the child routes we defined above. */}
      <Outlet />
    </div>
  );
}
```

라우터 새버전(App.js)

```
function NoMatch() {
  return (
    <div>
      <h2>Nothing to see here!</h2>
      <p>
        <Link to="/">Go to the home page</Link>
      </p>
    </div>
  );
}

export default App;
```

./component/app.js 예시

```
function About() {
  return (
    <div >
      This is About
    </div>
  );
}

export default About;
```

props 와 state

props

- props 는 컴포넌트에서 사용 할 데이터 중 변동되지 않는 데이터를 다룰 때 사용됩니다.
- parent 컴포넌트에서 child 컴포넌트로 데이터를 전할 때, props 가 사용됩니다.
- parent에서 전달된 값은 무조건 this.props로 전달 받을 수 있음

./prop_test/header.js

```
import React from 'react';
//부모쪽에서 title 변수에 값을 전달해야 함
class Header extends React.Component {
  render(){
    return (
      <h1>{ this.props.title }</h1>
    );
  }
}

export default Header;
```

./prop_test/content.js

```
import React from 'react';

class Content extends React.Component {
  render(){
    return (
      <div>
        <h2>{ this.props.title }</h2>
        <p> { this.props.body } </p>
      </div>
    );
  }
}

export default Content;
```

app.js수정

```
import React, { Component } from 'react';
import Header from './props_test/header'
import Content from './props_test/content'

class App extends Component {
    render() {
        const Data =
        {
            headerTitle: "Welcome react",
            contentTitle:"react.js is beautiful",
            contentBody: "It' front end librry"
        }
    }
}
```

app.js수정

```
return (
  <div className="App">
    <header className="App-header">
      <h1>Hello React</h1>
    </header>
    <h3>부모로부터 받은 데이터 전달하기 </h3>
    <Header title={ this.props.headerTitle }/>
    <Content title={ this.props.contentTitle }
             body={ this.props.contentBody }>

    <br/><br/>
    <h3>내가 갖고 있는 데이터 전달하기 </h3>
    <Header title={ Data.headerTitle }/>
    <Content title={ Data.contentTitle }
             body={ Data.contentBody }>

  </div>
);
```

index.js

```
ReactDOM.render(<App
  headerTitle = "Welcome!"
  contentTitle = "Stranger,"
  contentBody = "Welcome to example app"/>, document.getElementById('root'));
```

타입체크코드추가(content.js)

```
import React from 'react';
import PropTypes from 'prop-types'

class Content extends React.Component {
  render(){
    return (
      <div>
        <h2>{ this.props.title }</h2>
        <p> { this.props.body } </p>
      </div>
    );
  }
}

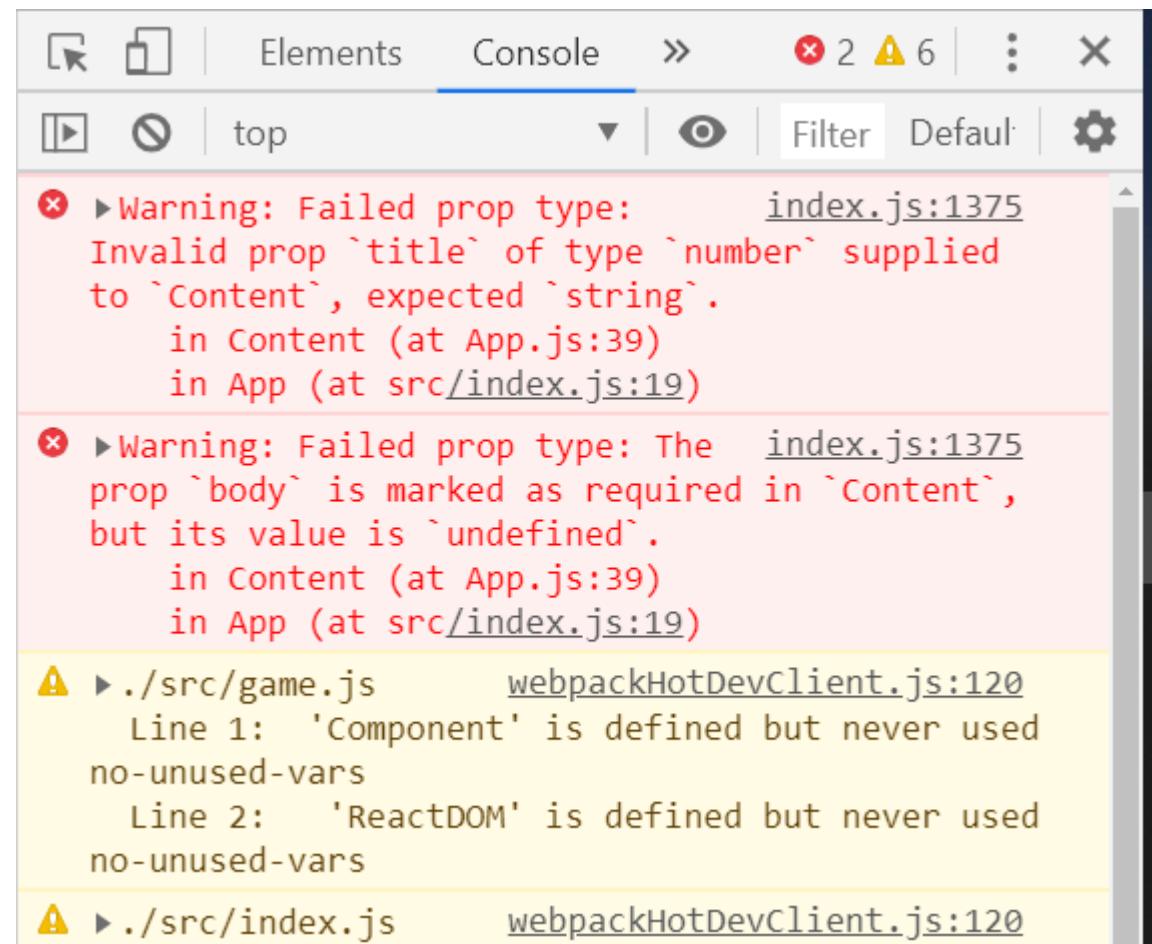
//타입 검증하기
Content.propTypes = {
  title: PropTypes.string,
  body: PropTypes.string.isRequired
};
export default Content;
```

app.js 설정

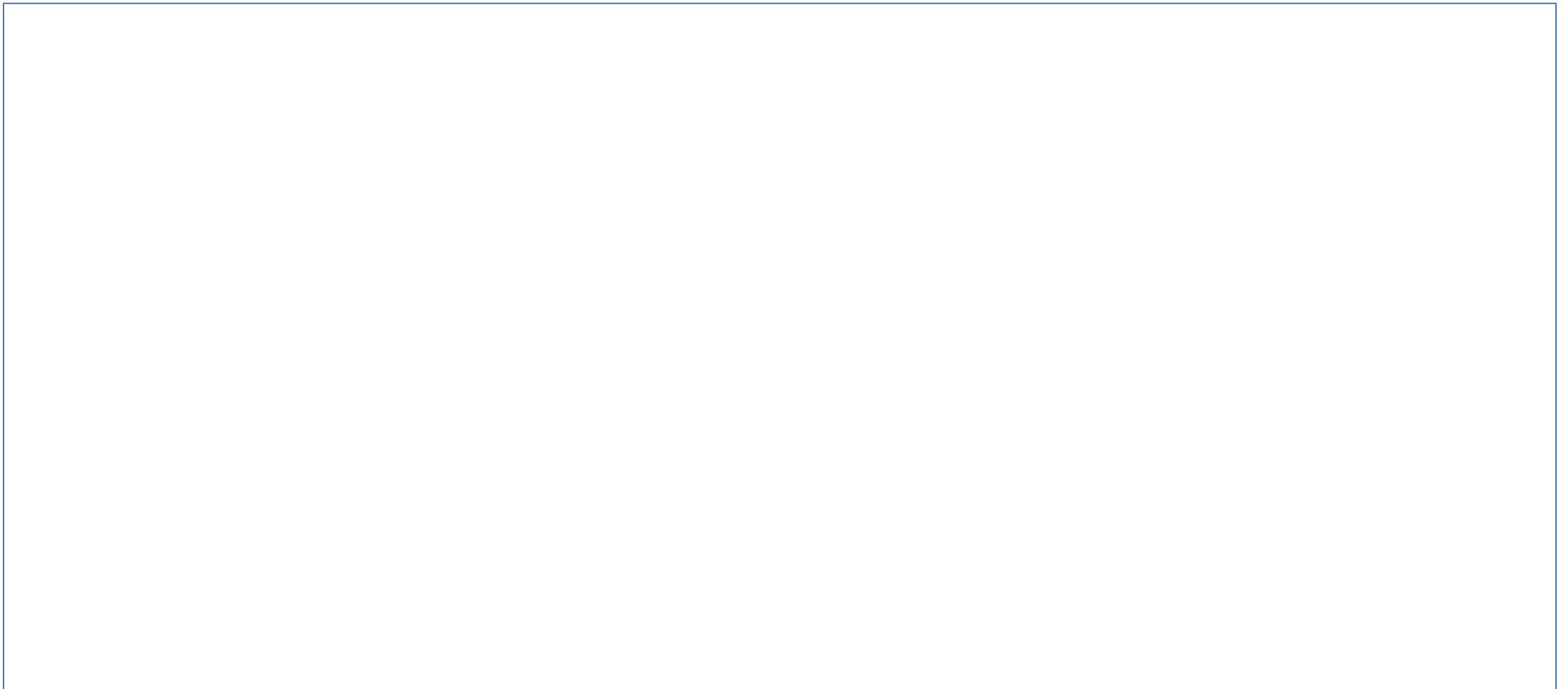
```
App.defaultProps = {
  headerTitle: 'Default header',
  contentTitle: 5,
  contentBody: undefined
};

