**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.**

****

1. **생성된 프로젝트로 이동한다.**

**텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명**

1. **필요한 lib install한다.**

|  |
| --- |
| npm install react-router-dom  npm install react-axios |

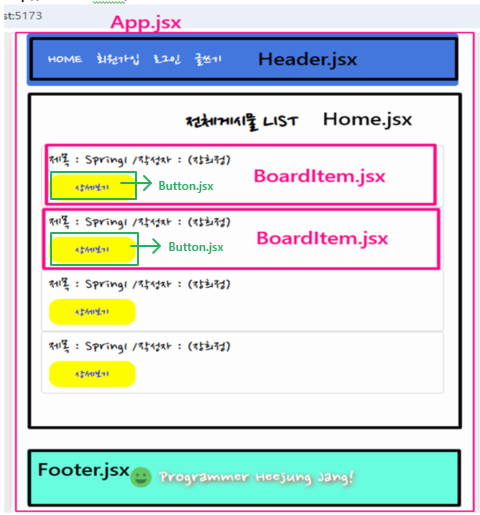
**package.json파일확인**

**텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

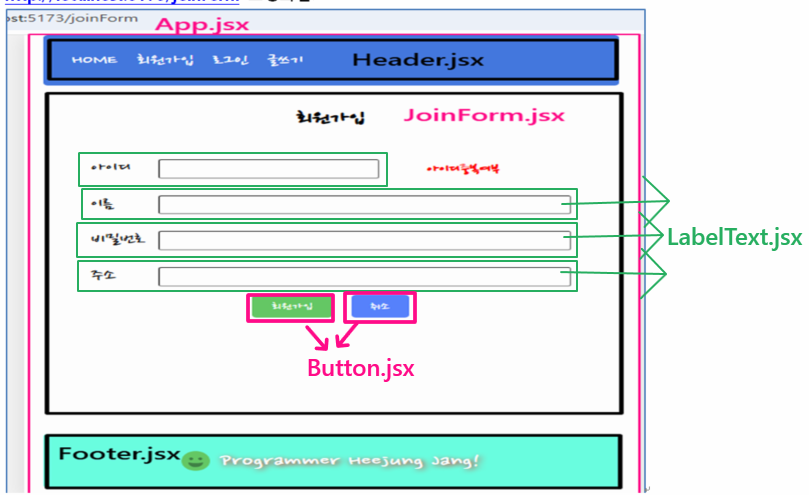
자동 생성된 설명**

**전체화면구성**

**http://localhost:5173/요청 화면**

****

[**http://localhost:5173/joinForm**](http://localhost:5173/joinForm) **요청화면**

****

[**http://localhost:5173/loginForm**](http://localhost:5173/loginForm) **요청화면**

**텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

[**http://localhost:5173/saveForm**](http://localhost:5173/saveForm) **요청화면**

**텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

[**http://localhost:5173/board/1**](http://localhost:5173/board/1) **요청화면**

**텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명**

[**http://localhost:5173/updateForm/3**](http://localhost:5173/updateForm/3) **요청화면**

**텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**프로젝트 Directory구조**

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

자동 생성된 설명**

**먼저, 각 화면을 담당하는 컴포넌트 파일들을 만들어 놓는다**.

**main.jsx에 BrowserRouter를 연결한다.**

**텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명**

**App.jsx피일에 Route를 연결한다.**

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명**

**Route 세팅이 완료 된 후에 요청주소로 접속하여 페이지가 잘 열리지는 확인한다.**

예) <http://localhost:5173/> - Home 컴포넌트

예) <http://localhost:5173/saveForm> - SaveForm 컴포넌트

예) <http://localhost:5173/board/1> - Detail 컴포넌트

예) <http://localhost:5173/updateForm/1> - UpdateForm 컴포넌트

예) <http://localhost:5173/loginForm> - LoginForm 컴포넌트

예) <http://localhost:5173/joinForm> - JoinForm 컴포넌트

**-회원관리 화면 구성**

**1)회원 가입**

user > JoinForm.jsx | JoinForm.css

**2) 로그인**

user > LoginForm.jsx | LoginForm.css

**-게시판관리 화면구성**

**1) 게시물 등록**

board > SaveForm.jsx | SaveForm.css

**2) 전체게시물**

board > Home.jsx | Home.css

**3) 상세보기**

board > Detail.jsx | Detail.css

**4) 수정하기**

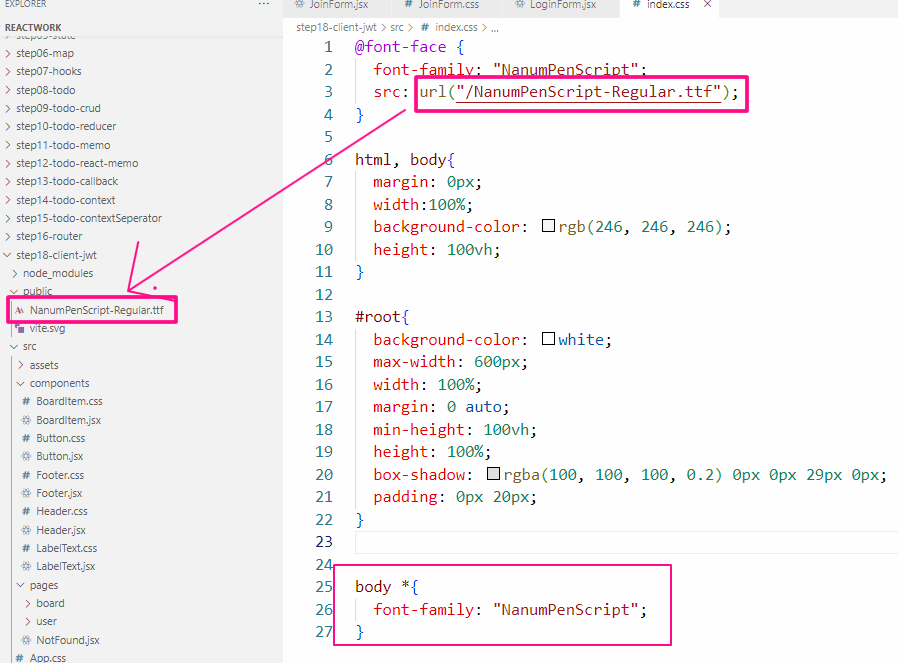
board > UpdateForm.jsx | UpdateForm.css

**화면구성 만들어보자!**

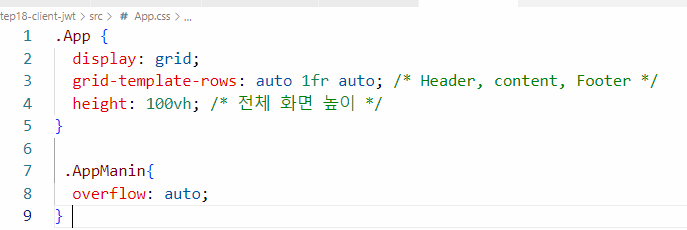
전체 페이지에 공통으로 적용 할 CSS

src > index.css | App.css

index.css파일



App.css 파일



src > components > Header.jsx | Header.css | Footer.jsx | Footer.css

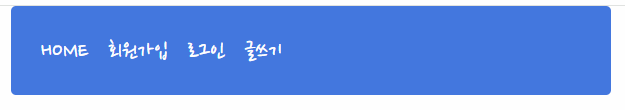
Header.jsx파일



Header.css파일



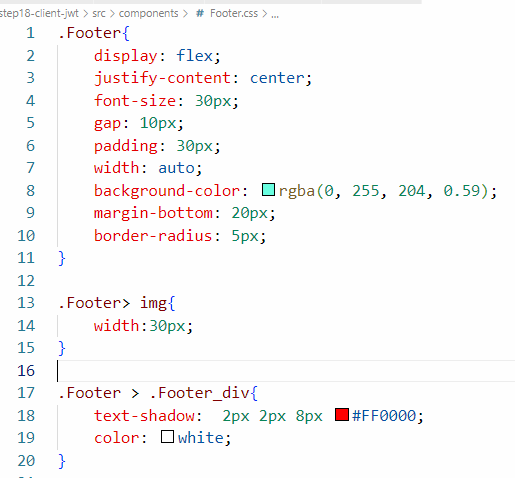
실행화면



Footer.jsx파일



Footer.css파일



실행화면



**LabelText.jsx파일 이해하기**

현재 프로젝트에서 아래의 화면이 여러 컴포넌트에서 사용되고 있다. 때문에 우리는 아래 부분만 따로 제작하여(LabelText.jsx) 필요할 때 재사용 해서 사용하도록 하자.

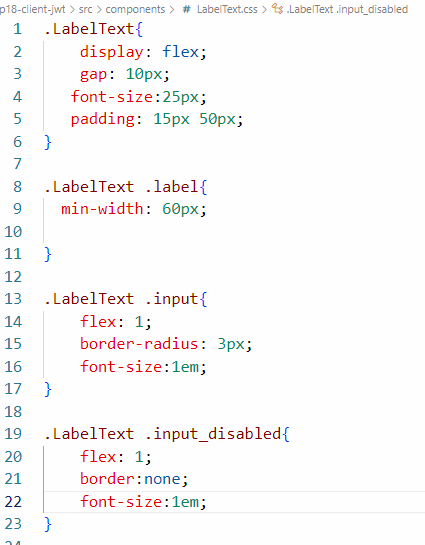
텍스트, 스크린샷, 라인, 폰트이(가) 표시된 사진

자동 생성된 설명

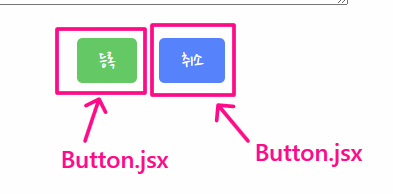
**LabelText.jsx파일**



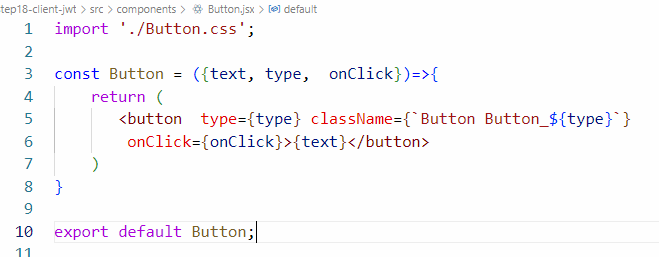
**LabelText.css**



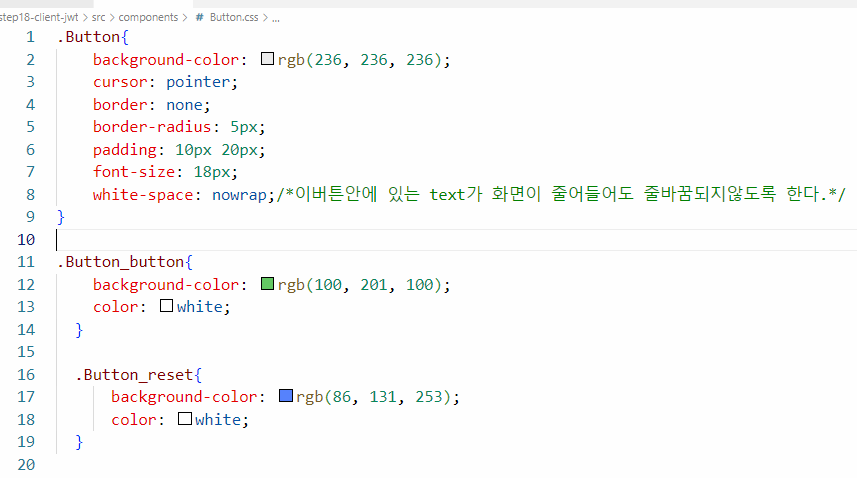
Button.jsx파일 이해 하기



전체 프로젝트에서 button이 여러 곳에서 쓰여지고 있고 text와 css만 다르기 때문에 컴포넌트로 만들어 놓고 재 사용하자.



