8253jang@daum.net
장희정

SpringBoot기반의 Restful API와 React 연결하기

주요기능

- 회원관리(아이디, 비밀번호,이름, 주소, role, 등록일)
  회원가입(아이디중복체크)
  로그인
  로그아웃

- 게시판관리(글번호, 제목, 내용, 작성자, 등록일, 수정일)
  게시물 전체조회 – 아무나 조회
  게시물 상세보기 – 인증된 사용자 상세보기
  게시물등록 – 인증된 사용자 만이 등록가능
  게시물수정 - 인증된 사용자 + 자신이 쓴 글인 경우만
  게시물 삭제- 인증된 사용자 + 자신이 쓴 글인 경우만

React환경 세팅

1. 프로젝트를 생성한다.

```
> npx create-react-app step10-client-jwt
```

2. 생성된 프로젝트로 이동한다.

```
cd step10-client-jwt
```

3. 필요한 lib install한다.

8253jang@daum.net
장희정

npm install react-router-dom

npm install react-axios

npm install react-bootstrap bootstrap

npm i styled-components

package.json파일확인

```
"bootstrap": "^5.3.3",
"react": "^18.3.1",
"react-axios": "^2.0.6",
"react-bootstrap": "^2.10.2",
"react-dom": "^18.3.1",
"react-router-dom": "^6.23.1",
"react-scripts": "5.0.1",
"styled-components": "^6.1.11",
"web-vitals": "^2.1.4"
```

8253jang@daum.net
장희정

App.js index.js 불필요한것들은 다 지우고 아래 상태로 만들어 놓고 시작한다.

```
nt > src > JS index.js > ...
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

const root = ReactDOM.createRoot(document
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

```
nt > src > JS App.js > ...
import React from 'react';
function App() {
  return (
    <div>
    </div>
  );
}
export default App;
```

4) 라우터 걸기

index.js

```
t > src > JS index.js > ...
import {BrowserRouter} from 'react-router-dom';
import 'bootstrap/dist/css/bootstrap.min.css';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <BrowserRouter>
    <App />
    </BrowserRouter>
  </React.StrictMode>
```

App.js

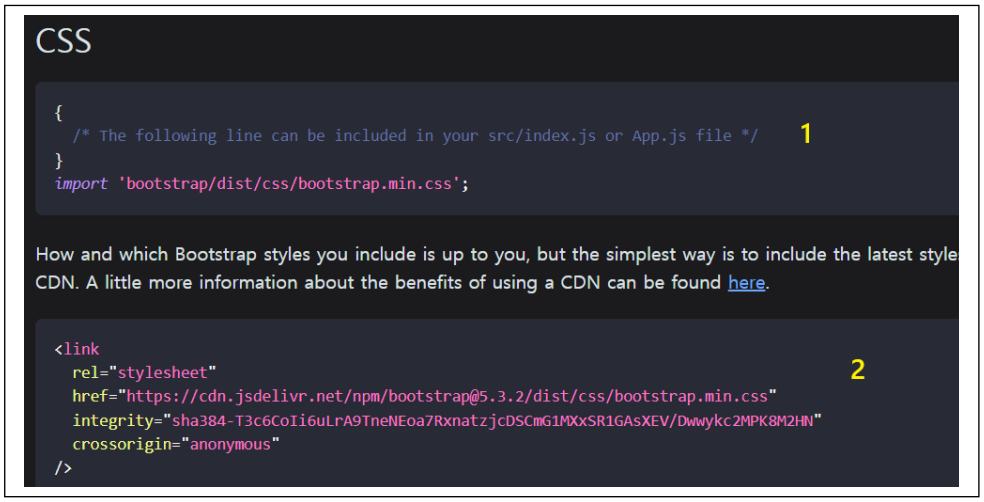
```
t > src > JS App.js > App
import {Route, Routes} from 'react-router-dom';
import { Container } from 'react-bootstrap';
import 'bootstrap/dist/css/bootstrap.min.css';
function App() {
  return (
    <div>
      <Container>
        <Routes>
          <Route path="/"  />
        </Routes>
      </Container>
    </div>
  );
}
```

8253jang@daum.net
장희정

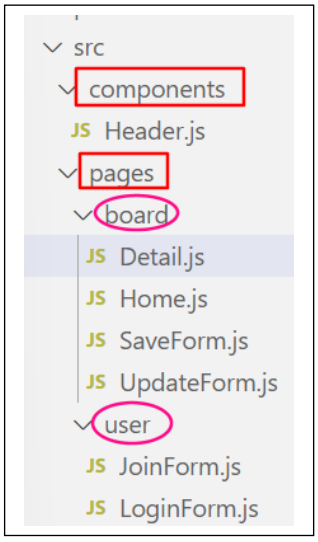
Bootstrap css 연결

구글 < react bootstrap < Get Started
https://react-bootstrap.netlify.app/

```
import 'bootstrap/dist/css/bootstrap.min.css';
```
-->index.js에 저 부분을 걸어준다.



components , pages 만들기



src< components, pages 폴더를 만들고

src< components< Header.js 생성

src< pages< board, user 폴더 생성한 후 js파일 생성

8253jang@daum.net
장희정

```
componets <
   BoardItem : 게시물 한개
   Header : 메뉴

pages >
   board >
          Detail : 상세보기
          Home : 전체목록
          SaveForm : 저장폼
          UpdateForm : 수정폼


   user >
        JoinForm :가입폼
        LoginForm : 로그인폼
```

React Bootstrap < Component<NavBar중에서 하나를 선택해서 모든 코드를 Header.js에 붙인다.

그리고, 메뉴를 아래와 같이 구성한다.


```
function App() {
  return (
    <div>
      <Header />
```

App.js에서 Header를 추가

Header.jsx 파일

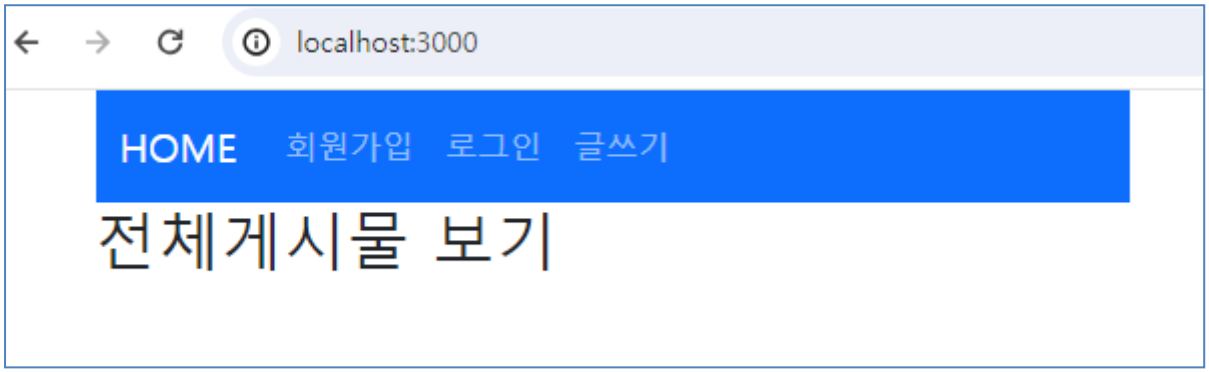
```
import Container from 'react-bootstrap/Container';
import Nav from 'react-bootstrap/Nav';
import Navbar from 'react-bootstrap/Navbar';
import { Link } from 'react-router-dom';

const Header = () => {
  return (
    <Navbar bg="primary" data-bs-theme="dark">
      <Container>
      <Nav className="me-auto">
        <Link to="/" className="navbar-brand">HOME</Link>
         <Link to="joinForm" className="nav-link">회원가입</Link>
         <Link to="/loginForm" className="nav-link">로그인</Link>
         <Link to="/saveForm" className="nav-link">글쓰기</Link>
      </Nav>
      </Container>
    </Navbar>
  );
}

export default Header;
```

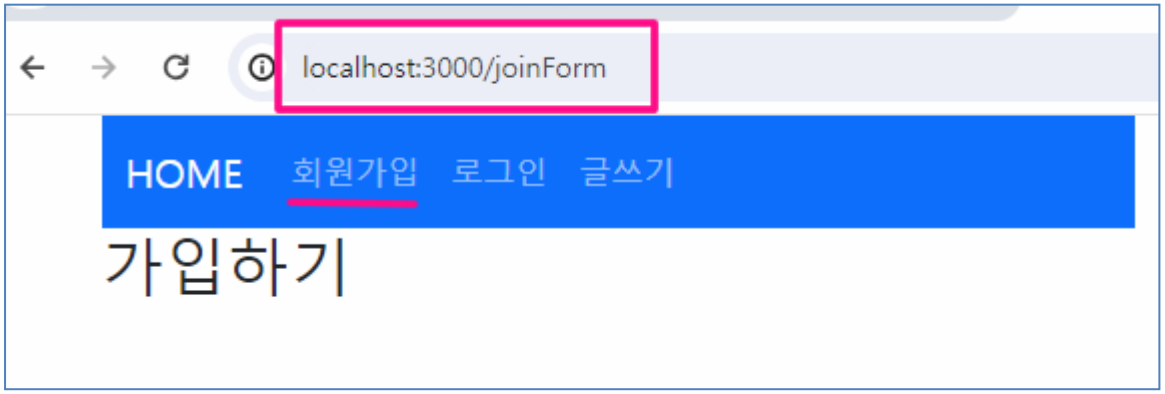
App.jsx

```
import './App.css';
import { Route, Routes } from 'react-router-dom';
import {Container} from 'react-bootstrap';
import Header from './components/Header';
import Home from './pages/board/Home';
import Detail from './pages/board/Detail';
import LoginForm from './pages/user/LoginForm';
import JoinForm from './pages/user/JoinForm';
import UpdateForm from './pages/board/UpdateForm';
import SaveForm from './pages/board/SaveForm';

function App() {
  return (
```

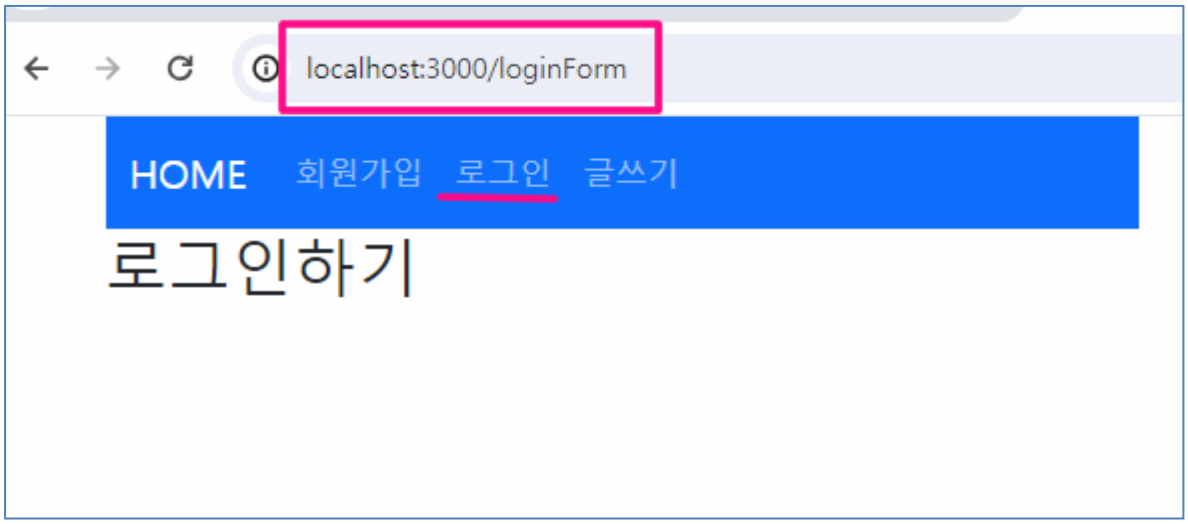
```
    <div>
      <Container>
        <Header/>
     <Routes>
       <Route path='/' element={<Home/>}/>
       <Route path='/saveForm' element={<SaveForm/>}/>
       <Route path='/board/:id' element={<Detail/>}/>
       <Route path='/updateForm/:id' element={<UpdateForm/>}/>
       <Route path='/loginForm' element={<LoginForm/>}/>
       <Route path='/joinForm' element={<JoinForm/>}/>
     </Routes>
     </Container>
    </div>
  );
}