export default App;
```

결과는 출력되나 경고가 나온다







데이터베이스 연동하기

MySQL

MySql 설치하기

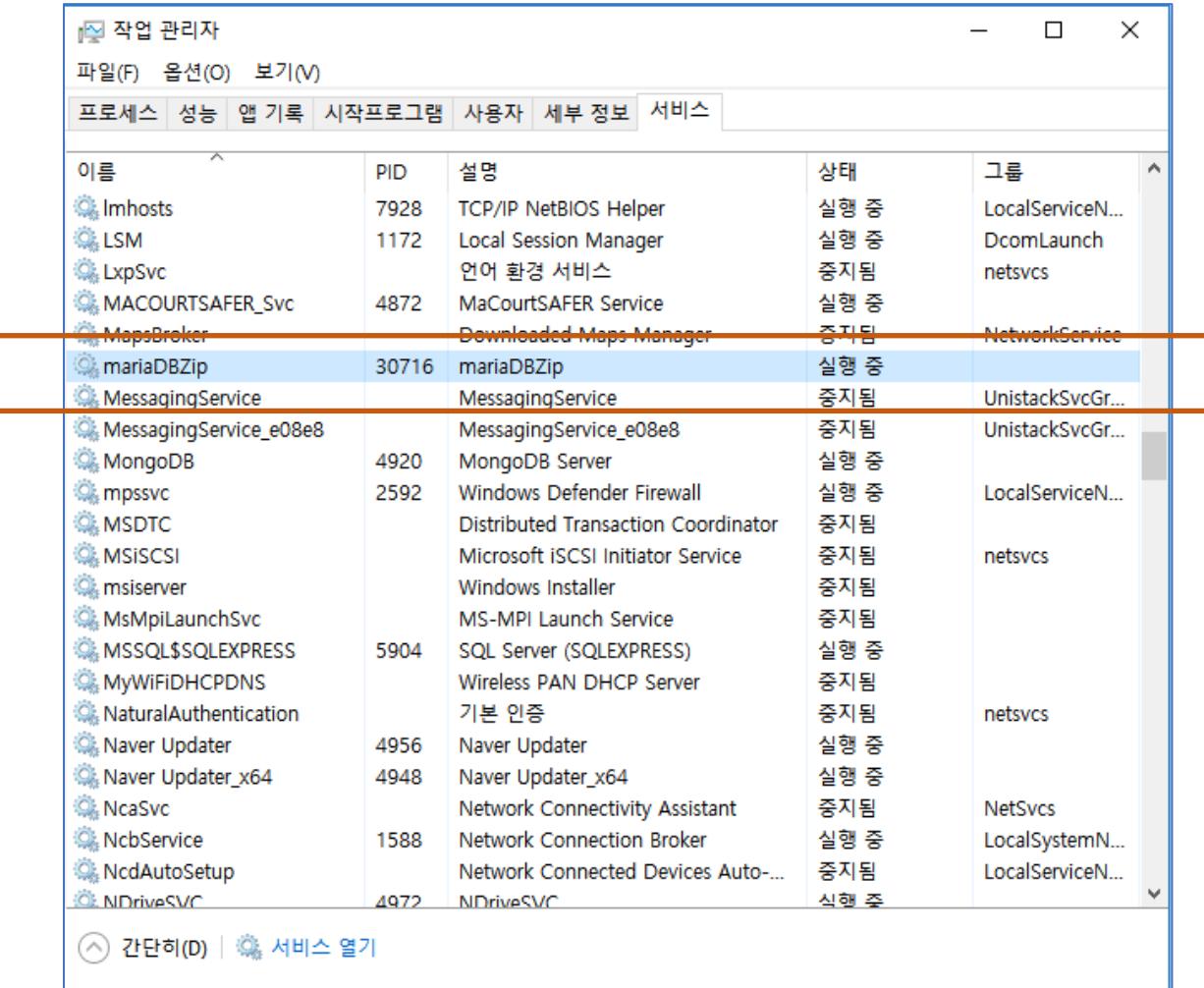
- <https://downloads.mariadb.com/MariaDB/mariadb-10.0/winx64-packages/>



MySql 설치

- mariadb 실행방법
- c:\maridb에 압축을 풀다
- cmd 창을 관리자권한으로 실행시킨다.
- cd /mariadb/bin
- mysql_install_db --datadir=C:\mariadb\data --service=mariaDBZip --port=5306 --password=1234

서비스 시작하기



오른쪽 버튼
눌러서 가동
을 눌러주자

MySQL과 연동하기

- mysql -u root -p --port=5306
- 패스워드 : 1234

```
C:\mariadb\bin>mysql -u root -p --port=5306
Enter password: ****
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 9
Server version: 10.3.14-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
4 rows in set (0.003 sec)

MariaDB [(none)]>
```

디비 시작하기

- 디비연결하기

```
mysql -u root -p --port=5306
```

```
Enter your password :1234
```

- 디비 생성

```
CREATE DATABASE mydb default CHARACTER SET UTF8;
```

```
SHOW DATABASES # > #은 mysql에서 주석입니다.
```

- 계정만들기

```
GRANT ALL PRIVILEGES ON mydb.* TO user01@localhost IDENTIFIED BY '1234';
```

```
EXIT;
```

- 새로운 계정으로 시작하기

```
mysql -u user01 -p
```

```
use mydb
```

HeidiSql

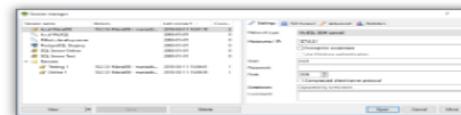
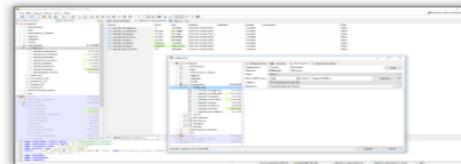
- <https://www.heidisql.com/>

The screenshot shows the official website for HeidiSQL at <https://www.heidisql.com/>. The page has a green header with the HeidiSQL logo and navigation links for Home, Downloads, Screenshots, Forum, Donate, Bugtracker, and Help. A banner on the right side features Korean text and a donation amount of "W25,000". The main content area includes a "What's this?" section with a brief description of the software, download links, and screenshots of the application interface. Below this is a "News" section with two recent posts: one about a filter box for snippets and another about a bugfix release.

What's this?

HeidiSQL is free software, and has the aim to be easy to learn. "Heidi" lets you see and edit data and structures from computers running one of the database systems MariaDB, MySQL, Microsoft SQL or PostgreSQL. Invented in 2002 by Ansgar, with a development peak between 2009 and 2013, HeidiSQL belongs to the most popular tools for MariaDB and MySQL worldwide.

Download HeidiSQL, read further about [features](#), take part in [discussions](#) or see some [screenshots](#).

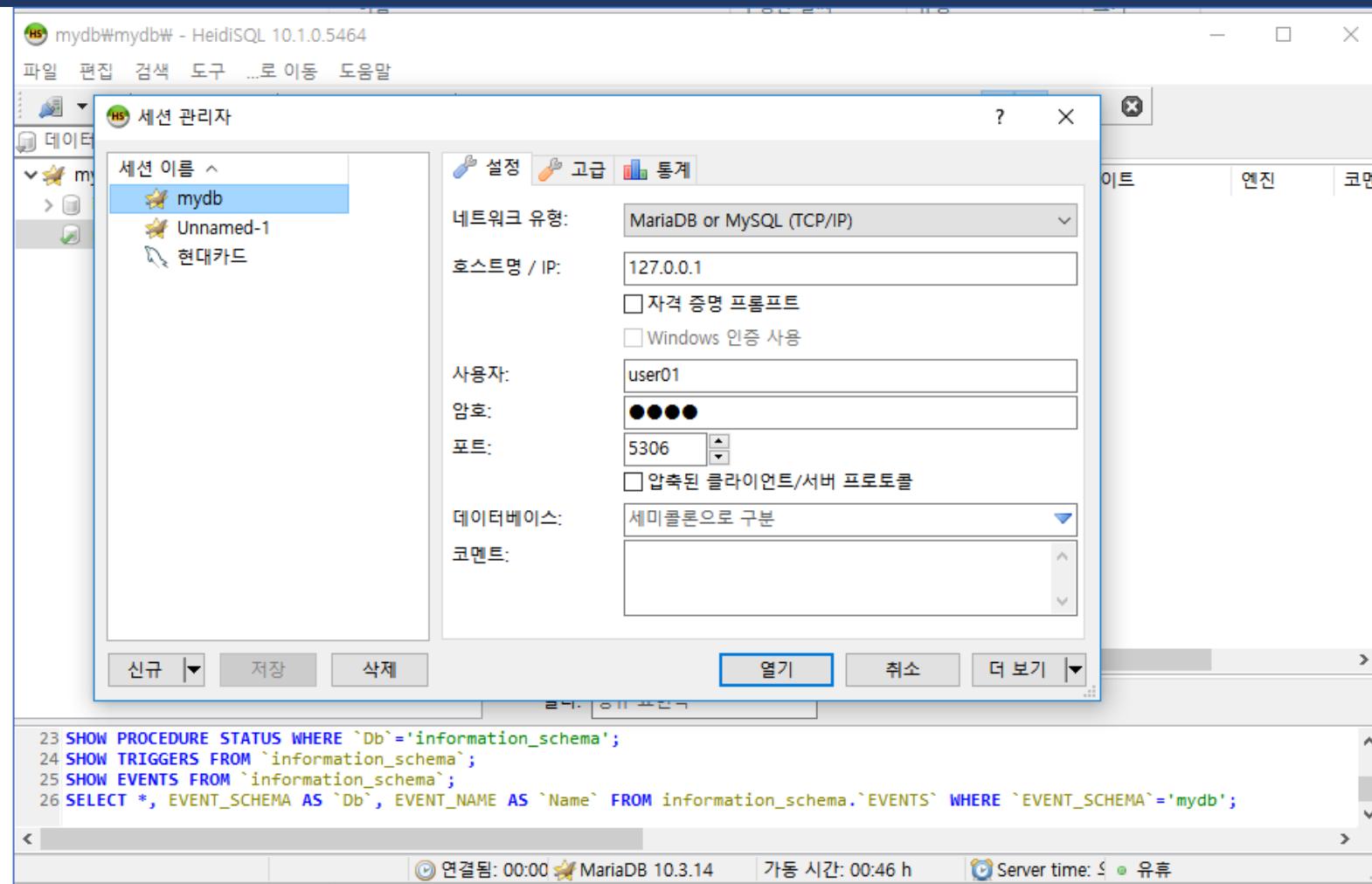


News [RSS](#)

07 Apr Filter box for snippets, functions etc.
I often found myself searching in the many snippets I stored over the years. To make it quick and easy to find anything in the query helpers box, I just added a filter box on top of it. The tree wi ...

26 Jan HeidiSQL 10.1 bugfix release
This is a new release which mainly fixes the installer, which named the executable in your program files folder "heidisql32.exe", not "heidisql.exe". Sounds minor, but I suppose there are quite a f ...

HeidiSql



guestbook table 작성

```
CREATE TABLE guestbook (

    id INT(11) NOT NULL AUTO_INCREMENT,
    title VARCHAR(300) NULL DEFAULT NULL,
    contents LONGTEXT NULL DEFAULT NULL,
    writer VARCHAR(50) NULL DEFAULT NULL,
    wdate DATE NULL DEFAULT NULL,
    PRIMARY KEY (id)
);
```

데이터추가