Button.css파일



**1)회원 가입**

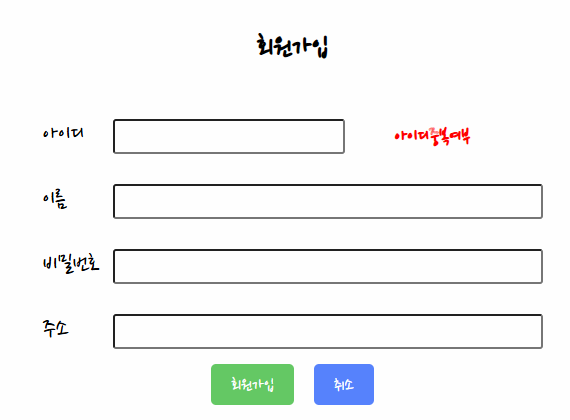
user > JoinForm.jsx | JoinForm.css



JoinForm.css 파일



**실행화면**

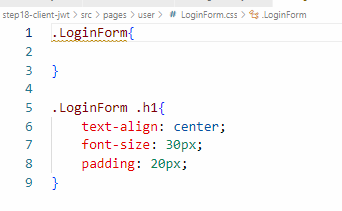
****

**2) 로그인**

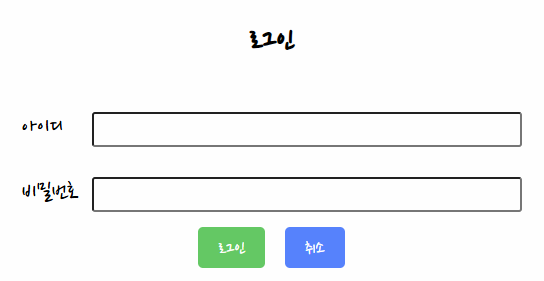
user >LoginForm.jsx | LoginForm.css



LoginForm.css



실행화면



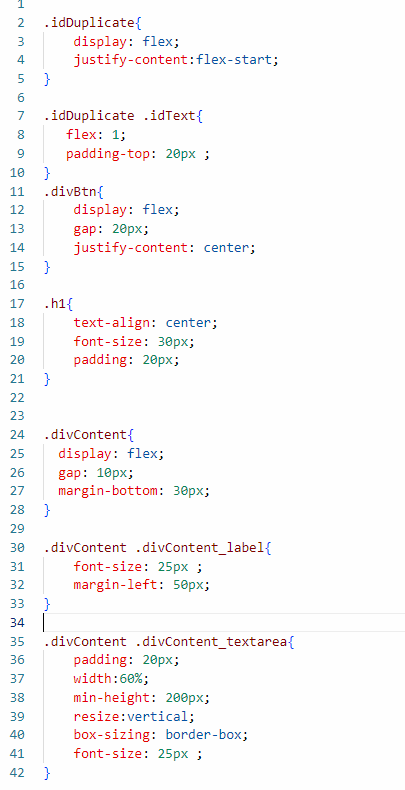
**1) 게시물 등록**

board > SaveForm.jsx | SaveForm.css

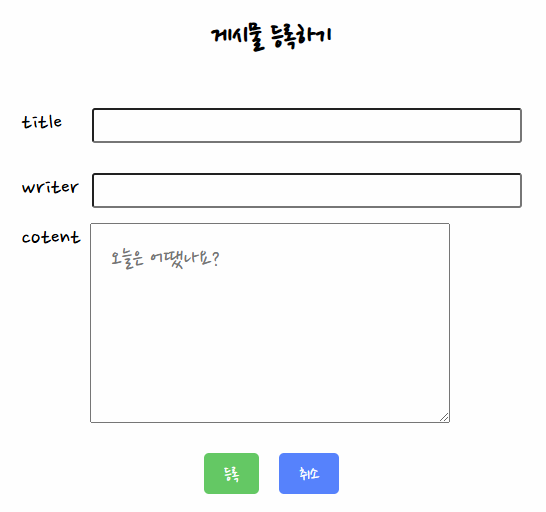
SaveForm.jsx파일