export default App;
```

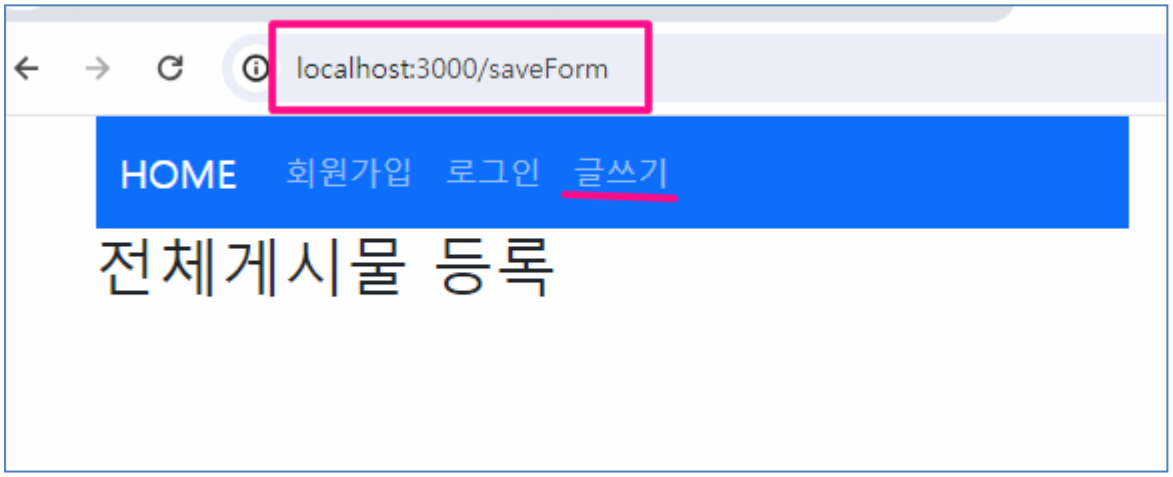
실행결과





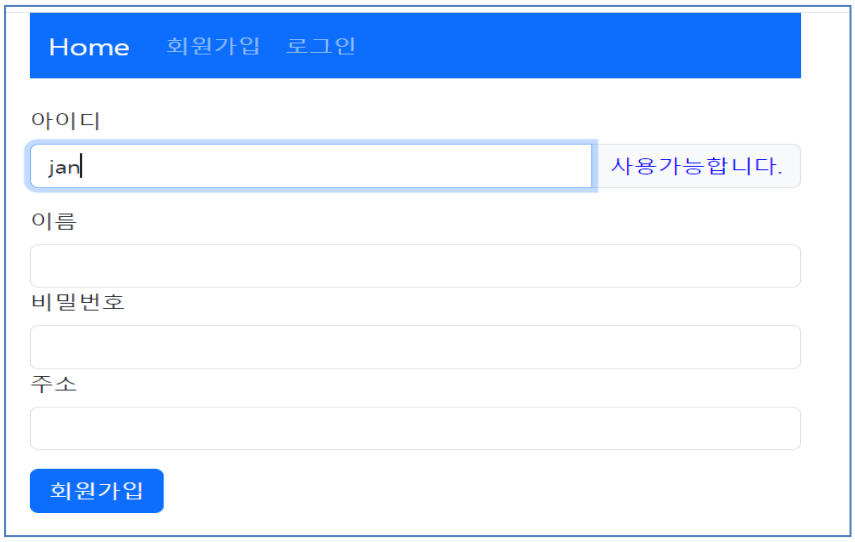
8253jang@daum.net
장희정

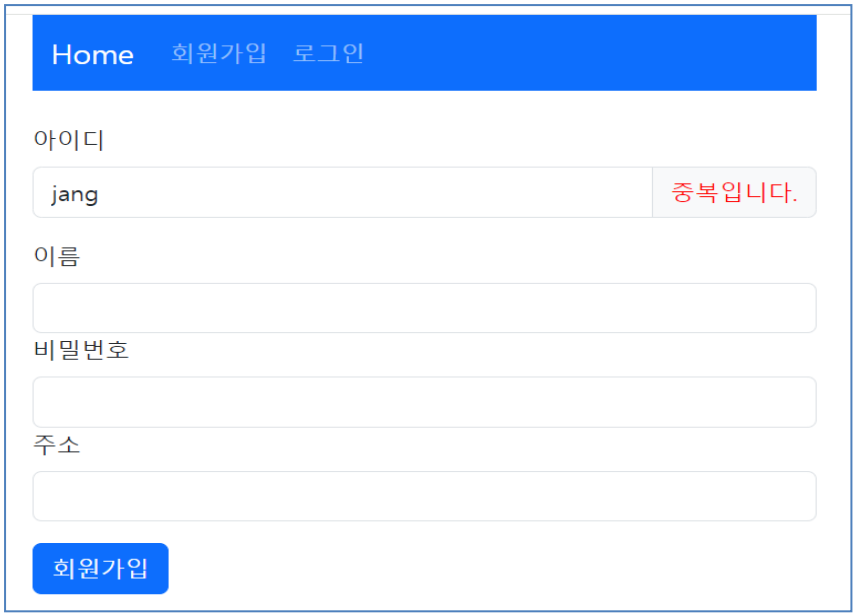




회원가입(id를 입력하면 값이 변경될 때마다 중복여부 체크)

8253jang@daum.net
장희정





☞ **가입 폼 디자인**

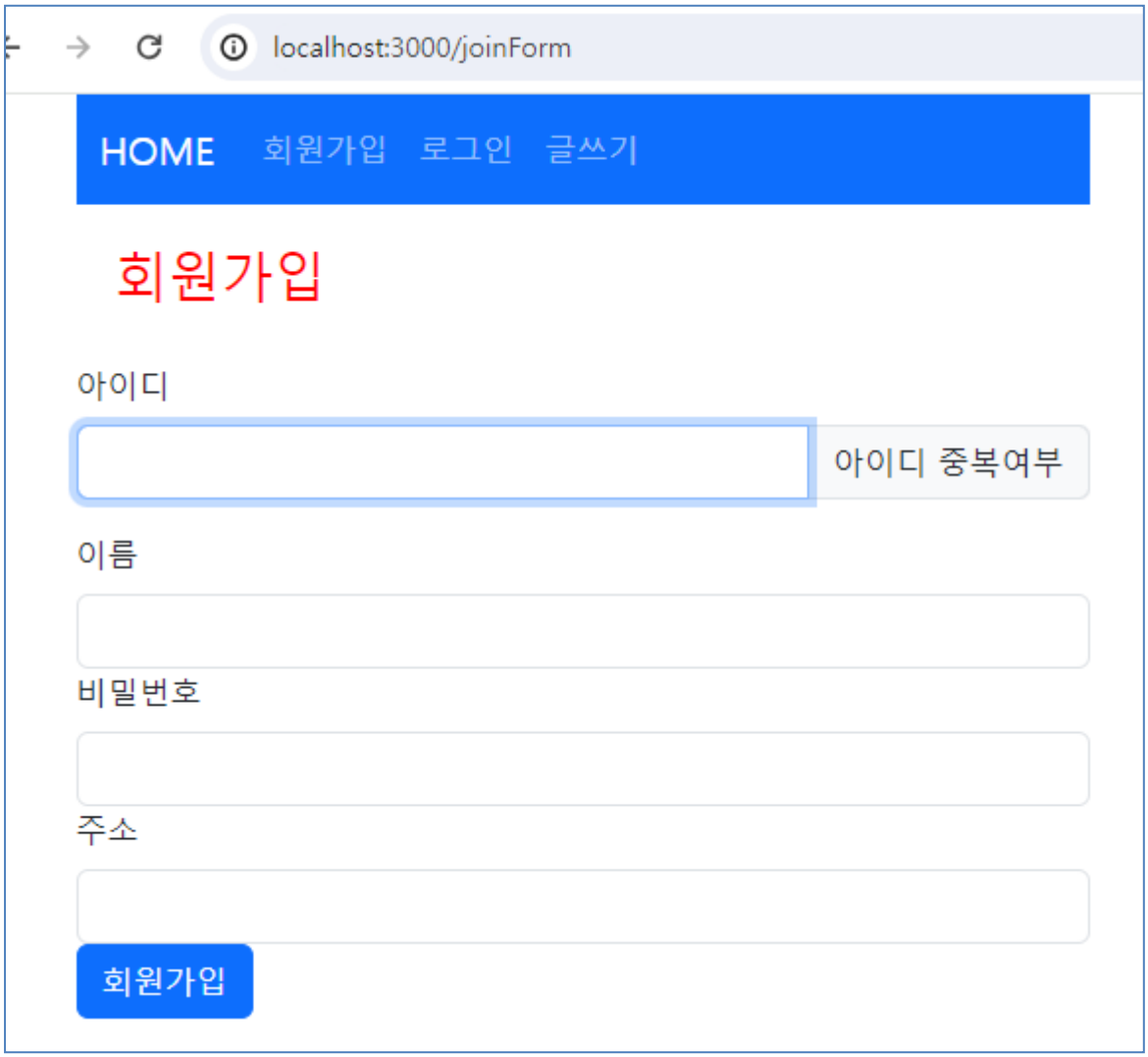
https://react-bootstrap.netlify.app/docs/forms/form-text

```
import { Button, InputGroup } from 'react-bootstrap';
import Form from 'react-bootstrap/Form';

const JoinForm = ()=> {
   //각 text 박스에 값이 변경되었을 때
   const changeValue =()=>{

   }
```

```
 //가입하기
 const submitJoin = ()=>{

 }

return (
  <>
  <h2 style={{padding:"20px",color:"red"}}>회원가입</h2>
  <Form>
    <Form.Label htmlFor="id">아이디</Form.Label>
    <InputGroup className="mb-3">
    <Form.Control
      type="text"
      id="id"
      name="id"
      onChange={changeValue}
    />
     <InputGroup.Text  >아이디 중복여부</InputGroup.Text>
    </InputGroup>

     <Form.Label htmlFor="name">이름</Form.Label>
      <Form.Control
          type="text"
          id="name"
          name="name"
          onChange={changeValue}
      />

     <Form.Label htmlFor="pwd">비밀번호</Form.Label>
    <Form.Control
      type="password"
      id="pwd"
      name="pwd"
      onChange={changeValue}
    />
     <Form.Label htmlFor="address">주소</Form.Label>
    <Form.Control
      type="text"
      id="address"
```

```
          name="address"
          onChange={changeValue}
        />
        <p>
        <Button variant="primary"
onClick={submitJoin}>회원가입</Button>
        </p>
      </Form>
      </>
    );
}

export default JoinForm;
```

실행화면



8253jang@daum.net
장희정

☞ 가입정보에 대한 상태관리를 위해서 useState를 선언한다.
회원가입의 속성들을 하나의 객체로 관리한다.

```
const [member, setMember] = useState({
    id :'' ,
    name:'',
    pwd:'',
    address:''
  });
```

☞ input 요소의 value가 변경되었을 때 setMember를 이용하여 변경한다.

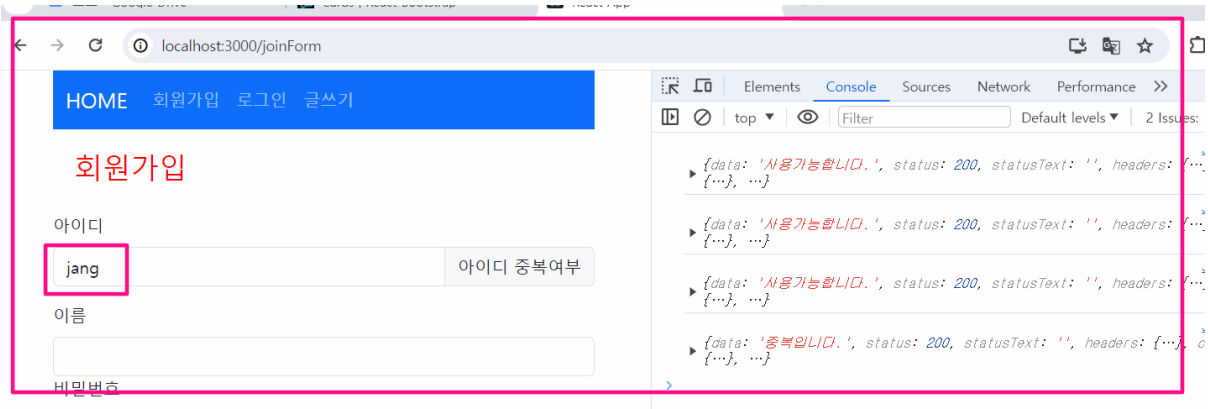
```
 const changeValue = (e)=>{
      //console.log(e.target.name);
     //console.log(e.target.value);
      setMember({
        ...member,
        [e.target.name]: e.target.value}
        )
}
```

☞ 아이디 중복체크 하기
Axios를 이용하여 backEnd에 요청을 보내고 그 결과를 받는다.