```
insert into guestbook(title, contents, writer, wdate) values  
('제목1', '내용1', '홍길동1', now());  
  
insert into guestbook(title, contents, writer, wdate) values  
('제목2', '내용2', '홍길동2', now());  
  
insert into guestbook(title, contents, writer, wdate) values  
('제목3', '내용3', '홍길동3', now());  
  
insert into guestbook(title, contents, writer, wdate) values  
('제목4', '내용4', '홍길동4', now());  
  
insert into guestbook(title, contents, writer, wdate) values  
('제목5', '내용5', '홍길동5', now());  
  
select * from guestbook;
```

MongoDB

MongoDB 설치하기

- 몽고디비 다운로드 및 설치
- <https://www.mongodb.com/download-center?jmp=nav#community>
- 설치 후 사용하기
- 사용 방법
- 환경 변수 path 에 다음 경로 추가하기
- C:\Program Files\MongoDB\Server\4.0\bin

MongoDB 사용하기

- 몽고디비 서버 작동하기
- mongod
- 데이터 들어갈 경로 없다고 함
- 경로 만들기 c:/data/db
- 데이터베이스
- c:/data/db
- 몽고디비 서버 작동하기
- mongod

MongoDB 사용하기

- 서버와 통신하기
- mongo
- 명령어
- >db
- test
- 디비를 모두 보여줘라
- >show dbs
- admin 0.000GB
- config 0.000GB
- local 0.000GB

MongoDB 사용하기

디비 사용 명령어 - 없으면 알아서 만들어준다

```
>use mydb
```

디비 삭제 명령어

```
>db.dropDatabase()
```

```
> show collections
```

```
> db.createCollection('person')
```

```
{ "ok" : 1 }
```

```
> show collections
```

```
person
```

MongoDB에서는 table도 record도 없다.
대신 비슷한 개념으로 각각 collection과 document가 존재한다.
RDBMS로 따지면 table과 record와 비슷한 개념이다.

MongoDB 사용하기

- 데이터 추가하기 형식
- db.<컬렉션명>.insert(<json>)
- db.person.insert({'name':'홍길동', 'age':26, 'gender':'m'})
- db.person.insert({'name':'장길산', 'age':62, 'gender':'m'})
- db.person.insert({'name':'임꺽정', 'age':28, 'gender':'m'})
- db.person.find()

MongoDB 사용하기

```
db.createCollection("member");

db.person.remove();

db.member.insert({'member_id':'test1', 'password':'1234',
                  'username':'홍길동', 'phone':'010-0000-0000'});
db.member.insert({'member_id':'test2', 'password':'1234',
                  'username':'김길동', 'phone':'010-1111-1111'});
db.member.insert({'member_id':'test3', 'password':'1234',
                  'username':'이길동', 'phone':'010-2222-2222'});
db.member.insert({'member_id':'test4', 'password':'1234',
                  'username':'장길동', 'phone':'010-3333-3333'});
db.member.insert({'member_id':'test5', 'password':'1234',
                  'username':'고길동', 'phone':'010-4444-4444'});

db.member.find();
```

MongoDB 사용하기

- 데이터 검색하기

1. 모든 데이터 출력

```
db.<컬렉션명>.find()
```

```
db.person.find()
```

2. 조건 출력

```
db.<컬렉션명>.find(<json>)
```

```
db.person.find({'name':'홍길동'})
```

find내에 조건을 넣지 않는다면 현재 컬렉션 내의 모든 데이터를 보여준다.

만약 조건을 건다면 해당 조건에 일치하는 데이터만 보여준다.

만약 해당 조건에 RDBMS처럼 LIKE(부분일치)로 데이터를 보고 싶다면 정규표현식을 조건으로 사용하면된다.

MongoDB 사용하기

- 데이터수정하기

```
db.<컬렉션명>.update(<json1>, <json2>)
```

```
db.person.update({'name':'홍길동'}, {$set:{'age':40}})
```

```
db.person.find()
```

- 데이터 삭제하기

```
db.<컬렉션명>.remove(<json>)
```

```
db.person.remove({'name':'홍길동'})
```

```
db.person.find()
```

MongoDB 사용하기

- 조건 부여하기
- \$eq >equals) 주어진 값과 일치하는 값
- \$gt =greater than) 주어진 값보다 큰 값
- \$gte =greather than or equals) 주어진 값보다 크거나 같은 값
- \$lt =less than) 주어진 값보다 작은 값
- \$lte =less than or equals) 주어진 값보다 작거나 같은 값
- \$ne (not equal) 주어진 값과 일치하지 않는 값
- \$in 주어진 배열 안에 속하는 값
- \$nin 주어진 배열 안에 속하지 않는 값

```
db.person.find({age:{$gte:40, $lt:60}})
```

```
db.person.find({age:{$gte:40}})
```

```
db.person.find({name:{$in:["장", "임"]}})
```

MongoDB 사용하기

- 자동 시퀀스

```
db.customSequences.insert(  
  {  
    _id: "hero", // 시퀀스 이름  
    seq: 0      // 초기 값  
  }  
)
```

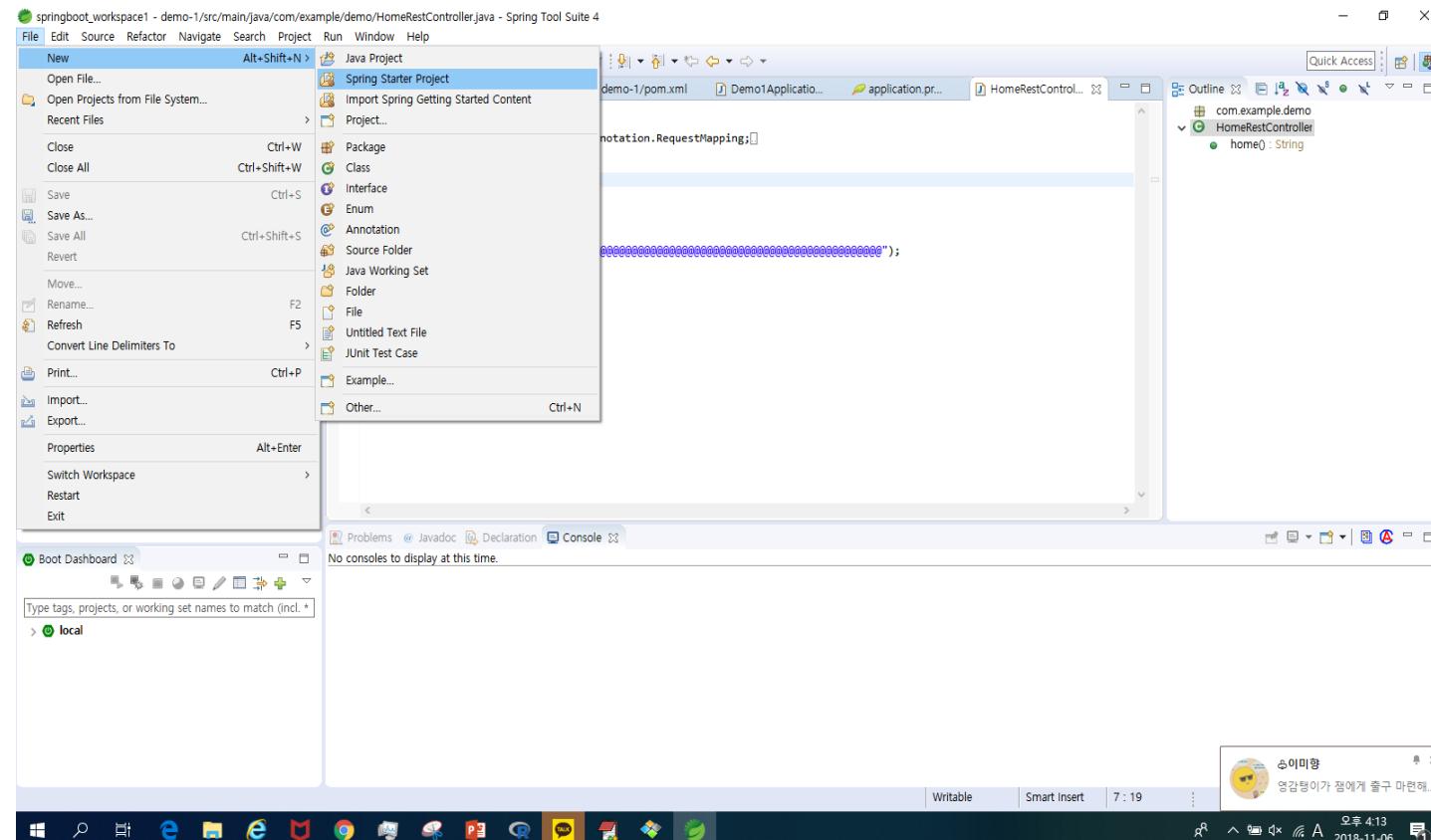
MongoDB 사용하기

```
db.system.js.save(  
  
{  "_id" : "getNextSequence",  
  "value" : function(name) {  
    var ret = db.customSequences.findAndModify(  
      {  
        query: { _id:name},    update: {$inc: { seq:1}}, new: true  
      });  
    return ret.seq;  
  }  
});  
  
db.loadServerScripts()  
  
getNextSequence("hero");
```

스프링 부트

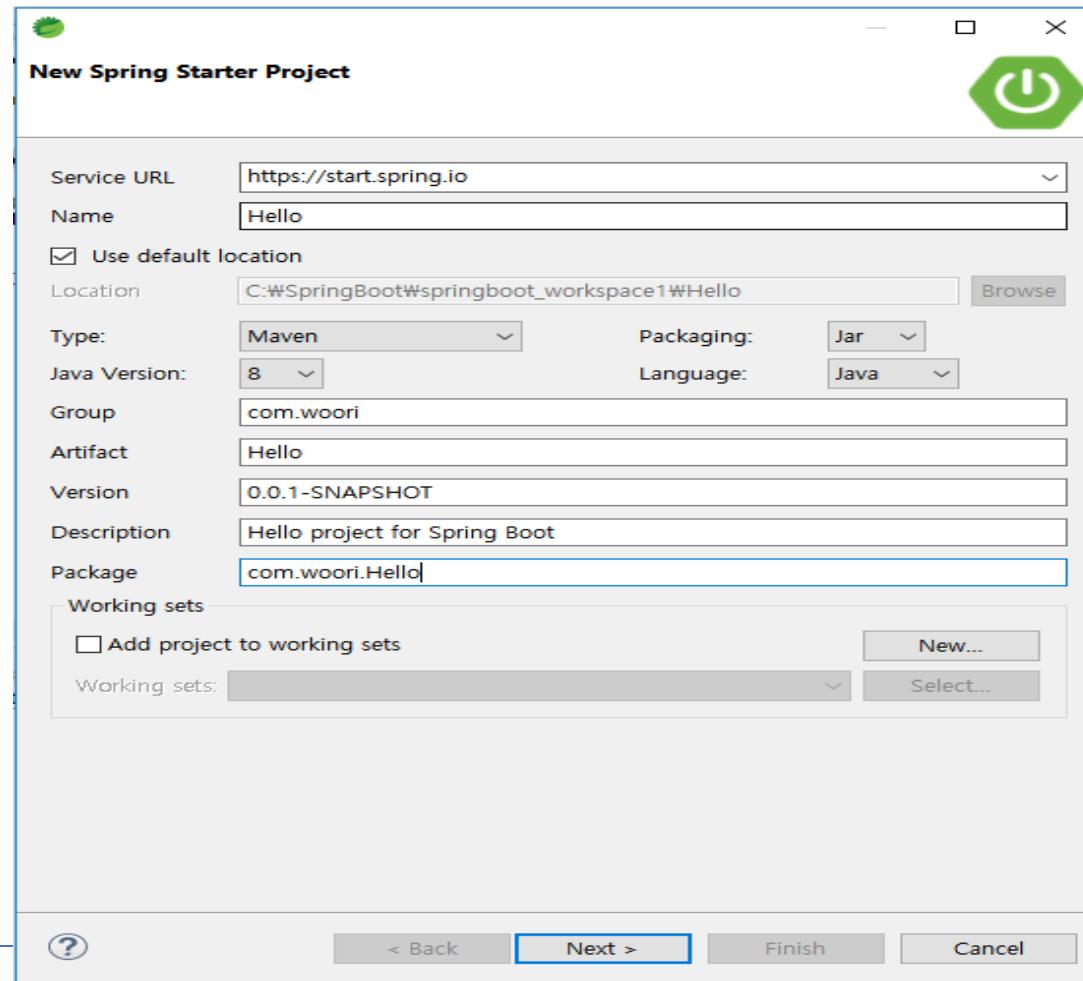
Hello, Web Application

- file – new – spring starter project

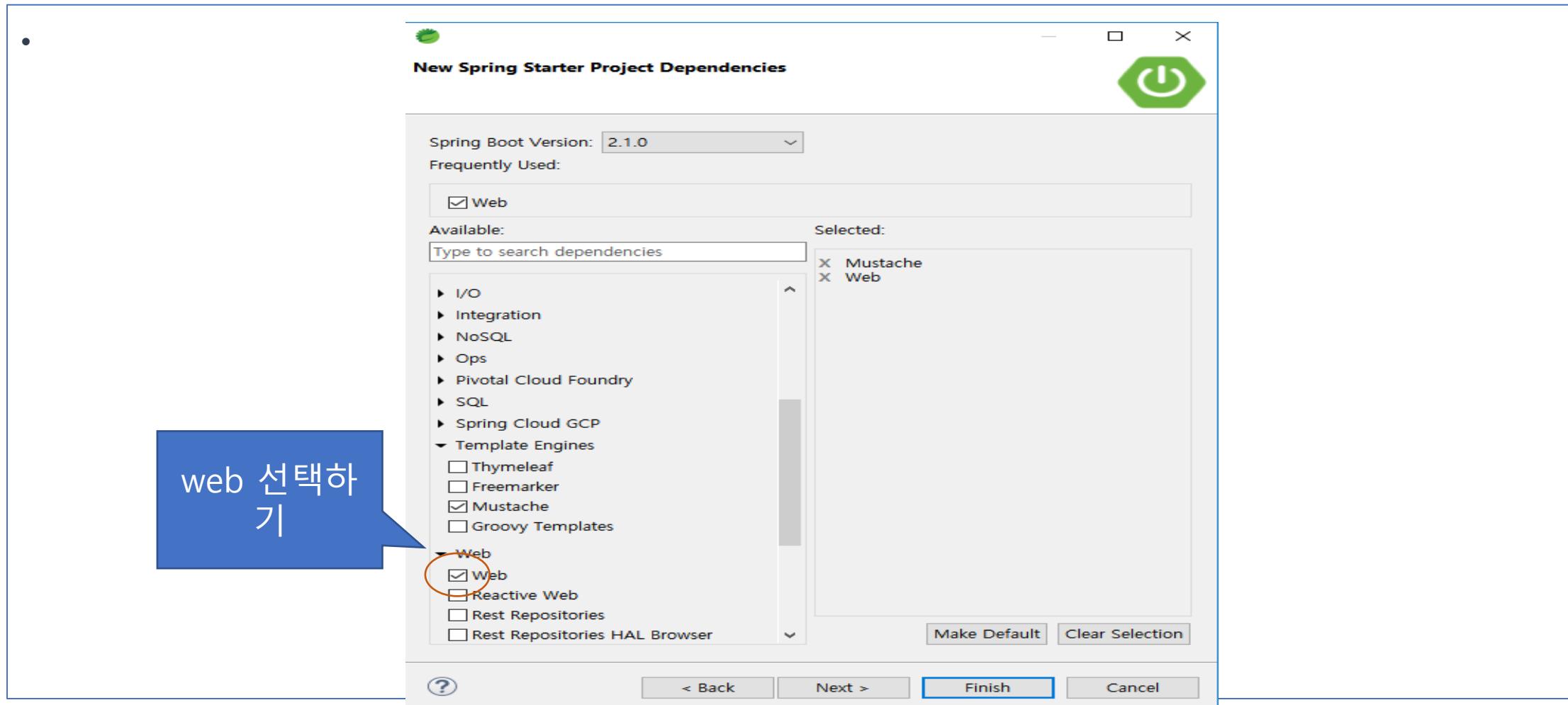


Hello, Web Application

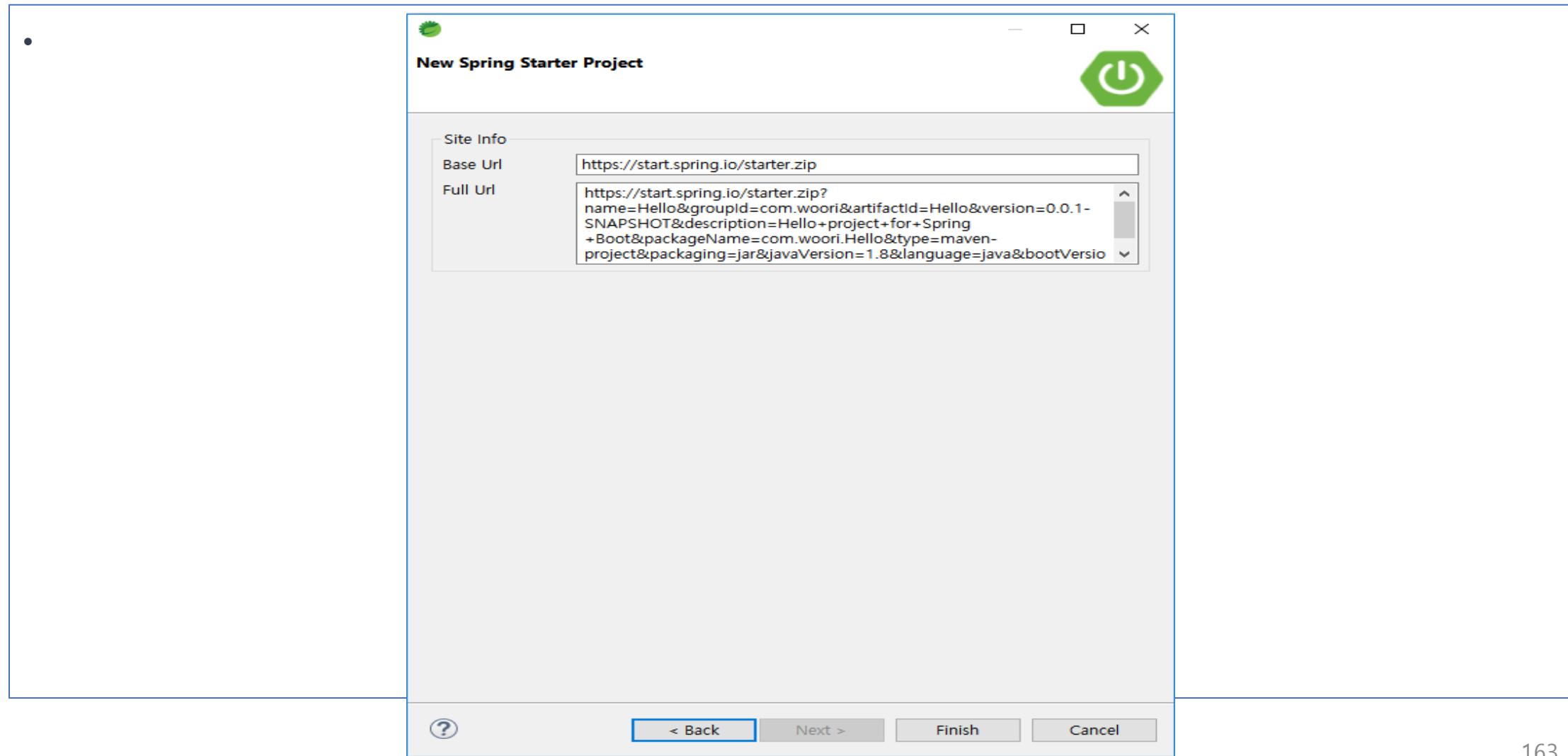
- Hello



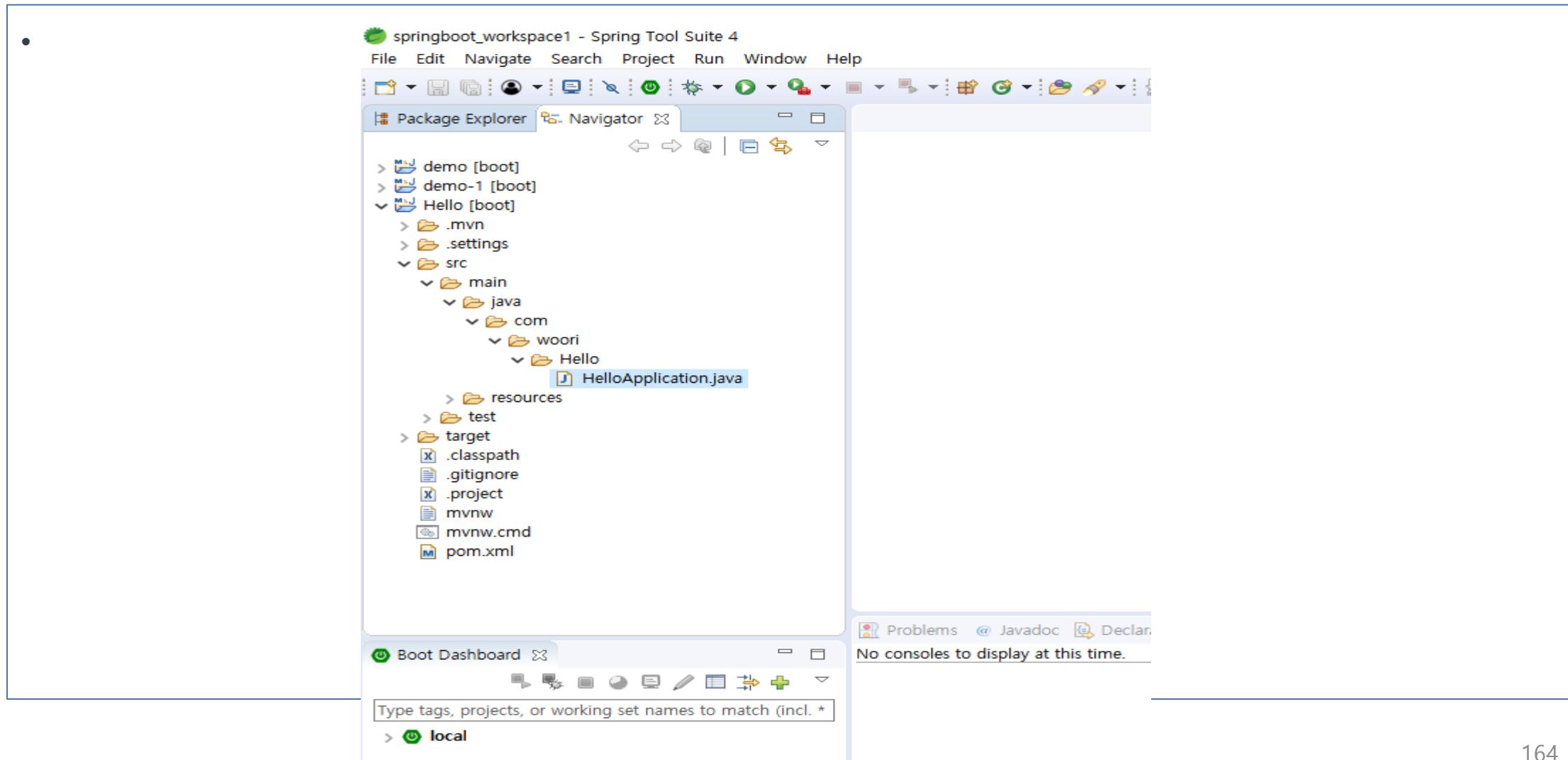
Hello, Web Application



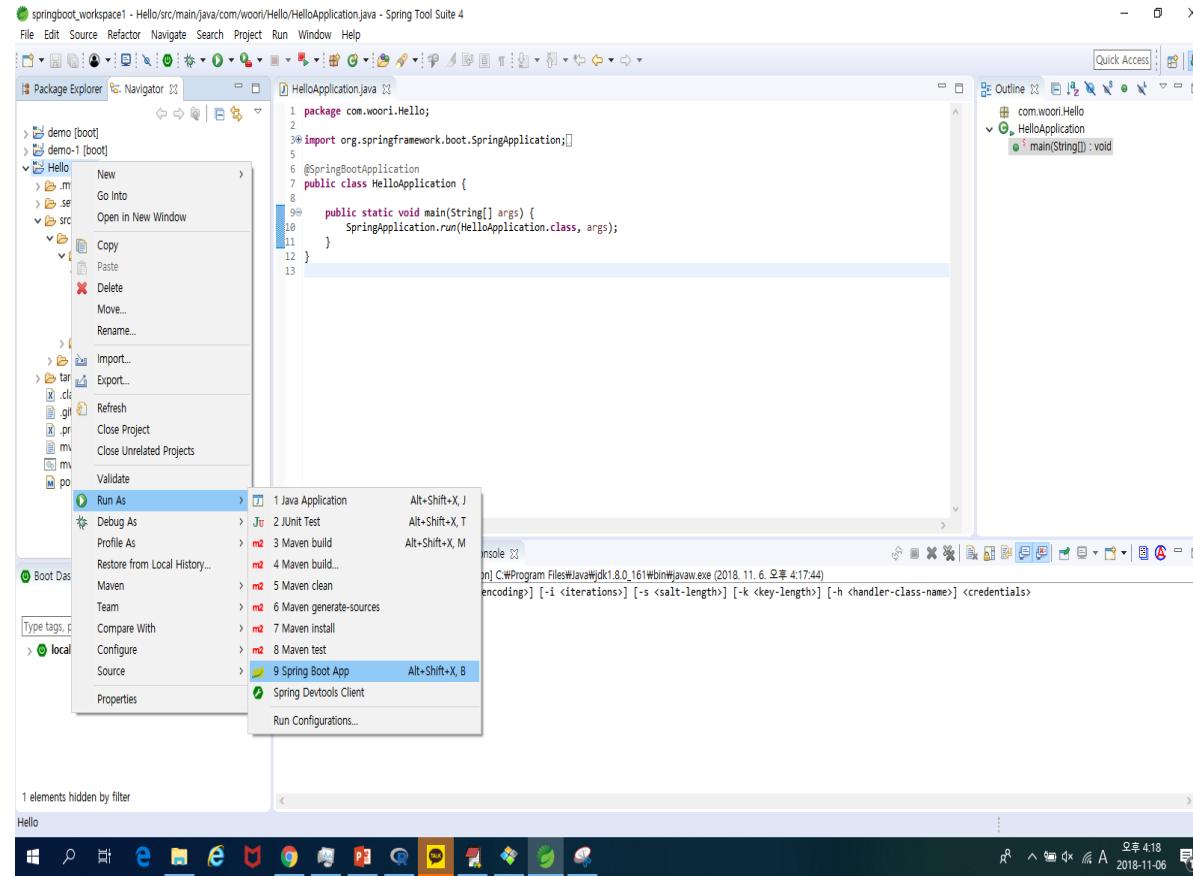
Hello, Web Application



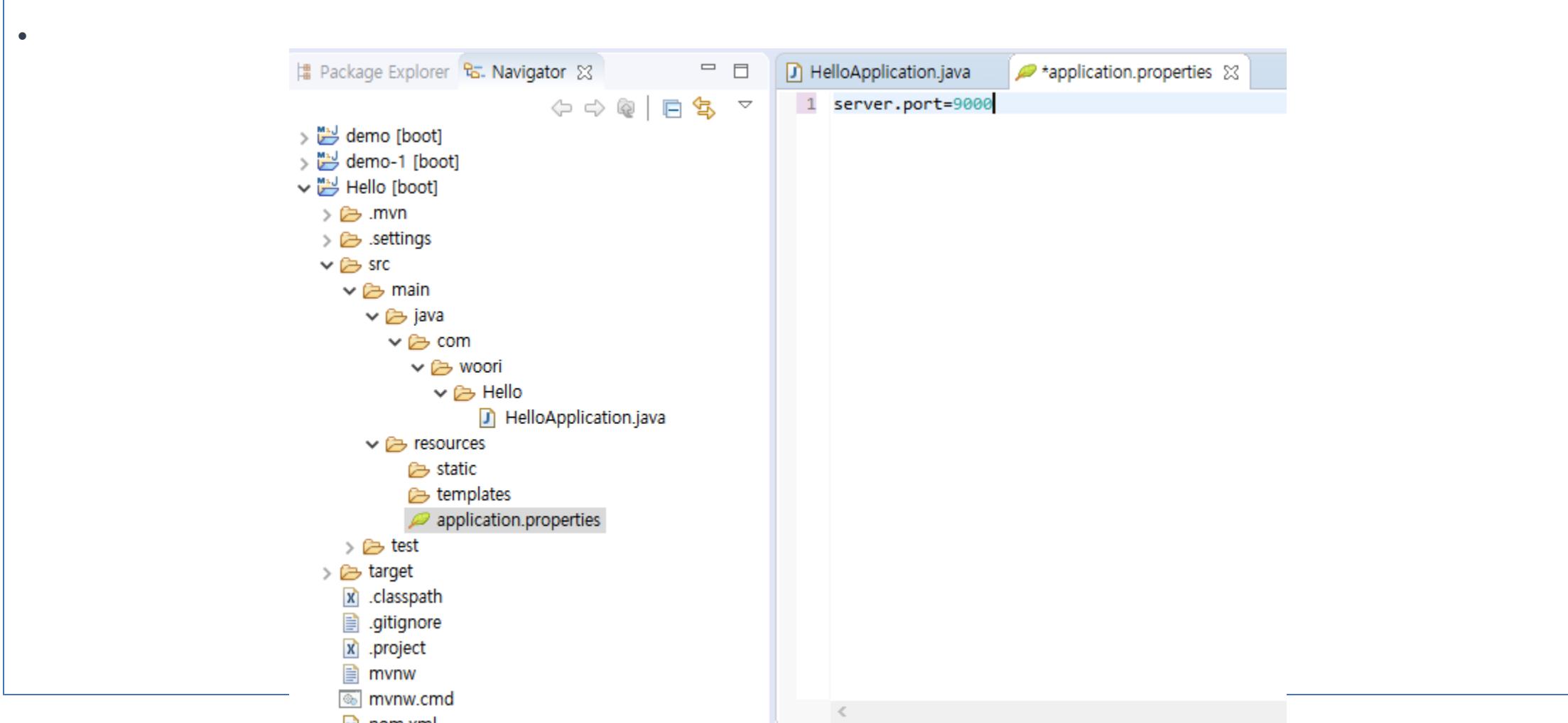
Hello, Web Application



Hello, Web Application



Hello, Web Application



Hello, Web Application