```
 if(e.target.name==="id" && e.target.value!==""){
       axios({
         method:"GET",
         url : "http://localhost:9000/members/"+e.target.value,
        // data : {"id" : e.target.value},
       })
       .then((res)=>{
          console.log(res);
       })
       .catch((err)=>{
        //실패
         let errMessage = err.response.data.type +"\n";
         errMessage +=  err.response.data.title +"\n";
         errMessage +=  err.response.data.detail +"\n";
```

```
            errMessage +=  err.response.data.status +"\n";
            errMessage +=  err.response.data.instance +"\n";
            errMessage +=  err.response.data.timestamp;
            alert(errMessage);
        });
    }
```

.



☞ **중복체크 결과를 아이디 text박스 옆에 출력한다.**

```
// 중복체크 결과 값을 저장 할  idCheckResult
const [idCheckResult , setIdCheckResult] = useState("");


// 아이디 중복여부에 따른 css 를 적용하기 위해 상태 변수
const [isCheckResult , setIsCheckResult] = useState(false);
```

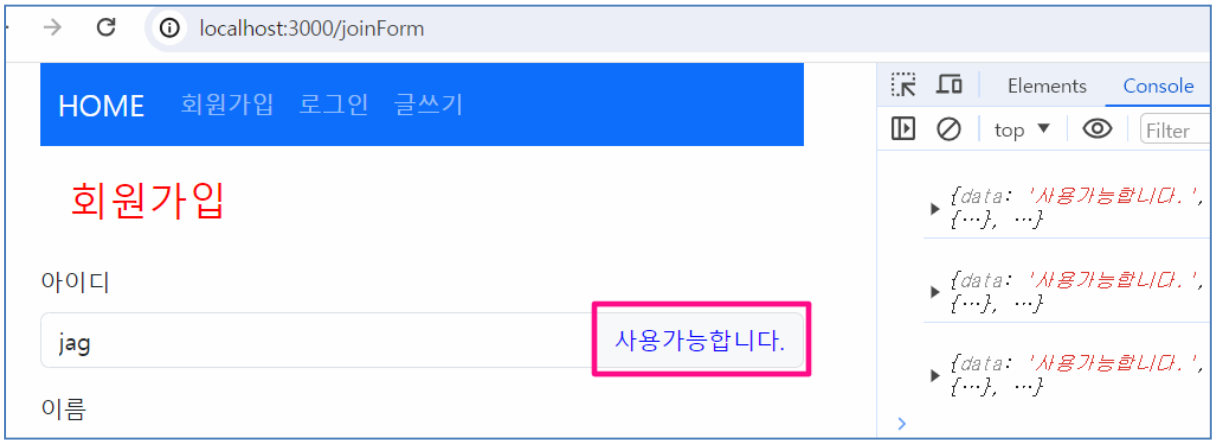
☞ **axios 결과를 받아서** setIdCheckResult , setIsCheckResult 값 변경

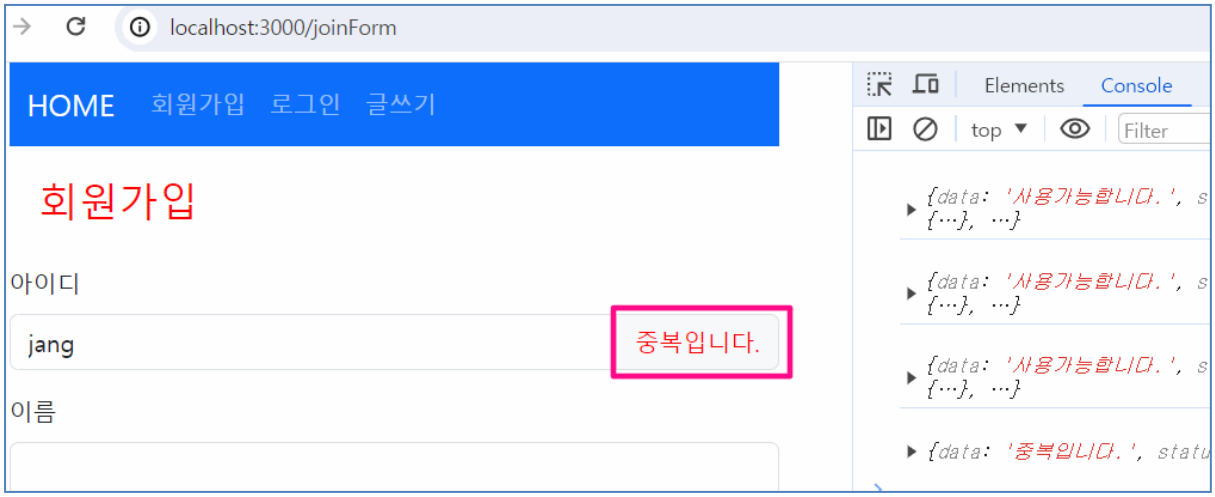
```
.then((res)=>{
    setIdCheckResult(res.data);

   res.data==="중복입니다." ?
           setIsCheckResult(true) : setIsCheckResult(false);
 })
```

☞ **아이디 text박스 옆에 결과를 출력한다.**

```
<InputGroup.Text
   style={ isCheckResult ? {color: "red"} : {color: "blue" } } >

     {idCheckResult}

</InputGroup.Text>
```

실행결과





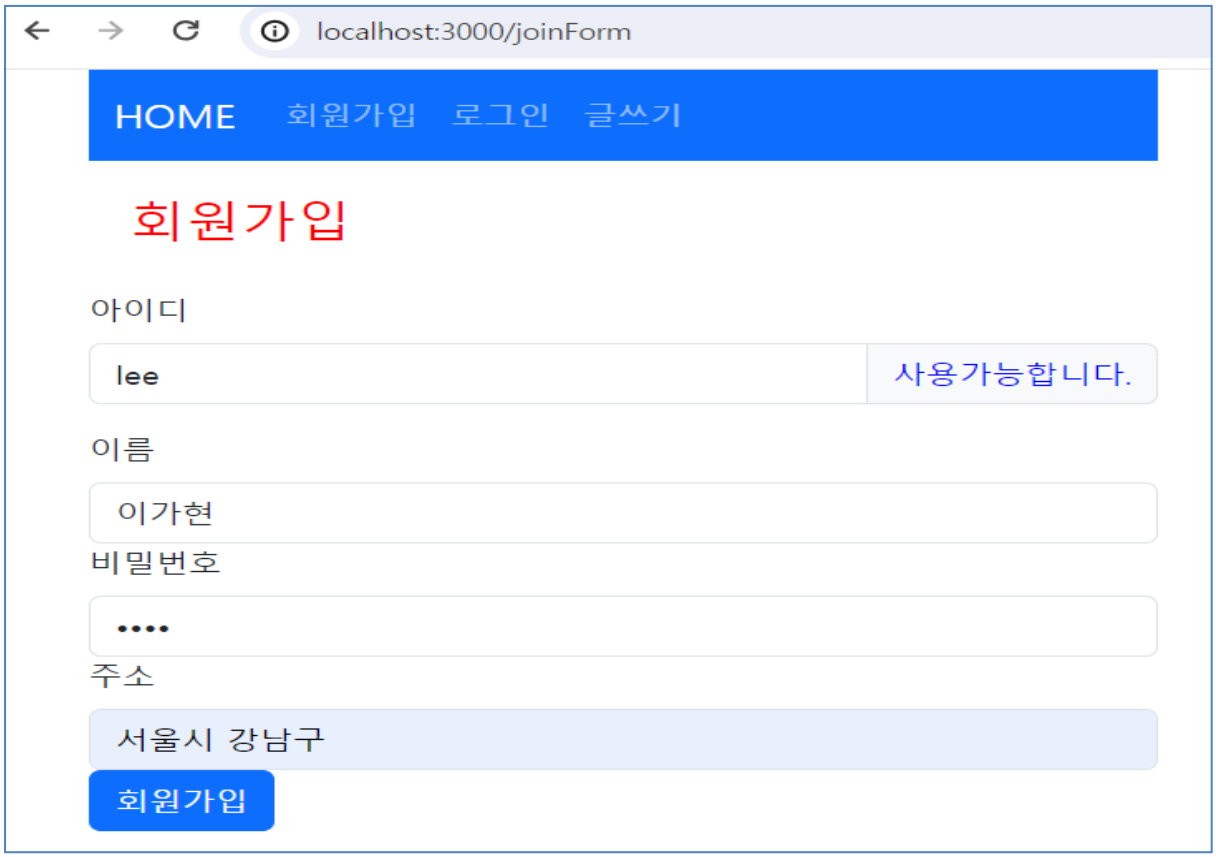
☞ **회원가입 클릭**

: 가입을 성공하면 "/" Home으로 이동한다.

: `import { useNavigate } from 'react-router-dom';`

```
const navigator = useNavigate();

  const submitJoin = (e)=>{
    axios({
      method:"POST",
      url : "http://localhost:9000/members",
      data : member,
    })
    .then((res)=>{
        console.log(res);
        navigator("/")
    })
    .catch((err)=>{
          console.log(err)
          let errMessage = err.response.data.type +"\n";
          errMessage +=  err.response.data.title +"\n";
          errMessage +=  err.response.data.detail +"\n";
          errMessage +=  err.response.data.status +"\n";
          errMessage +=  err.response.data.instance +"\n";
          errMessage +=  err.response.data.timestamp;

          alert(errMessage);
    });
  }
```

실행결과



로그인 하기

: backend spring security를 이용하여 인증을 진행 할 예정이다.
때문에 입력한 **아이디와 비밀번호는 Controller가 아닌 Spring security의 filter에 전달된다.**
Security는
Parameter 정보 -> 아이디 : username , 비번 : password
parameter정보를 json이 아닌 Formdataf로 전송해야한다.
요청주소 -> /login

☞ **로그인 폼 디자인 – LoginForm.jsx**

```
import React from 'react';
import { Button } from 'react-bootstrap';
import Form from 'react-bootstrap/Form';

const LoginForm = () => {
    const changeValue = (e)=>{
```

```
    }//

    const submitLogin = (e)=>{

    }

    return (
 <div>
  <h3 style={{padding:"10px"}}>로그인하기 </h3>
   <Form onSubmit={submitLogin}>
      <Form.Label htmlFor="username">아이디</Form.Label>
      <Form.Control
        type="text"
        id="username"
        name="username"
        onChange={changeValue}

      />

       <Form.Label htmlFor="password">비밀번호</Form.Label>
      <Form.Control
        type="password"
        id="password"
        name="password"
        onChange={changeValue}
      />

      <p/>
      <Button variant='primary' type='submit'>로그인</Button>
      <p/>

      </Form>
    </div>
    );
};

export default LoginForm;
```

8253jang@daum.net
장희정

☞ **로그인 기능구현**

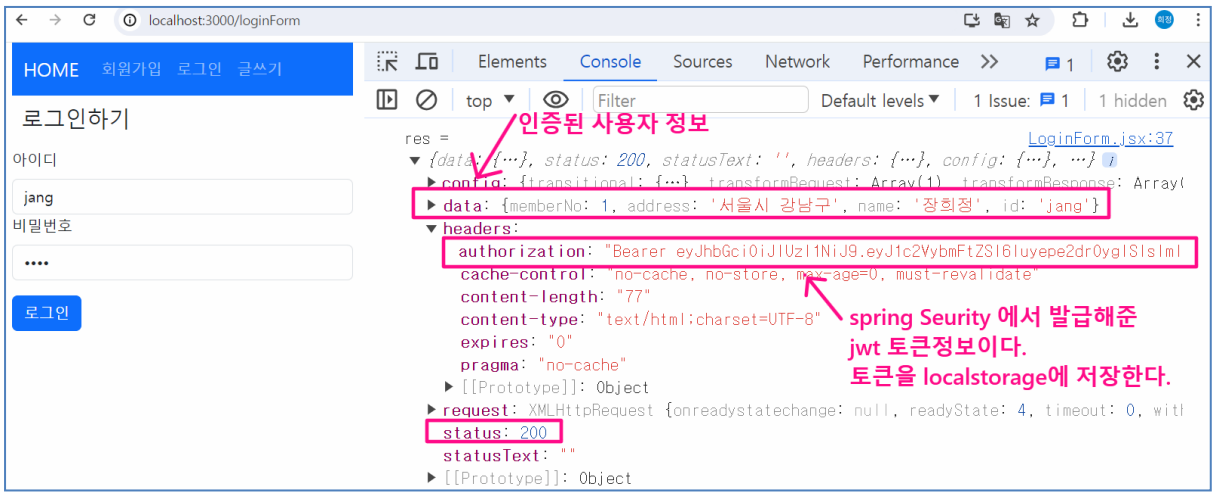
```
 // 인증에 필요한 username, password 상태관리를 위한 useState
const [member, setMember] = useState({
    username :'' ,
    password:'',
  });


// input 에 값이 입력될 때 상태 값 수정
const changeValue = (e)=>{
    setMember({
      ...member,
        [e.target.name]: e.target.value}
    )
  }//



const navigator = useNavigate();

//로그인 버튼을 클릭했을 때 axios
  const submitLogin = (e)=>{
     e.preventDefault();//submit 이벤트 막음

    let formData = new FormData(); //폼전송으로 보내기위한 작업
    formData.append("username", member.username);
    formData.append("password", member.password);

    axios({
      method:"POST",
      url : "http://localhost:9000/login",
      data : formData,
    })
    .then((res)=>{
      console.log("res = " , res)
    })
    .catch((err)=>{
        alert("정보를 다시 확인해주세요.");
        setMember({
          username :'' ,
```

```
        password:''})
    });
}
```

: username, password 속성에 value 추가한다.

Ex) value={member.username}
    value={member.password}

실행결과



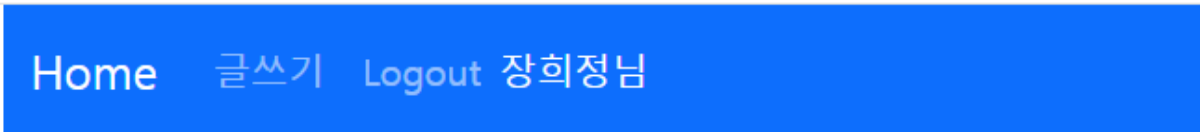
☞ **로그인이 성공하면 인증된 사용자의 정보를 localStorage에 저장한다.**

```
localStorage.setItem("memberNo", res.data.memberNo);
localStorage.setItem("id", res.data.id);
localStorage.setItem("name", res.data.name);
localStorage.setItem("address", res.data.address);

localStorage.setItem("Authorization", res.headers.authorization);
```

☞ 인증 여부에 따른 Header 메뉴를 변경해보자.

-인증되지 않은 경우

-인증된 경우



먼저, App.js문서에서

1) 로그인 여부를 체크 할 상태변수 isLoggedIn 과
상황에 따라 상태변수의 값을 변경 할 함수=handleLoggedChange를 만든다.

2) index.js 요청되면 -> App.js 로딩 -> useEffect()를 이용하여 localStorage에 인증된 정보가 있는지 확인하여 상태변수 isLoogedIn을 변경한다.(있으면 true, 없으면 false)

3) 로그인 여부를 체크 할 상태변수 isLoggedIn 과
상황에 따라 상태변수의 값을 변경 할 함수=handleLoggedChange를 만든다.

4) createContext()를 이용하여 하위 컴포넌트들이 isLoggedIn , handleLoggedChaange를 공유하도록 한다.

☞ **App.js 일부분**

```
/*useContext 를 이용해서 하위 컴포넌트들이 데이터 공유하기*/
export const LogingedContext = createContext();


function App() {
  const [isLoggedIn, setIsLoggedIn] = useState(false);

    //컴포넌트가 mount or update 될때 로그인 여부에 따른 상태값 변경
  useEffect(()=>{
        localStorage.getItem("id")!=null ?
          setIsLoggedIn(true) : setIsLoggedIn(false);

      console.log("App useEffect isLoggeedIn = ", isLoggedIn)
    });
```

```
   /*
     로그인(LoginForm.jsx) or 로그아웃(Header.jsx) 될 때 로그인여부
상태값을  변경할 이벤트
     handleLoggedChange 와 isLoggedIn 를 사용해야 하는 컴포넌트들이
여럿이기에
     createContex 를 이용하여 서로 공유할수 있도록 한다.
   */
  const handleLoggedChange = (isLoggedIn)=>{
    setIsLoggedIn(isLoggedIn);
  }

  return (
    <LogingedContext.Provider
         value={ {isLoggedIn:isLoggedIn ,
                onLoggedChange:handleLoggedChange } }>

      <div>
        <Container>
        <Header/>
          <Routes>
            <Route path='/' element={<Home/>}/>
            <Route path='/saveForm' element={<SaveForm/>}/>
            <Route path='/boards/:id' element={<Detail/>}/>
            <Route path='/updateForm/:id' element={<UpdateForm/>}/>
            <Route path='/loginForm' element={<LoginForm  />}/>
            <Route path='/joinForm' element={<JoinForm/>}/>
          </Routes>
        </Container>
      </div>

   </LogingedContext.Provider>
  );
}

export default App;
```

☞ **LoginForm.jsx 수정**

: useContext를 이용하여 전달된 Context를 사용한다

```
let logingedCon =useContext(LogingedContext);
```

☞로그인이 성공했을 때 콜백함수에서 인증여부 상태변수의 값을 변경한다.

```
  logingedCon.onLoggedChange(true);
```

그리고, useNavigate()를 이용하여 "/" Home으로 이동한다.

```
//페이지 이동하는 방법
  const navigator = useNavigate();

  //로그인 성공했을 때 콜백함수안에서
  navigator("/")
```

☞Header.jsx 문서에서 useCcontext 선언

```
let logingedCon = useContext(LogingedContext);
```

☞ Header.jsx 문서에서 로그아웃 되었을 때 이벤트

```
  const navigator = useNavigate();

  const logoutCheck = ()=>{
     localStorage.removeItem("memberNo");
     localStorage.removeItem("id");
     localStorage.removeItem("name");
     localStorage.removeItem("address");
     localStorage.removeItem("Authorization");

      logingedCon.onLoggedChange(false);

      navigator("/");
  }
```

☞ **Header.jsx 문서에서 인증여부에 따른 메뉴 출력**

```
<Navbar bg="primary" data-bs-theme="dark">
 <Container>
   <Link to="/" className="navbar-brand">Home</Link>

  <Nav className="me-auto">
     { logingedCon.isLoggedIn &&
    <Link to="/saveForm" className="nav-link">글쓰기</Link>}

    {!logingedCon.isLoggedIn &&
      <Link to="/joinForm" className="nav-link">회원가입</Link>}

    { logingedCon.isLoggedIn ?
      <Button onClick={logoutCheck} className="nav-link" >
        Logout
     </Button>
          :
   (<Link to="/loginForm" className="nav-link" >로그인</Link>)
   }

      
  {
    logingedCon.isLoggedIn &&
         <span>{localStorage.getItem("name")}님</span>
  }

     </Nav>
   </Container>
  </Navbar>
```

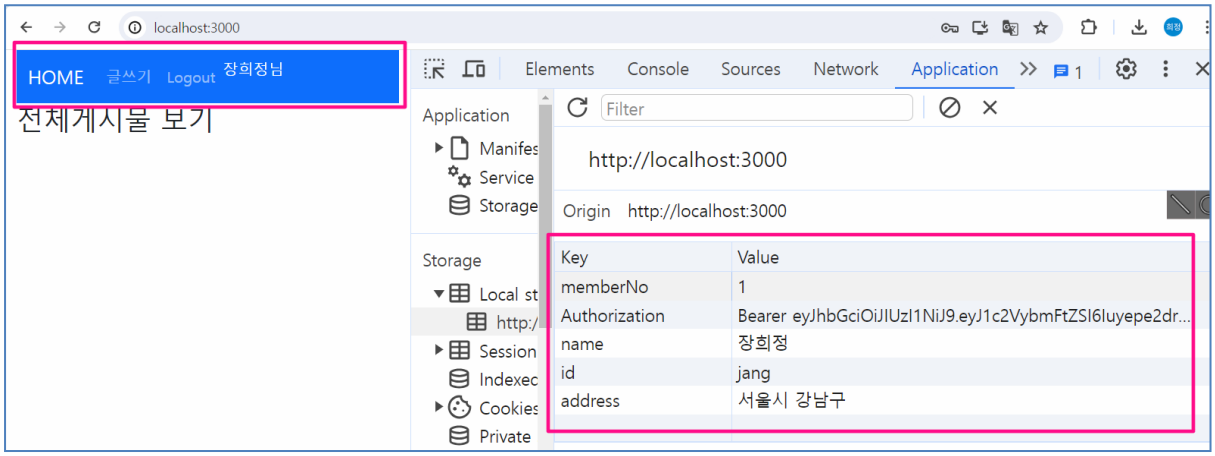
src > style.css

```
#root > div > div > nav > div > div {
  align-items: center;
  vertical-align: middle;
  color: white;
  font-size: 16px;
}
```

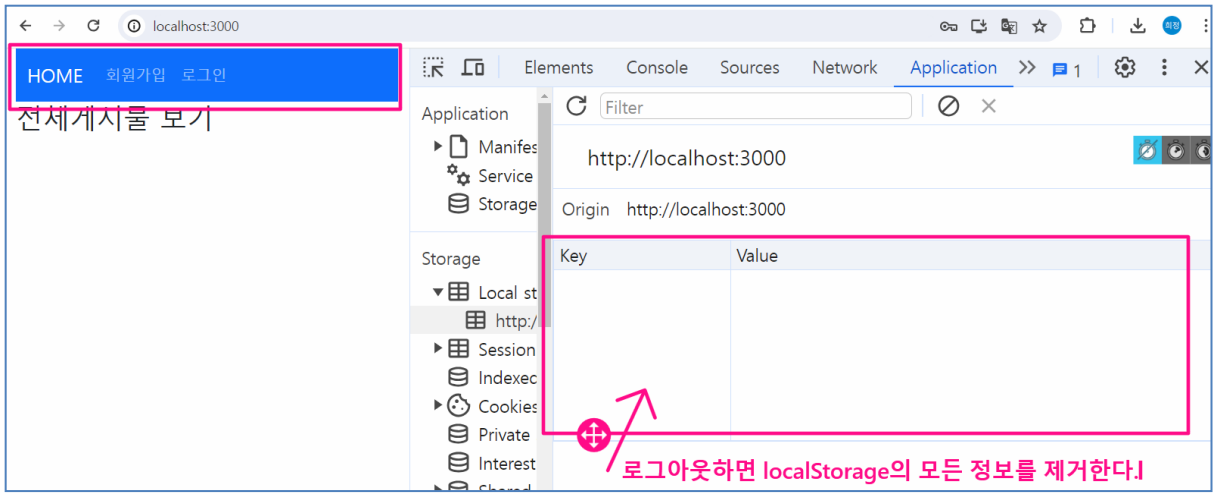
8253jang@daum.net
장희정

실행결과

로그인성공 했을 때



로그아웃했을 때



☞ 전체 게시물 보기

디자인 컴포넌트를 하나 생성 - BoardItem.jsx

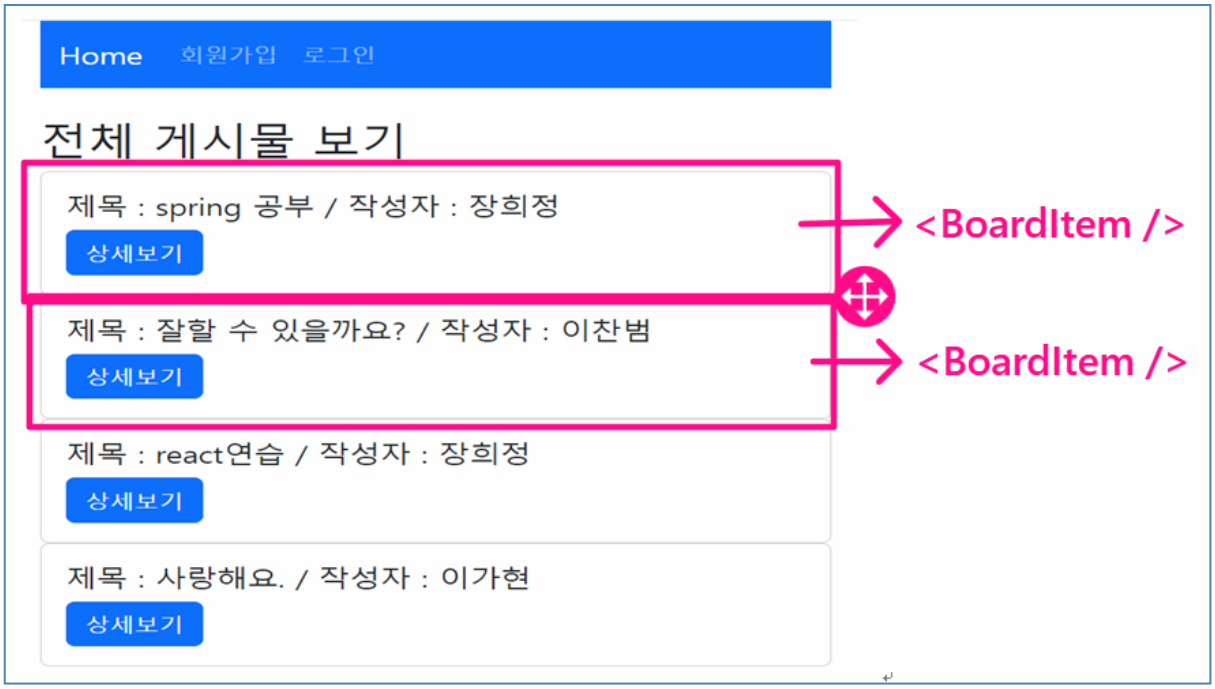
재사용되는 아이템(디자인 포함)은 컴포넌트에 따로 작성하는 것이 좋다.

이렇듯 재 사용할 수 있는 것들은 다 컴포넌트로 만들어놓자

React boostrap 사이트에서 components > Cards클릭

https://react-bootstrap.netlify.app/docs/components/cards

예시)



src> components > BoardItem.jsx

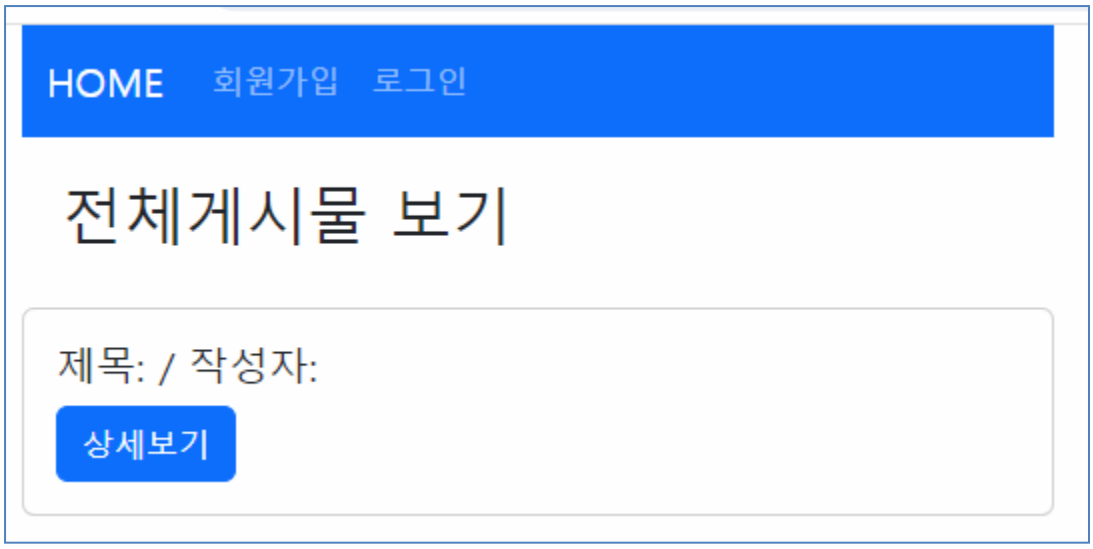
```
import Card from 'react-bootstrap/Card';
import { Link } from 'react-router-dom';