```
package com.woori.Hello;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

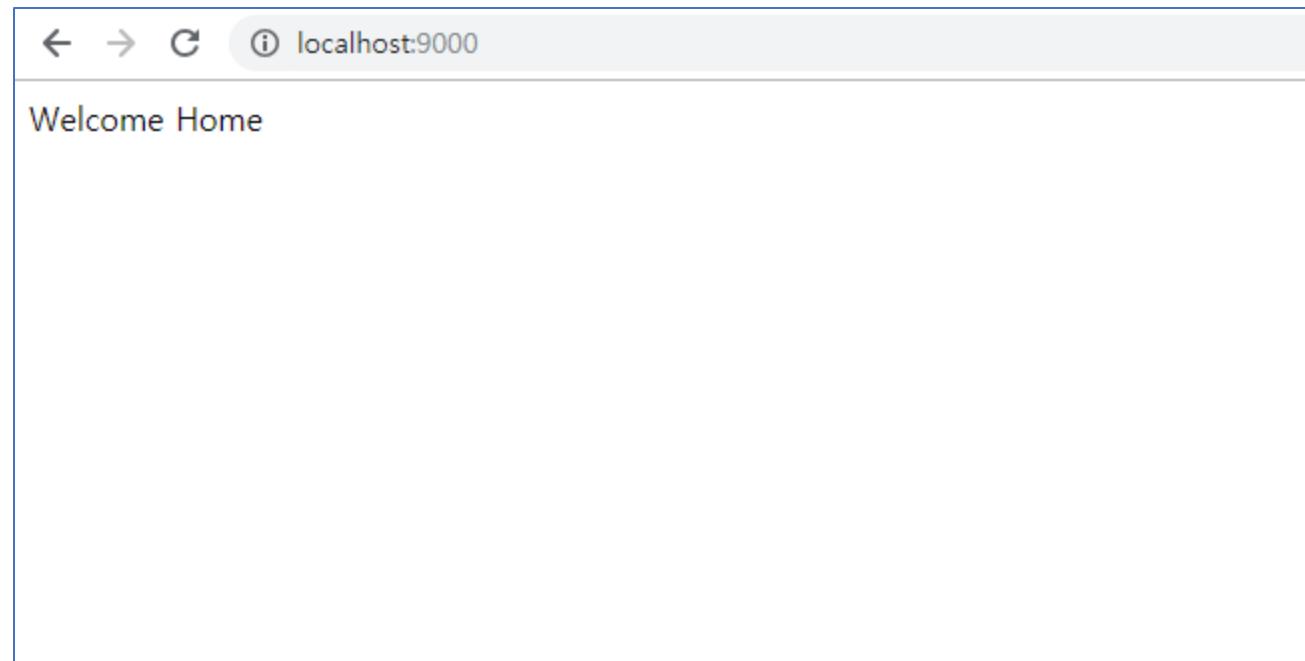
@RestController
public class HelloController {
    @RequestMapping("/")
    public String Hello()
    {
        return "Welcome Home";
    }
}
```

Hello, Web Application