function BoardItem() {
  return (
    <>
    <Card>
      <Card.Body>
        <Card.Title>제목:  / 작성자: </Card.Title>
        <Link to={"/boards/1"} className="btn btn-primary" >
            상세보기
        </Link>
      </Card.Body>
    </Card>
    </>
  );
}

export default BoardItem;
```

8253jang@daum.net
장희정

src > pages > Home.jsx

```
import React from 'react';
import BoardItem from '../../components/BoardItem';

const Home = () => {
    return (
        <div>
            <h1 style={{padding:"20px"}}>전체게시물 보기 </h1>
            <BoardItem/>
        </div>
    );
};

export default Home;
```

실행결과



--> BoardItem은 결국 글 목록이 있는 만큼 반복 되어져야 한다.
그러기 위해서는 DB 테이블에 있는 레코드를 받아와야 한다.
Home.js 에서 React Hook의 종류인 useEffect를 이용하여 DB에 들어있는 데이터를 비동기통신으로(axios) 받아온다

([ ] 안 넣는다면?) => 최초실행+상태값변경 때마다 실행됨

8253jang@daum.net
장희정

☞ axios를 이용하여 모든 게시물 조회

- **Home.jsx에 추가되는 import**

```
import React, { useEffect, useState } from 'react';
import BoardItem from '../../components/BoardItem';
import axios from 'axios';
```

-Home.jsx 마운트(최초로딩) 되었을 때 만 backend에서 데이터 가져오기

spring에서 spring security + jwt로 구현되어 있어 headers에 Authorization 에 해당하는 jwt 토큰을 함께 요청한다.

```
//DB 목록을 저장해서 관리 할 useState
  const [boards , setBoards] = useState([]);


 //컴포넌트가 마운트 되었을 때
     useEffect(()=>{
         axios
         .get("http://localhost:9000/boards", {
           headers: {
              Authorization: localStorage.getItem("Authorization"),
           },})
         .then((res)=>{
            setBoards(res.data);
         })
         .catch((err)=>{
           let errMessage = err.response.data.type +"\n";
           errMessage +=  err.response.data.title +"\n";
           errMessage +=  err.response.data.detail +"\n";
           errMessage +=  err.response.data.status +"\n";
           errMessage +=  err.response.data.instance +"\n";
           errMessage +=  err.response.data.timestamp;

           alert(errMessage);
         });

       }, [] );//( [ ] 안 넣는다면? ) => 최초실행+상태값변경 때마다 실행됨
```

8253jang@daum.net
장희정

-Home.jsx DB에서 조회된 boards의 정보를 map을 이용하여 출력한다.

BoardItem 컴포넌트에 props를 이용하여 board의 정보를 전달한다.

```
return (
   <div>
     <h1 style={{padding:"20px"}}>전체게시물 보기 </h1>

   {
  boards.map( (board)=> <BoardItem  key={board.id} board={board}/> )
   }

  </div>

 );
```

-BoardItem.jsx 에서 props로 전달된 데이터를 받아서 출력

```
2  import Card from 'react-bootstrap/Card';
3  import { Link } from 'react-router-dom';
4  function BoardItem(props) {→props정보를 받는다.
5    const {id, title, content, regDate , member} = props.board;
6    return (
7      <>
8      <Card>
9        <Card.Body>
10       <Card.Title>제목 : {title}  /  작성자 : {member.name}  </Card.Title>
11           <Link to={"/boards/"+id} className="btn btn-primary" >
12            상세보기
13           </Link>
14       </Card.Body>
15     </Card>
16     </>
17   );
18 }
```

→스프레드연산자를 이용하여 전달된 board를 변수에 저장

출력결과

8253jang@daum.net
장희정

☞ 동기 + 비동기 작업 흐름

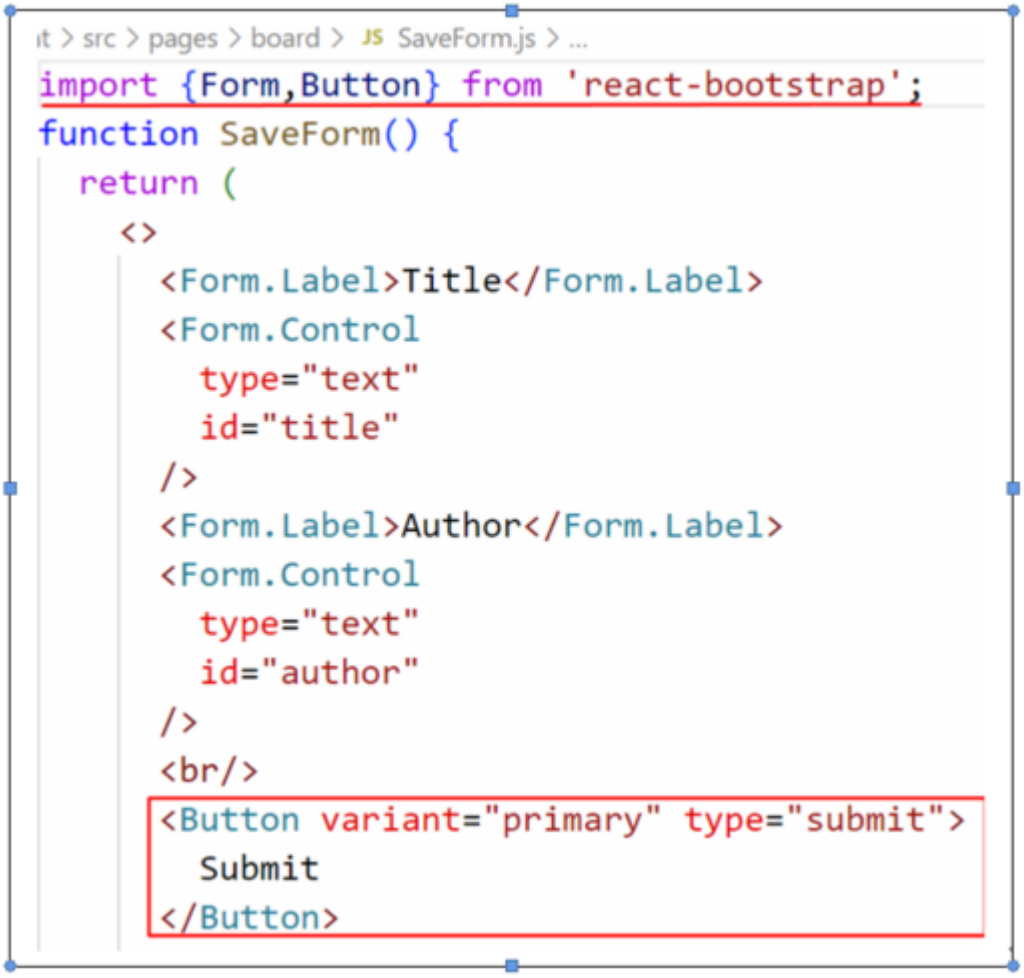
☞ 글 쓰기

react bootstrap < component < Form < Form Text 소스를 긁어온다.

이 부분은 재 사용할 일이 없기 때문에 components에 생성하지 않고 바로 SaveForm에 적용한다.

https://react-bootstrap.netlify.app/docs/forms/form-text

8253jang@daum.net
장희정



SaveForm.jsx파일

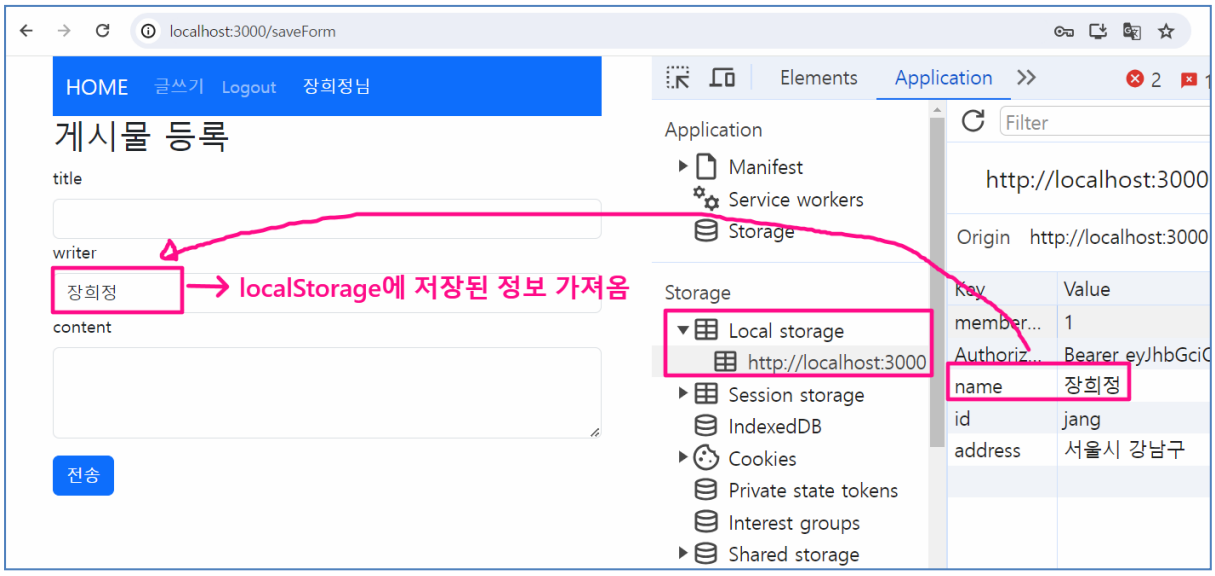
: 작성자 부분은 localStorage에 저장된 정보를 가져온다.

```
import { Button } from 'react-bootstrap';
import Form from 'react-bootstrap/Form';


const SaveForm = () => {
    //input 에 값이 변경될 때
    const changeValue = (e)=>{

    }

    //등록하기 클릭
    const submitBoard = (e)=>{
```

```
    }

    return (
        <div>
           <h1>게시물 등록 </h1>
           <Form onSubmit={submitBoard}>
      <Form.Label htmlFor="title">title</Form.Label>
      <Form.Control
        type="text"
        id="title"
        name="title"
        onChange={changeValue}
      />
      <Form.Label htmlFor="author">writer</Form.Label>
      <Form.Control
        type="text"
        id="name"
        name="name"
        readOnly
        value={localStorage.getItem("name")}
      />
      <Form.Label>content</Form.Label>
        <Form.Control as="textarea" rows={3} name="content"
         id="content" onChange={changeValue}/>
      <p/>

      <Button variant='primary' type='submit'>전송</Button>
      <p/>
      </Form>
        </div>
    );
};

export default SaveForm;
```

실행화면

8253jang@daum.net
장희정



- **text박스에 값이 입력될 때 state를 변경하기**
  : 작성자의 pk는 localStorage에서 조회

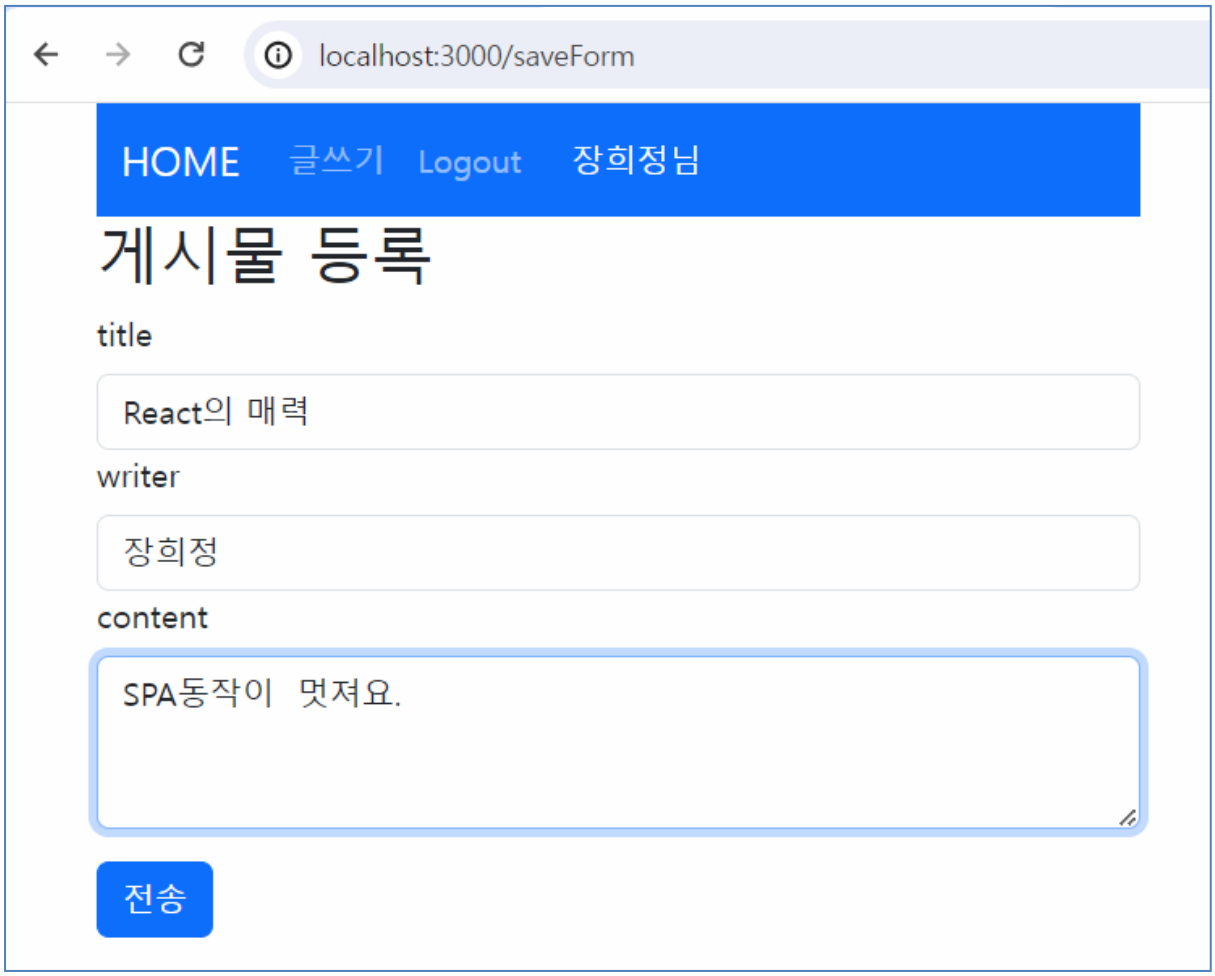
```
const [board, setBoard] = useState({
    title :'' ,
    content:'',
    memberNo:localStorage.getItem("memberNo"),
  });


  const changeValue = (e)=>{
      setBoard({
        ...board,
        [e.target.name]: e.target.value}
        )
  }
```

- **모든 값들을 입력 한 후 axios로 비동기 통신**
  : 등록이 성공하면 "/" 로 이동
  : `import { useNavigate } from "react-router-dom";`
  : `import axios from "axios";`

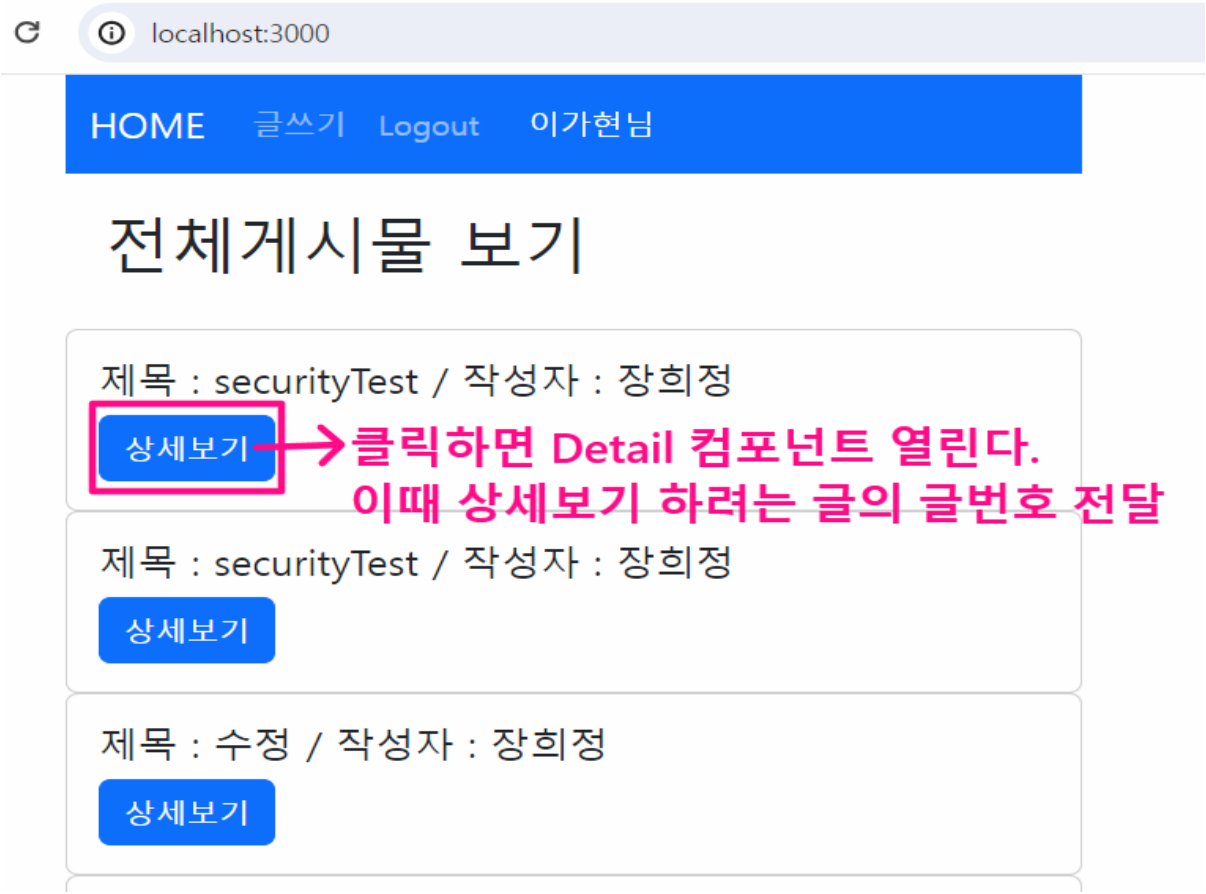
```
//페이지 이동하는 방법
  const navigator = useNavigate();

  const submitBoard = (e)=>{

    e.preventDefault();

    axios({
      method:"POST",
      url : "http://localhost:9000/boards/board",
      data : board,
      headers: {
       Authorization: localStorage.getItem("Authorization"),
       }
    })
    .then((res)=>{
        console.log(res);
        navigator("/")
    })
    .catch((err)=>{
          let errMessage = err.response.data.type +"\n";
          errMessage +=  err.response.data.title +"\n";
          errMessage +=  err.response.data.detail +"\n";
          errMessage +=  err.response.data.status +"\n";
          errMessage +=  err.response.data.instance +"\n";
          errMessage +=  err.response.data.timestamp;

          alert(errMessage);
   });
  }
```

실행화면

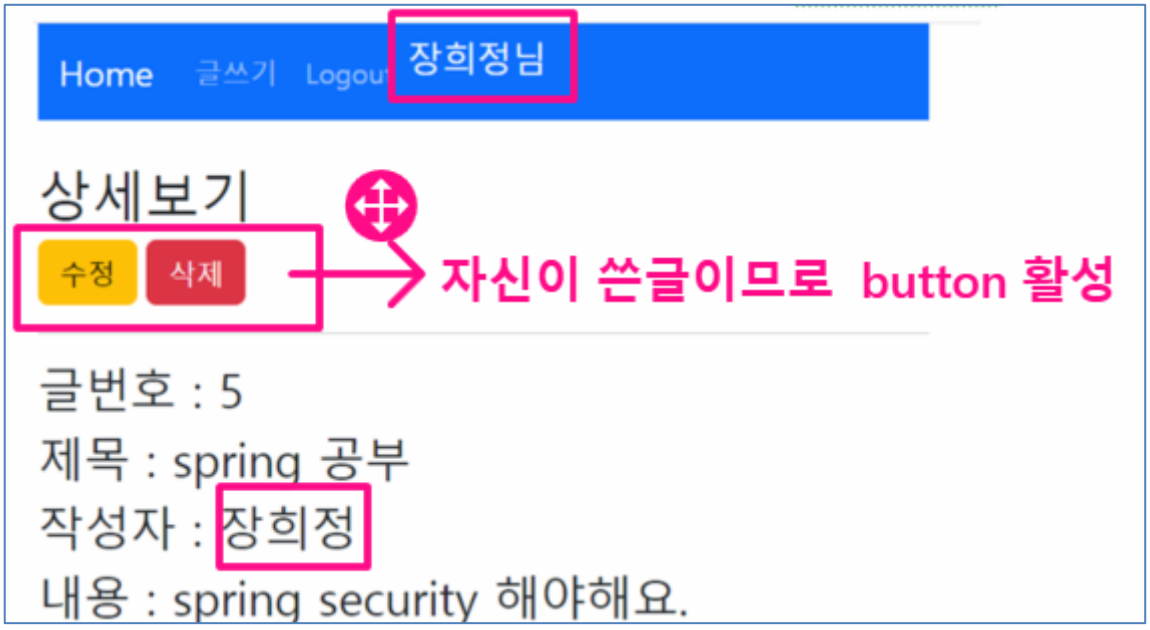
8253jang@daum.net
장희정



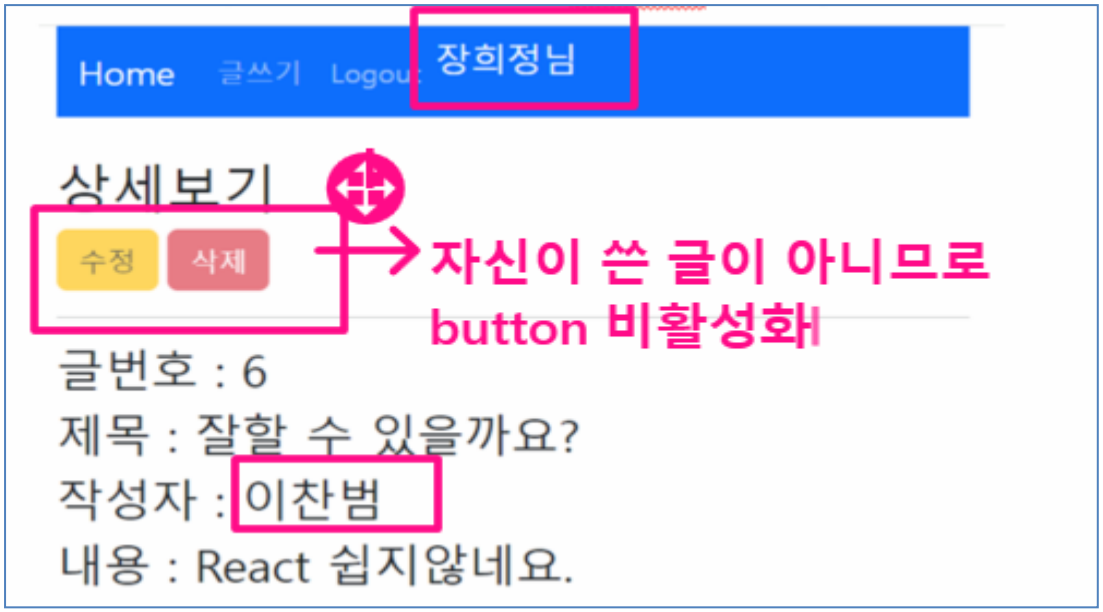
☞ **상세보기**



- **Detail.jsx 자신이 쓴 글 클릭하면, 수정, 삭제 버튼 활성화**



- Detail.jsx 자신이 쓴 글이 이나면 수정, 삭제 버튼 비 활성화



Home.jsx에서