```
      .--. 
     (( )) 
    \ \ / / 
   _ | | | 
  / \ / \ / 
 :: Spring Boot ::   (v2.1.0.RELEASE)

2018-11-06 16:21:11.625  INFO 16200 --- [           main] com.woori.Hello.HelloApplication      : Starting HelloApplication on LAPTOP-8IK90LES with PID 16200 (C:\SpringBoot\)
2018-11-06 16:21:11.629  INFO 16200 --- [           main] com.woori.Hello.HelloApplication      : No active profile set, falling back to default profiles: default
2018-11-06 16:21:12.611  INFO 16200 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 9000 (http)
2018-11-06 16:21:12.628  INFO 16200 --- [           main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2018-11-06 16:21:12.628  INFO 16200 --- [           main] org.apache.catalina.core.StandardEngine : Starting Servlet Engine: Apache Tomcat/9.0.12
2018-11-06 16:21:12.634  INFO 16200 --- [           main] o.a.catalina.core.AprLifecycleListener : The APR based Apache Tomcat Native library which allows optimal performance
2018-11-06 16:21:12.763  INFO 16200 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[]       : Initializing Spring embedded WebApplicationContext
2018-11-06 16:21:12.763  INFO 16200 --- [           main] o.s.web.context.ContextLoader          : Root WebApplicationContext: initialization completed in 1083 ms
2018-11-06 16:21:12.806  INFO 16200 --- [           main] o.s.b.w.servlet.ServletRegistrationBean : Servlet dispatcherServlet mapped to [/]
2018-11-06 16:21:12.810  INFO 16200 --- [           main] o.s.b.w.servlet.FilterRegistrationBean  : Mapping filter: 'characterEncodingFilter' to: [*]
2018-11-06 16:21:12.810  INFO 16200 --- [           main] o.s.b.w.servlet.FilterRegistrationBean  : Mapping filter: 'hiddenHttpMethodFilter' to: [*]
2018-11-06 16:21:12.810  INFO 16200 --- [           main] o.s.b.w.servlet.FilterRegistrationBean  : Mapping filter: 'formContentFilter' to: [*]
2018-11-06 16:21:12.810  INFO 16200 --- [           main] o.s.b.w.servlet.FilterRegistrationBean  : Mapping filter: 'requestContextFilter' to: [*]
2018-11-06 16:21:13.006  INFO 16200 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2018-11-06 16:21:13.224  INFO 16200 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 9000 (http) with context path ''
2018-11-06 16:21:13.227  INFO 16200 --- [           main] com.woori.Hello.HelloApplication      : Started HelloApplication in 1.944 seconds (JVM running for 2.921)
```

Hello, Web Application

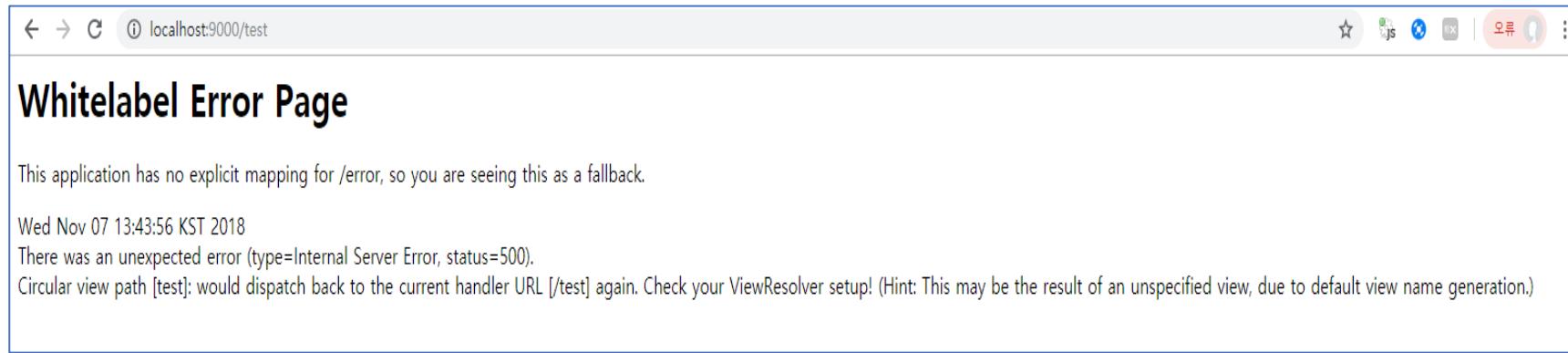


Hello, Web Application

- **@RestController** : 별도의 html 페이지가 필요 없다.
- **@Controller** : 별도의 html 페이지가 필요하다

```
1 package com.woori.Hello;
2
3 import org.springframework.web.bind.annotation.RestController;
4 import org.springframework.web.bind.annotation.RequestMapping;
5 import org.springframework.stereotype.Controller;
6
7 @Controller
8 public class TestController {
9
10    @RequestMapping("/test")
11    public String test()
12    {
13        return "test";
14    }
15}
16
```

Hello, Web Application



Hello, Web Application

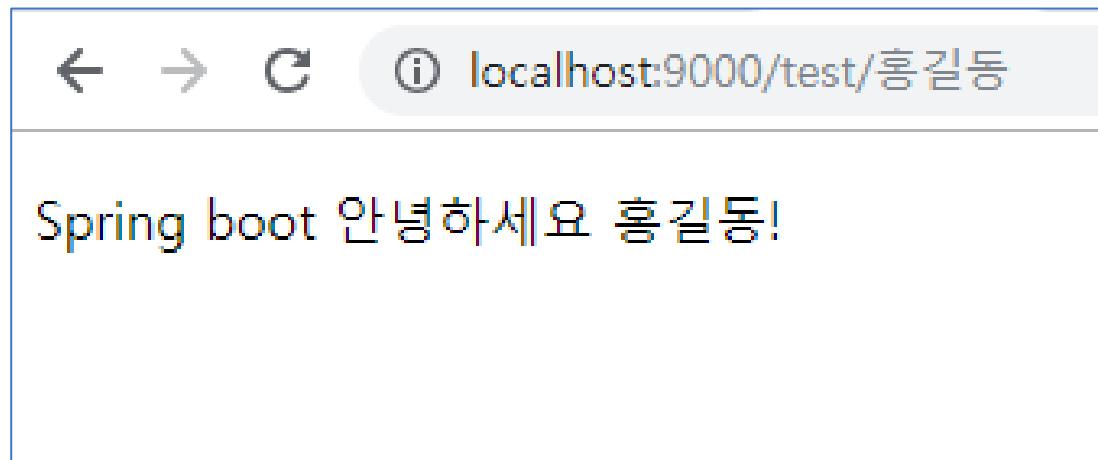
- html 뷰 지원하려면 반드시 아래 코드를 pom.xml에 넣어야 한다

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mustache</artifactId>
</dependency>    <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mustache</artifactId>
</dependency>
```

Hello, Web Application

- `src/main/resources/application.properties` 파일에
- 아래처럼 기술해야 한다
- `spring.mustache.suffix: .html`

실행 결과



참고

- 자동 import 구문 안보일때 pom.xml 파일에 다음 추가

```
<dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-actuator</artifactId>  
</dependency>  
    <dependency>  
        <groupId>org.springframework.boot</groupId>  
        <artifactId>spring-boot-starter-actuator</artifactId>  
    </dependency>
```

jsp 사용하려면

- **eclipse –help – market place**
- **Eclipse EE Developer Tool 설치하기**

pom.xml에 추가

```
<!-- jsp 지원하기 -->

<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
</dependency>

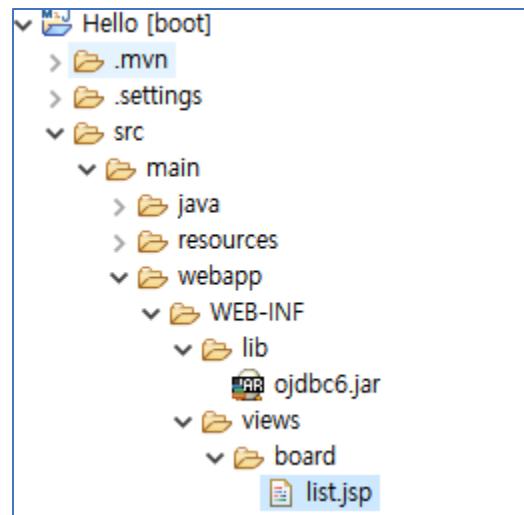
<!-- jstl 지원하기 -->

<dependency>
    <groupId>javax.servlet.jsp.jstl</groupId>
    <artifactId>javax.servlet.jsp.jstl-api</artifactId>
    <version>1.2.1</version>
</dependency>
```

application.properties

- 추가
- `spring.mvc.view.prefix=/WEB-INF/views/`
- `spring.mvc.view.suffix=.jsp`

폴더 구조



몽고디비와 연동(몽고디비 계정만들기)

```
use admin

db.createUser({user:'test', pwd:'1234',
    roles: [ "userAdminAnyDatabase",
        "dbAdminAnyDatabase",   "readWriteAnyDatabase"]});

use mydb

db.createUser({ user: "test",
    pwd: "1234",
    roles: ["dbAdmin", "readWrite"]
})

})
```

몽고디비와 연동하기

pom.xml 작업내역

```
<!-- mongo db -->
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-mongodb -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
    <version>2.1.3.RELEASE</version>
</dependency>

<!-- 클래스에 대한 도움을 받는다. 가급적 붙여놓고 시작하자 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

application.property

```
server.port=9000

spring.data.mongodb.uri=mongodb://127.0.0.1/mydb

#Local MongoDB config

#spring.data.mongodb.authentication-database=admin

spring.data.mongodb.username=test

spring.data.mongodb.password=1234

spring.data.mongodb.database=mydb

spring.data.mongodb.port=27017

spring.data.mongodb.host=127.0.0.1

spring.application.name=BootMongo

server.context-path=/user
```

몽고디비 연동(데이터 추가)

```
db.createCollection('hero');

db.hero.remove({});

db.hero.insert({'id':1, name:'이순신1', description:'임진왜란 격퇴1'});

db.hero.insert({'id':2, name:'이순신2', description:'임진왜란 격퇴2'});

db.hero.insert({'id':3, name:'이순신3', description:'임진왜란 격퇴3'});

db.hero.insert({'id':4, name:'이순신4', description:'임진왜란 격퇴4'});

db.hero.insert({'id':5, name:'이순신5', description:'임진왜란 격퇴5'});

db.hero.insert({'id':6, name:'이순신6', description:'임진왜란 격퇴6'});

db.hero.insert({'id':7, name:'이순신7', description:'임진왜란 격퇴7'});

db.hero.find();
```

AutoSequence부여

```
db.customSequences.insert(  
{  
    _id: "hero", // 시퀀스 이름  
    seq: 0       // 초기 값  
}  
)  
  
db.system.js.save(  
{  
    "_id" : "getNextSequence",  
    "value" : function(name) {  
        var ret = db.customSequences.findAndModify(  
        {  
            query: { _id:name},  
            update: {$inc: { seq:1}},  
            new: true  
        }  
        );  
        return ret.seq;  
    }  
});
```

AutoSequence부여

```
db.loadServerScripts()  
  
getNextSequence("hero");  
  
db.hero.remove({});  
  
db.hero.insert({id:getNextSequence("hero"), name:'이순신1', description:'임진왜란 격퇴1'});  
  
db.hero.insert({id:getNextSequence("hero"), name:'이순신2', description:'임진왜란 격퇴2'});  
  
db.hero.insert({id:getNextSequence("hero"), name:'이순신3', description:'임진왜란 격퇴3'});  
  
db.hero.insert({id:getNextSequence("hero"), name:'이순신4', description:'임진왜란 격퇴4'});  
  
db.hero.insert({id:getNextSequence("hero"), name:'이순신5', description:'임진왜란 격퇴5'});  
  
db.hero.insert({id:getNextSequence("hero"), name:'이순신6', description:'임진왜란 격퇴6'});
```

HeroDto

```
import org.springframework.data.annotation.Id;  
  
import org.springframework.data.mongodb.core.mapping.Document;  
  
@Document(collection = "hero")  
  
public class HeroMongoDto {  
  
    @Id //시스템 내의 object id임  
  
    private String _id;  
  
    private int id;  
  
    private String name;  
  
    private String description;      //getter/setter 추가해야 함
```

HeroMongoDao

```
@Component("mongoDao")
public class HeroMongoDao{
    @Autowired
    MongoTemplate mongoTemplate;

    public List<HeroMongoDto> getList()
    {
        HeroMongoDto dto = new HeroMongoDto();
        //전부 가져오기
        //List<HeroMongoDto> list = mongoTemplate.findAll(HeroMongoDto.class);

        // Limit 조건 정의
        LimitOperation limit = Aggregation.limit(10);

        // 여러 조건을 정의함
        // 지금은 Limit 만 정의
        //10개만 가져오기
        Aggregation aggregation = Aggregation.newAggregation(limit);

        AggregationResults<HeroMongoDto> result
            = mongoTemplate.aggregate(aggregation, "hero", HeroMongoDto.class);

        List<HeroMongoDto> list = result.getMappedResults();

        for(int i=0; i<list.size(); i++)
        {
            System.out.println(list.get(i).getName());
        }
        return list;
    }
}
```

HeroMongoDao

```
@Component("mongoDao")
public class HeroMongoDao{
    @Autowired
    MongoTemplate mongoTemplate;

    public List<HeroMongoDto> getList()
    {
        HeroMongoDto dto = new HeroMongoDto();
        //전부 가져오기
        //List<HeroMongoDto> list = mongoTemplate.findAll(HeroMongoDto.class);

        // Limit 조건 정의
        LimitOperation limit = Aggregation.limit(10);

        // 여러 조건을 정의함
        // 지금은 Limit 만 정의
        //10개만 가져오기
        Aggregation aggregation = Aggregation.newAggregation(limit);

        AggregationResults<HeroMongoDto> result
            = mongoTemplate.aggregate(aggregation, "hero", HeroMongoDto.class);

        List<HeroMongoDto> list = result.getMappedResults();

        for(int i=0; i<list.size(); i++)
        {
            System.out.println(list.get(i).getName());
        }
        return list;
    }
}
```

HeroMongoDao

```
public void insert(HeroMongoDto dto)
{
    //getNextSequence("hero")

    dto.setId( SequenceGenerator.generateSequence(mongoTemplate, "hero"));
    mongoTemplate.insert(dto);
    mongoTemplate.save(dto);
}

public void update(int id, HeroMongoDto dto)
{
    //Criteria criteria = new Criteria("id");
    //criteria.is(id);

    //Query query = new Query(criteria);
    mongoTemplate.save(dto, "hero");

    /*
     *
     * user = mongoTemplate.findOne(
    Query.query(Criteria.where("name").is("Jack")), User.class);
    user.setName("Jim");
    */
}

}
```

HeroMongoDao

```
public void delete(int id)
{
    Criteria criteria = new Criteria("id");
    criteria.is(id);

    Query query = new Query(criteria);

    mongoTemplate.remove(query, "hero");

}
```

SequenceGenerator

```
import org.springframework.data.mongodb.core.FindAndModifyOptions;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;
import org.springframework.data.mongodb.core.query.Update;

public class SequenceGenerator {
    private SequenceGenerator() {}

    public static int generateSequence(MongoTemplate mongoTemplate, String seqName) {
        //get sequence id
        Query query = new Query(Criteria.where("_id").is(seqName));

        //increase sequence id by 1
        Update update = new Update();
        update.inc("seq", 1);

        //return new increased id
        FindAndModifyOptions options = new FindAndModifyOptions();
        options.returnNew(true);

        //this is the magic happened.
        CustomSequence seqId = mongoTemplate.findAndModify(query, update, options, CustomSequence.class);

        if(seqId==null) return 1;
        return seqId.getSeq();
    }
}
```

CustomSequence.java

```
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "customSequences")
public class CustomSequence {
    @Id
    String id;
    int seq;
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public int getSeq() {
        return seq;
    }
    public void setSeq(int seq) {
        this.seq = seq;
    }
}
```

HeroController.java

```
//중요
@CrossOrigin(origins = "http://127.0.0.1:4200")
@RestController
public class HeroMongoController {
    @Resource(name="mongoDao")
    HeroMongoDao mongoDao;

    @RequestMapping("/mongo/list")
    public List<HeroMongoDto> getHeros() {
        return mongoDao.getList();
    }

    @RequestMapping("/mongo/{id}")
    public HeroMongoDto getHero(@PathVariable("id") int id) {
        return mongoDao.getHero(id);
    }

    @RequestMapping("/mongo/save")
    public HeroMongoDto saveHero(@RequestBody HeroMongoDto heroDto) {
        mongoDao.insert(heroDto);
        return heroDto;
    }
}
```

HeroController.java

```
@RequestMapping("/mongo/update")
public HeroMongoDto updateHero (@RequestBody HeroMongoDto heroDto) {

    mongoDao.update(heroDto.getId(), heroDto);
    HeroMongoDto dto = mongoDao.getHero(0);
    return null;
}

@RequestMapping("/mongo/delete/{id}")
public HeroMongoDto deleteHero (@PathVariable("id") int id) {

    mongoDao.delete(id);
    HeroMongoDto heroDto = mongoDao.getHero(0);
    return null;
}