```
<Link to={"/boards/"+id} className="btn btn-primary" >
 상세보기
</Link>
```

☞ **Detail.jsx 의 컴포넌트가 호출되면**

- 전달된 파라미터를 **useParams**로 받는다.

- 게시물에 대한 정보를 저장할 **useState**를 선언한다.

- **useEffect 함수에서** 최초의 마운트 되었을 때 axios를 이용하여 서버에 게시물글번호를 전송한 후 게시물의 정보를 응답 받는다.

- 응답결과가 error이면 에러메시지를 출력하고 **useNavigate()를** 이용하여 "/"로 이동한다.

- 추가 import

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

```
import { Button } from 'react-bootstrap';
import { useNavigate, useParams } from 'react-router-dom';
import axios from 'axios';
```

```
//파라미터를 받는다.
   const {id} = useParams(); //

   const [board, setBoard] = useState({
    id:'',
    title:'' ,
    content:'',
    member:{}
   });

const navigator = useNavigate();

   useEffect(()=>{
      axios
      .get("http://localhost:9000/boards/"+id ,
         {
          headers: {
            Authorization: localStorage.getItem("Authorization"),
          } ,
       }
      )
      .then((res)=>{
           setBoard(res.data);
      })
      .catch((err)=>{
          errFun(err);
      });
   },[]);


  //에러 출력 함수
const errFun = (err)=>{
      if(err.response.status===403){
```

```
        alert("로그인하고 이용해주세요.");
           }else{
        let errMessage ="오류 = " +  err.response.data.type +"\n";
        errMessage += err.response.data.title +"\n";
        errMessage += err.response.data.status +"\n";
        errMessage += err.response.data.instance +"\n";
        errMessage += err.response.data.timestamp;

        alert(errMessage);
    }

     navigator("/");
   }
```

- **Detail.jsx 화면 디자인**

```
 <div>
 <h1>상세보기</h1>

 <Button variant='warning'
        onClick={updateBoard}
        disabled={board.member.name ===
localStorage.getItem("name") ? false : true}>
  수정
</Button>

           {'   '}

<Button variant='danger'
       onClick={()=>deleteBoard(board.id)}
          disabled={board.member.id ===
localStorage.getItem("id") ? false : true}>
  삭제
</Button>

  <hr/>
  <h2>글번호 : {board.id}</h2>
  <h2>제목 : {board.title}</h2>
  <h2>작성자 : {board.member.name}</h2>
  <h2>내용 : {board.content}</h2>
```

```
  </div>
```

-수정 and 삭제 이벤트

```
//삭제
const deleteBoard = (id)=>{

 }


   //수정 클릭
   const updateBoard = ()=>{

   }
```

☞ 삭제하기

: 삭제를 클릭하면 글번호를 인수로 받아 서버에 전송한다.
이때 headers를 이용하여
Authorization: localStorage.getItem("Authorization")
함께 전송한다.

: 삭제가 성공하면 "/" 으로 이동한다.

```
const deleteBoard = (id)=>{
    axios({
       method:"DELETE",
       headers: {
          Authorization: localStorage.getItem("Authorization"),
        },
       url : "http://localhost:9000/boards/"+id,
     })
     .then((res)=>{
        if(res.data ==="ok") navigator("/") ;
```

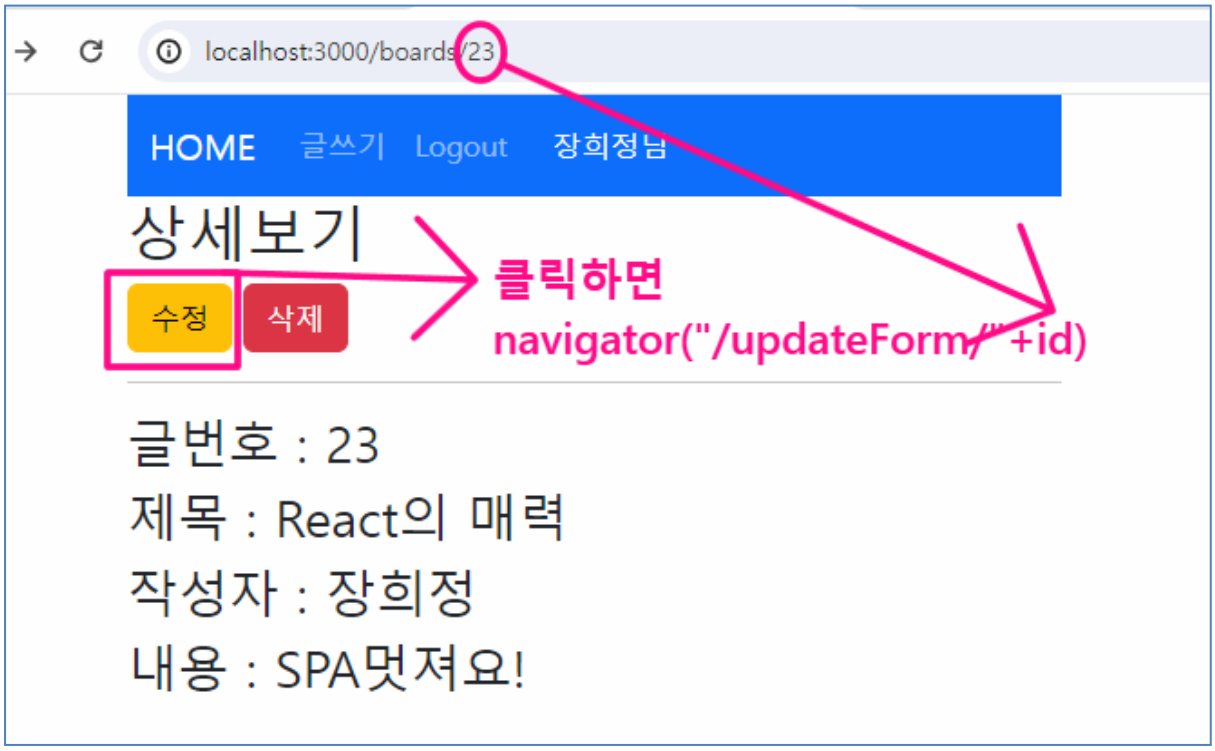
```
        else
          alert("삭제되지 않았습니다.");
    })
   .catch((err)=>{
      errFun(err);
   });
}
```

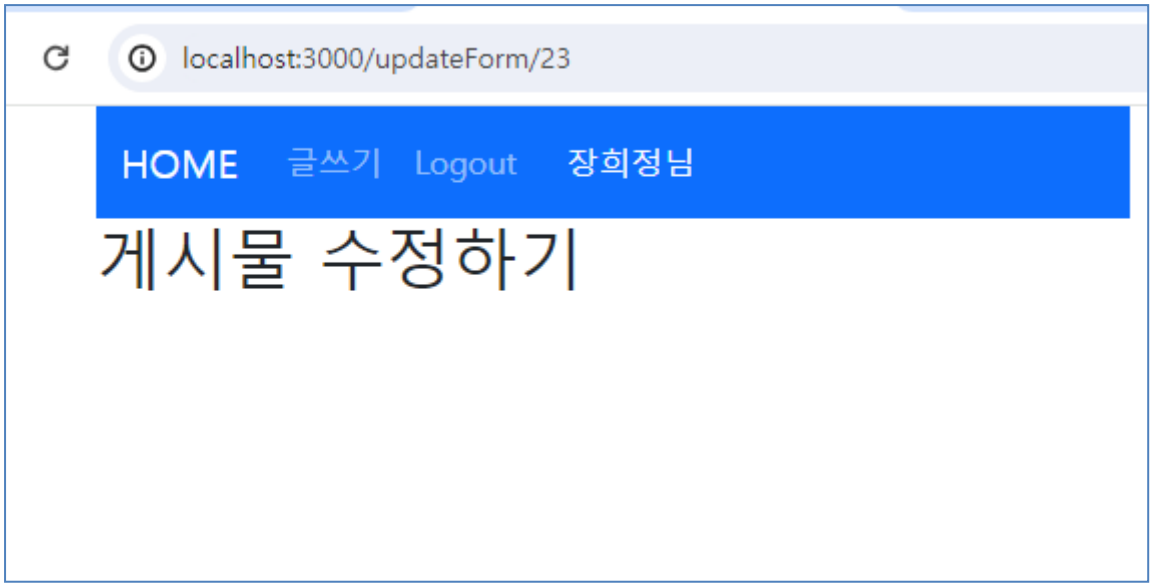
☞ 수정하기

: 수정을 클릭하면 UpdateForm.jsx 컴포넌트로 이동한다.
이때 수정하려는 게시물의 글번호를 전달한다.

```
//수정하기
   const updateBoard = ()=>{
       navigator("/updateForm/"+id);
   }
```

실행화면





☞ UpdateForm.jsx 에서

: usePramas()를 이용하여 전달된 글번호를 받는다.

: useState()를 이용하여 게시물의 정보를 저장한 변수를 선언한다.

: useEffect()를 이용하여 컴포넌트가 마운트(로딩)되었을 때 서버와 통신하여 게시물의 정보를 가져온다.

headers에 Authorization: localStorage.getItem("Authorization") 전송

- 추가 import 문

```
import { useEffect, useState } from 'react';
import { Button } from 'react-bootstrap';
import Form from 'react-bootstrap/Form';
import { useNavigate, useParams } from 'react-router-dom';
import axios from 'axios';
```

```
const {id} = useParams(); //

const [board, setBoard] = useState({
    title :'' ,
    content:'',
```

```
    member:{}
  });

//페이지 이동하는 방법
  const navigator = useNavigate();

  useEffect(()=>{
    axios
    .get("http://localhost:9000/boards/"+id , {
      headers: {
         Authorization: localStorage.getItem("Authorization"),
      },})
    .then((res)=>{
      setBoard(res.data);
    })
    .catch((err)=>{
        errFun(err);
    });
 },[]);


 const errFun = (err)=>{
    if(err.response.status===403){
        alert("로그인하고 이용해주세요.");

    }else{
       let errMessage ="오류 = " +  err.response.data.type +"\n";
        errMessage += err.response.data.title +"\n";
        errMessage += err.response.data.status +"\n";
        errMessage += err.response.data.instance +"\n";
        errMessage += err.response.data.timestamp;

        alert(errMessage);
    }

    navigator("/");
}
```

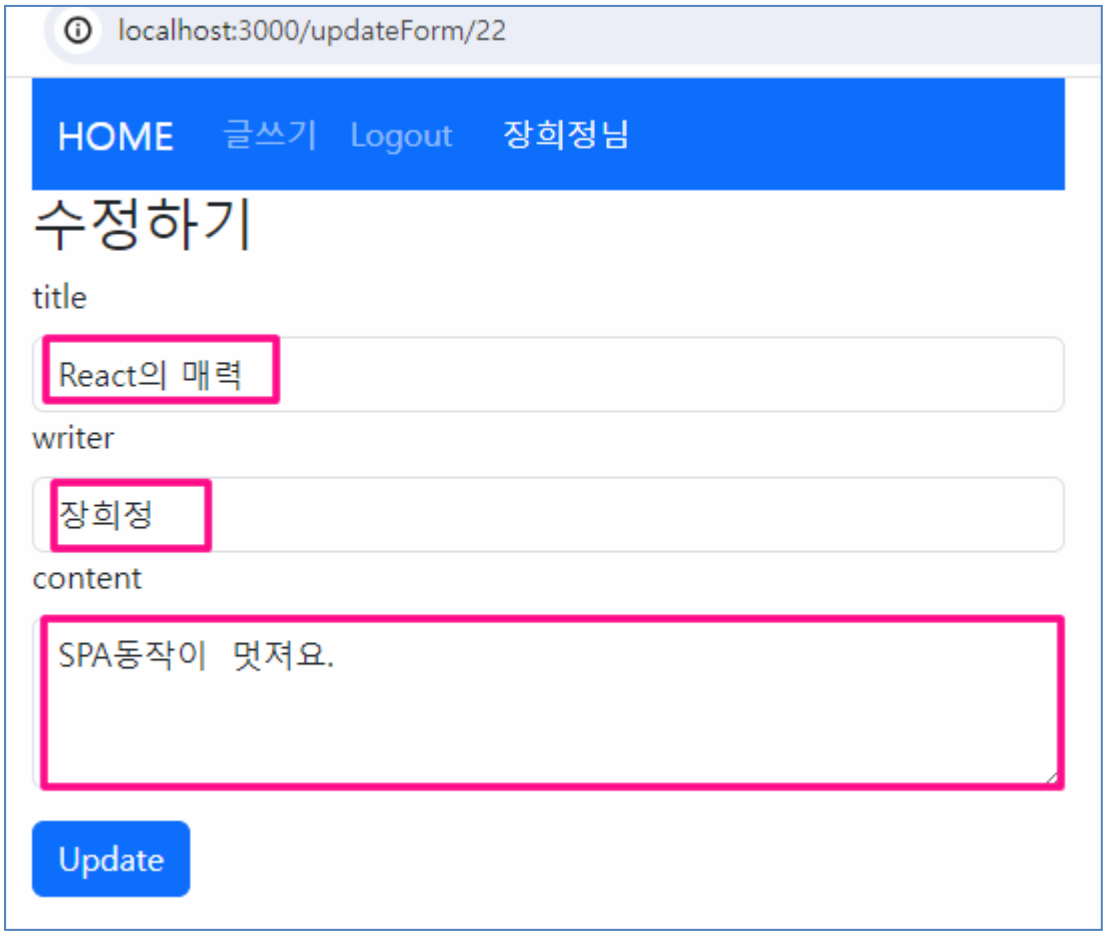
- 수정 폼 디자인

```
<>
<h2>수정하기</h2>

    <Form onSubmit={submitBoard}>

      <Form.Label htmlFor="title">title</Form.Label>
      <Form.Control
        type="text"
        id="title"
        name="title"
        onChange={changeValue}
        value={board.title}
      />

      <Form.Label htmlFor="author">writer</Form.Label>
      <Form.Control
        type="text"
        id="name"
        name="name"
        readOnly
        value={localStorage.getItem("name")}
      />

      <Form.Label>content</Form.Label>
        <Form.Control as="textarea"
       rows={3}
         name="content"
          id="content"
         onChange={changeValue}
         value={board.content}/>

      <p/>

      <Form.Control
        type="hidden"
        id="memberNo"
        name="memberNo"
        value={localStorage.getItem("memberNo")}
      />
```

```
      <p/>
      <Button variant='primary' type='submit'>Update</Button>
      <p/>

   </Form>
</>
```

```
const changeValue = (e)=>{

 }


 const submitBoard = (e)=>{

  };
```

실행결과



- text박스에 값이 입력될 때 board의 정보를 수정한다.

```
const changeValue = (e)=>{
        setBoard({
          ...board,
          [e.target.name]: e.target.value}
          )
    }
```

-Update 를 클릭하면 서버에 수정을 요청하고 성공하면
navigator("/boards/"+id); 로 이동한다.

```
const submitBoard = (e)=>{
        e.preventDefault();
```

```
        axios({
          method:"PUT",
          url : "http://localhost:9000/boards/"+id,
          data : board,
          headers: {
            Authorization: localStorage.getItem("Authorization"),
          }
        })
        .then((res)=>{
            navigator("/board/"+id);
        })
        .catch((err)=>{
           errFun(err);
       });
    }
```

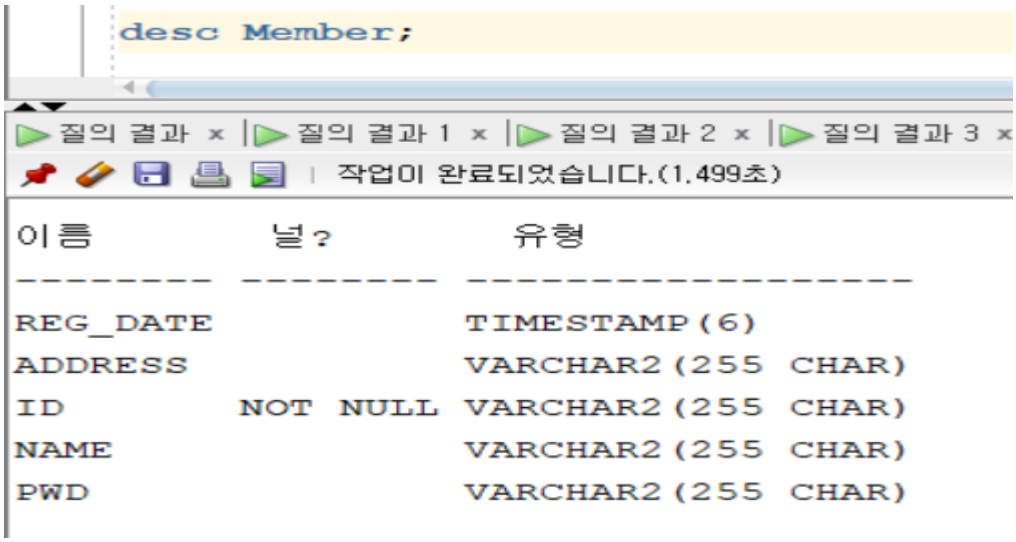
SpringBoot Entity 생성

☞ **Member.java**

```
@Entity
@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Member {
    @Id
    private String   id;
    private String   pwd;
    private String   name;
    private String   address;

    @CreationTimestamp
```

```
    private LocalDateTime regDate;
}
```

Table 생성 결과



☞ **Board.java**

```
@AllArgsConstructor
@NoArgsConstructor
@Setter
@Getter
@ToString
@Entity //서버 실행시에 해당 객체로 테이블 매핑생성
@Builder
public class Board {

    @Id//pk 를 해당 필드로 한다
    //@GeneratedValue(strategy = GenerationType.IDENTITY)//해당 디비 번호증가 전략을 따라가겠다.
    //@GeneratedValue(strategy = GenerationType.AUTO)//해당 디비 번호증가 전략을 따라가겠다.
    @GeneratedValue(strategy = GenerationType.SEQUENCE ,generator = "board_id")
    @SequenceGenerator(allocationSize = 1, sequenceName = "board_id" , name = "board_id")
    private Long id;//글번호
```

```
    private String title;//제목

    @Column(length =100)
    private String content;//내용

    @ManyToOne
    //@JoinColumn(name ="member_id")
    //@ManyToOne(fetch = FetchType.LAZY)
    private Member member;//작성자

    @CreationTimestamp
    private LocalDateTime regDate;//등록일

    @UpdateTimestamp
    private LocalDateTime updateDate;//수정일
}
```

Table결과

```
desc Board;
```

질의 결과 × | 질의 결과 1 × | 질의 결과 2 × | 질의 결과 3 ×

작업이 완료되었습니다.(1.029초)

이름	널?	유형
ID	NOT NULL	NUMBER(19)
REG_DATE		TIMESTAMP(6)
UPDATE_DATE		TIMESTAMP(6)
CONTENT		VARCHAR2(100 CHAR)
MEMBER_ID		VARCHAR2(255 CHAR)
TITLE		VARCHAR2(255 CHAR)

☞ **MemberRepository.java**

```
2 usages
public interface MemberRepository extends JpaRepository<Member, String>{

}
```

☞ **MemberRepository.java**

```
2 usages
public interface BoardRepository extends JpaRepository<Board, Long>{
    //@Query("select distinct b from Board b join fetch b.member")
    no usages
    @Query("select  b from Board b join fetch b.member")
     List<Board> join();
}
```

☞ **BoardReq.java**

```
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class BoardRes {
    private Long id;//글번호
    private String title;//제목
    private String content;//내용
    private MemberRes member;//작성자
    private String regDate;//등록일

    public BoardRes(Board board) {
        id = board.getId();
        title=board.getTitle();
        content=board.getContent();
        regDate=board.getRegDate().toString();
        member=new MemberRes(board.getMember().getId() ,
board.getMember().getName());

    }

    /*public BoardRes toBoardRes(Board board){
        return BoardRes.builder()
                .id(board.getId())
                .title(board.getTitle())
```

```
                .content(board.getContent())
                .regDate(board.getRegDate().toString())
                .member(new MemberRes(board.getMember().getId() ,
board.getMember().getName()))
                .build();
    }*/
}
```

☞ **BoardRes.java**

```
@Setter
@Getter
@ToString
public class BoardReq {
    private String title;//제목
    private String content;//내용
    private String memberId;//작성자

    public Board toBoard(BoardReq boardReq){
        return Board.builder().title(boardReq.getTitle()).content(boardReq.getContent())
                .member(Member.builder().id(boardReq.memberId).build())
                .build();
    }

}
```

☞ **MemberRes.java**

```
@Setter
@Getter
@AllArgsConstructor
@NoArgsConstructor
public class MemberRes {
    private String   id;
```

```
    private String   name;


}
```

☞ **MemberService.java**

```
public interface MemberService {
    /**
     * 가입
     * */
     void  signUp(Member member);

     String duplicateCheck(String id);

    /**
     * 로그인
     * */
    Member  signIn(String id, String pwd);
}
```

☞ **MemberServiceImpl.java**

```
@Service
@RequiredArgsConstructor
public class MemberServiceImpl   implements   MemberService {
    private final MemberRepository memberRepository;


    @Transactional
     @Override
     public void signUp(Member member) {
        Member m = memberRepository.save(member); //동일한 id가 들어오면 수정됨
         System.out.println("m = " + m);
     }


     @Transactional(readOnly = true)
     @Override
     public String duplicateCheck(String id) {
         Member member = memberRepository.findById(id).orElse(null) ;
```

```
            System.out.println("member = " + member);
            if(member==null) return "사용가능합니다.";
            else return "중복입니다.";

        }

        @Transactional(readOnly = true)
        @Override
        public Member signIn(String id, String   pwd) {
            Member member= memberRepository.findById(id).orElseThrow(
                    ()->new MemberAuthenticationException("아이디를 다시
확인해주세요.","Wrong Id"));
            if(!member.getPwd().equals(pwd)){
                throw new MemberAuthenticationException("비밀번호를 확인해주세요.",
"Wrong Pass");
            }

            return member;
        }
}
```

☞ **MemberRes.java**

☞ **MemberRes.java**

☞ Spring Security + JWT + React 연동 CORS 설정

: SecurityConfig.java 문서의 filterChain 메소드안에 추가

```
//CORS 설정
http
        .cors((corsCustomizer ->
corsCustomizer.configurationSource(new CorsConfigurationSource()
{
            @Override
            public CorsConfiguration
```

```
getCorsConfiguration(HttpServletRequest request) {

            CorsConfiguration configuration = new
CorsConfiguration();

configuration.setAllowedOrigins(Collections.singletonList("http:/
/localhost:3000"));

configuration.setAllowedMethods(Collections.singletonList("*"));
            configuration.setAllowCredentials(true);

configuration.setAllowedHeaders(Collections.singletonList("*"));
            configuration.setMaxAge(3600L);


configuration.setExposedHeaders(Collections.singletonList("Author
ization"));

            return configuration;
         }

      }));
```

WebMvcConfig.java 작성

```
package web.mvc.cofig;

import org.springframework.context.annotation.Configuration;
import
org.springframework.web.servlet.config.annotation.CorsRegistry;
import
org.springframework.web.servlet.config.annotation.EnableWebMvc;
import
org.springframework.web.servlet.config.annotation.WebMvcConfigure
r;

/**
 * WebMvcConfigurer를 이용해서 @CrossOrigin 글로벌 설정
 * */
@Configuration
@EnableWebMvc
public class WebMvcConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowedOrigins("http://localhost:3000")
                .allowedMethods("OPTIONS","GET","POST","PUT","DELETE
");
```

```
    }

}
```