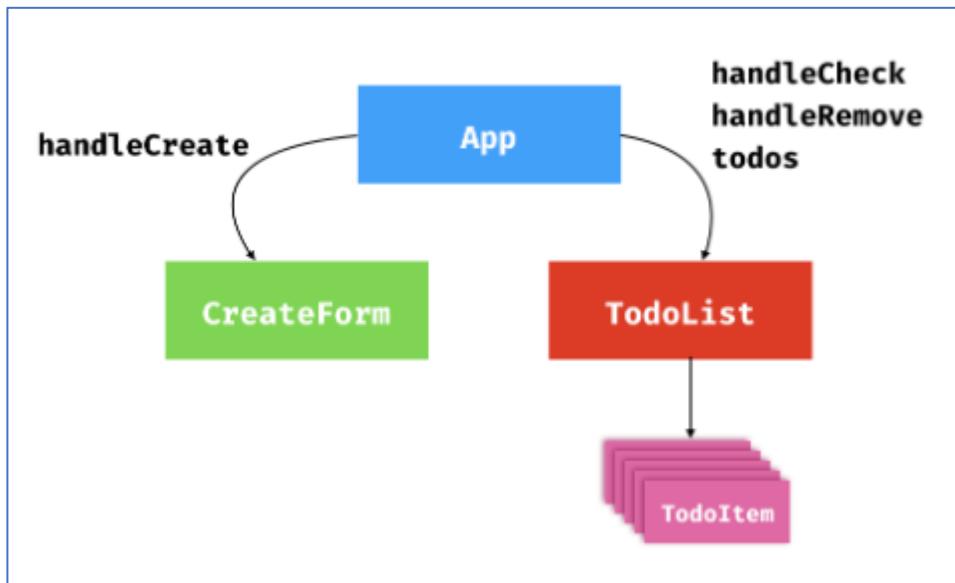
}
```

리덕스

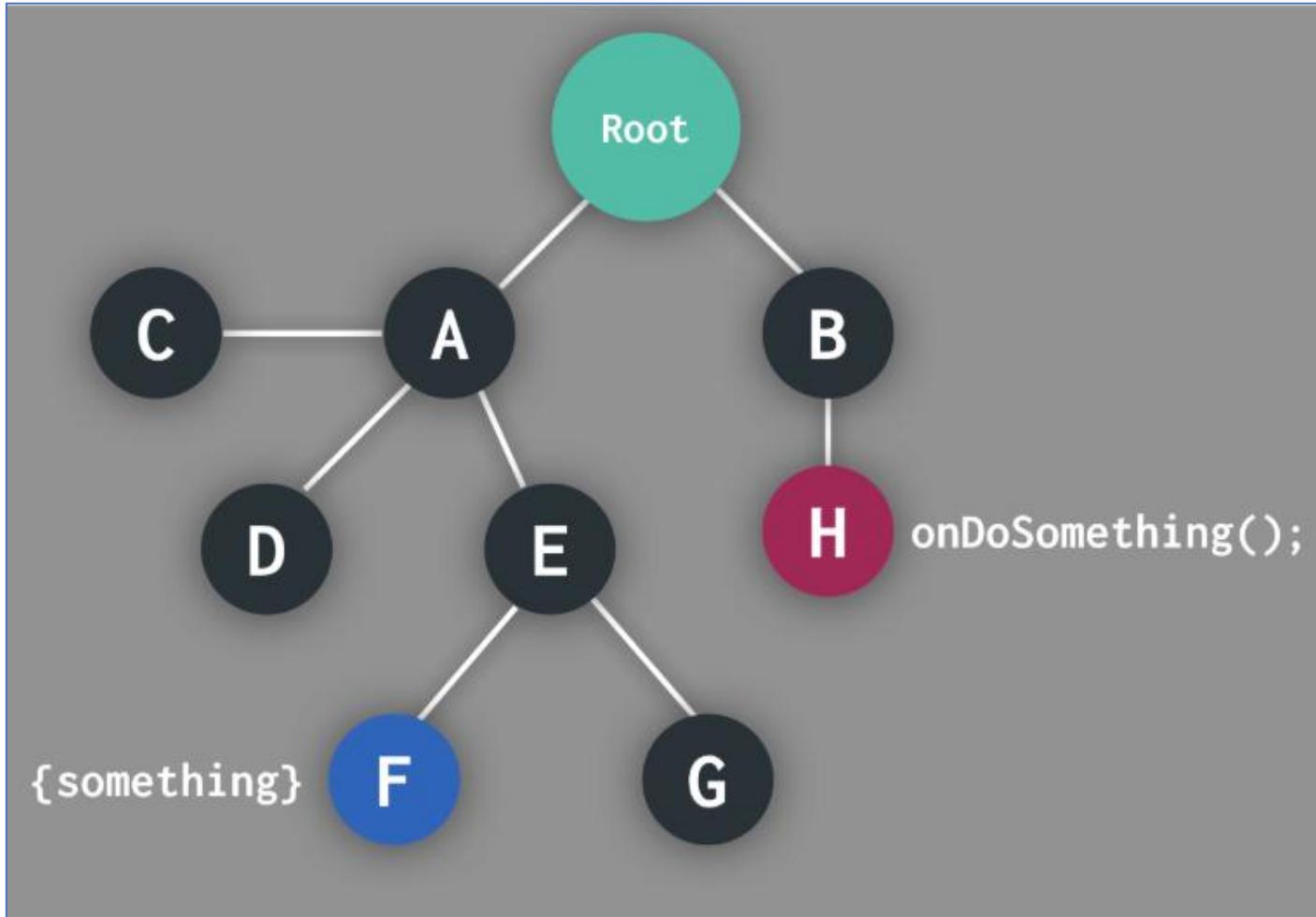
리덕스란

- 리덕스는, 가장 사용률이 높은 상태관리 라이브러리입니다. 리덕스를 사용하면, 여러분이 만들게 될 컴포넌트들의 상태 관련 로직들을 다른 파일들로 분리시켜서 더욱 효율적으로 관리 할 수 있습니다. 또한, 컴포넌트끼리 상태를 공유하게 될 때 여러 컴포넌트를 거치지 않고도 손쉽게 상태 값을 전달 할 수 있습니다.
- 추가적으로, 리덕스의 미들웨어라는 기능을 통하여 비동기 작업, 로깅 등의 확장적인 작업들을 더욱 쉽게 할 수도 있게 해줍니다

리덕스를 사용하지 않을 때



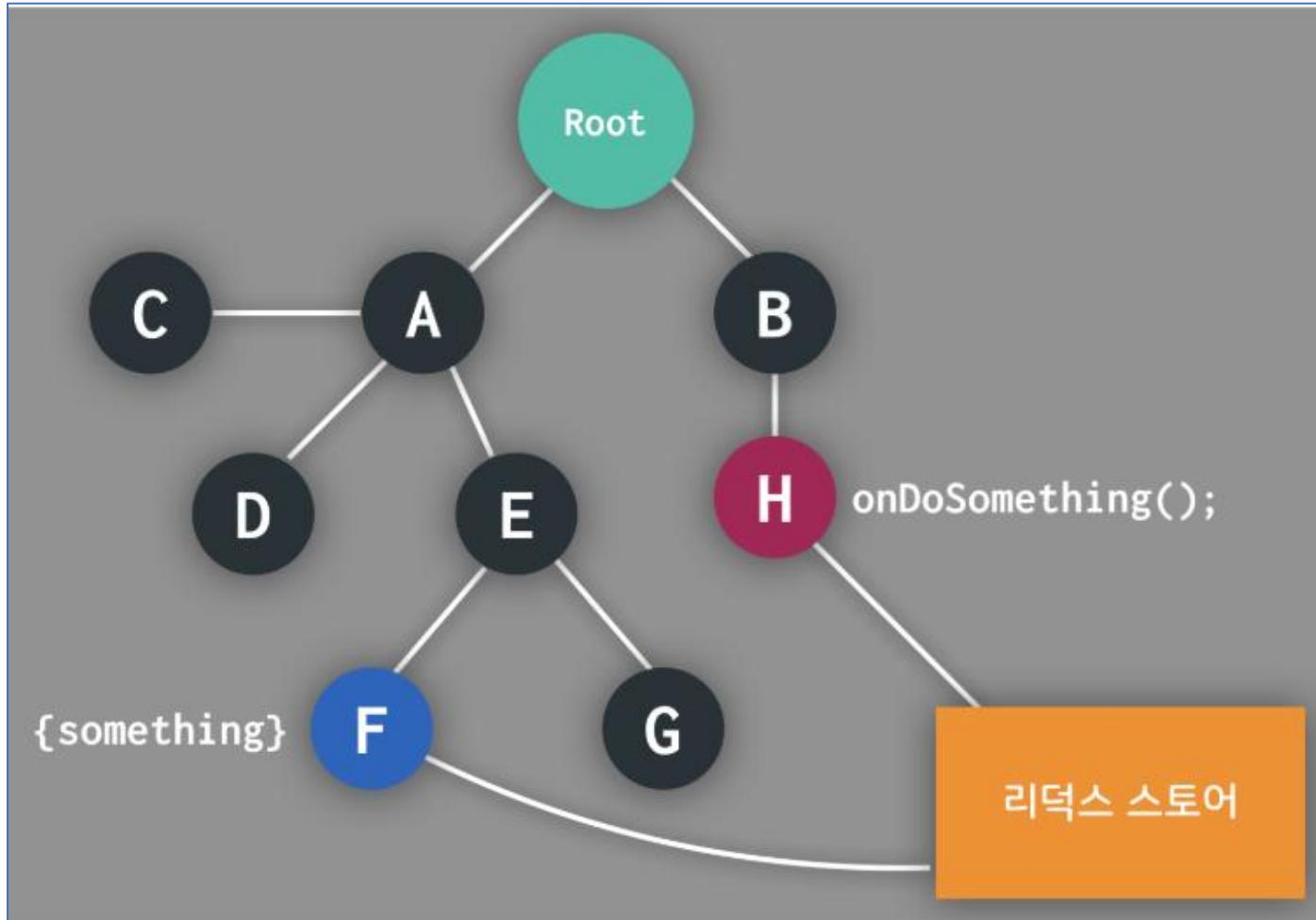
리덕스를 사용하지 않을 때



리덕스 사용안할때

- Root 컴포넌트에는 something이라는 상태 값이 있고, onDoSomething이라는 함수가 something 값에 변화를 줍니다.
- onDoSomething은 Root -> B -> H로 전달되고, H에서 이벤트가 발생하여 이 함수가 호출되면 something이 Root -> A -> E -> F로 전달됩니다.
- props가 필요한 곳으로 제대로 전달되게 하기 위하여, 실제로는 해당 props를 사용하지 않는 컴포넌트를 거쳐 가야 한다는 것은 리렌더링하게 될 때 비효율적이기도하고, 굉장히 귀찮은 작업이기도 합니다. 상위 컴포넌트에서 props 이름을 바꿔준다면 그 아래에도 쭉 바꿔줘야 하니까요.
- 리덕스가 있다면, 다음과 같은 구조로 작업을 진행 할 수 있게 됩니다.

리덕스 사용시



리덕스 사용이유

- 앱이 지니고 있는 상태와, 상태 변화 로직이 들어있는 **스토어**를 통하여, 우리가 원하는 컴포넌트에 원하는 상태 값과 함수를 직접 주입해줄 수 있게 됩니다.
- 이런 식으로, 더 쉬운 글로벌 상태 관리를 위하여 리덕스를 사용하기도 하고, 조금 더 체계적이고 편리한 상태 관리를 하기 위하여 사용을 하는데

액션 (Action)

- 상태에 어떠한 변화가 필요하게 될 땐, 우리는 액션이란 것을 발생시킵니다.
- 액션 객체는 type 필드를 필수적으로 가지고 있어야 한다

```
{  
  type: "ADD_TODO",  
  data: {  
    id: 0,  
    text: "리덕스 배우기"  
  }  
}
```

액션 생성함수 (Action Creator)

- 액션 생성함수는, 액션을 만드는 함수입니다. 단순히 파라미터를 받아와서 액션 객체 형태로 만들어줍니다.

```
function addTodo(data) {
  return {
    type: "ADD_TODO",
    data
  };
}

// 화살표 함수로도 만들 수 있습니다.
const changeInput = text => ({
  type: "CHANGE_INPUT",
  text
});
```

리듀서 (Reducer)

- 리듀서는 변화를 일으키는 함수입니다.
- 리듀서는 두가지의 파라미터를 받아옵니다
- 리듀서는, 현재의 상태와, 전달 받은 액션을 참고하여 새로운 상태를 만들어서 반환합니다

```
function reducer(state, action) {  
  // 상태 업데이트 로직  
  return alteredState;  
}
```

스토어 (Store)

- 리덕스에서는 한 애플리케이션 당 하나의 스토어를 만들게 됩니다. 스토어 안에는, 현재의 앱 상태와, 리듀서가 들어가있고, 추가적으로 몇가지 내장 함수들이 있습니다.
- 두개이상 만들수는 있지만 권장하지는 않습니다

디스패치 (dispatch)

- 디스패치는 스토어의 내장함수 중 하나입니다.
- 디스패치는, 액션을 발생 시킵니다
- dispatch 라는 함수에는 액션을 파라미터로 전달합니다.
- 그렇게 호출을 하면, 스토어는 리듀서 함수를 실행시켜서 해당 액션을 처리하는 로직이 있다면 액션을 참고하여 새로운 상태를 만들어줍니다.

구독 (subscribe)

- subscribe 함수는, 함수 형태의 값을 파라미터로 받아옵니다.
- subscribe 함수에 특정 함수를 전달해주면, 액션이 디스패치 되었을 때 마다 전달해준 함수가 호출됩니다.

참고자료

- <https://react.vlpt.us/basic/01-concept.html> 라우팅