SaveForm.css파일



실행화면

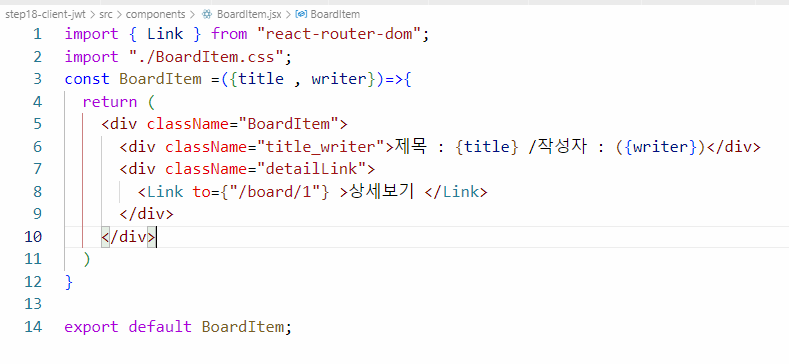


**2) 전체게시물**

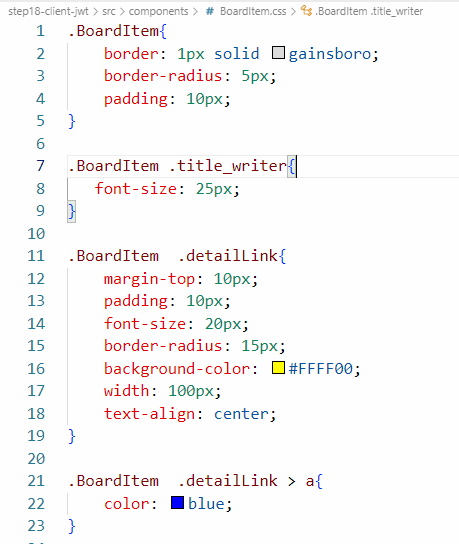
components > BoardItem.jsx | BoardItem.css

board > Home.jsx | Home.css

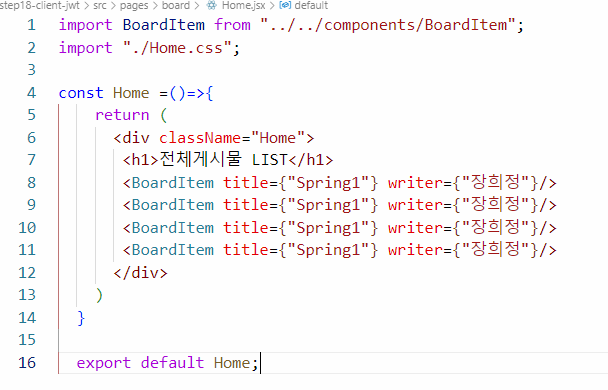
BoardItem.jsx



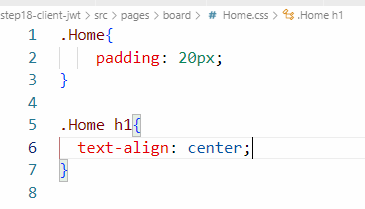
BoardItem.css파일



Home.jsx파일



Home.css 파일



실행화면



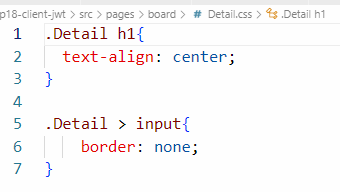
**3) 상세보기**

board > Detail.jsx | Detail.css

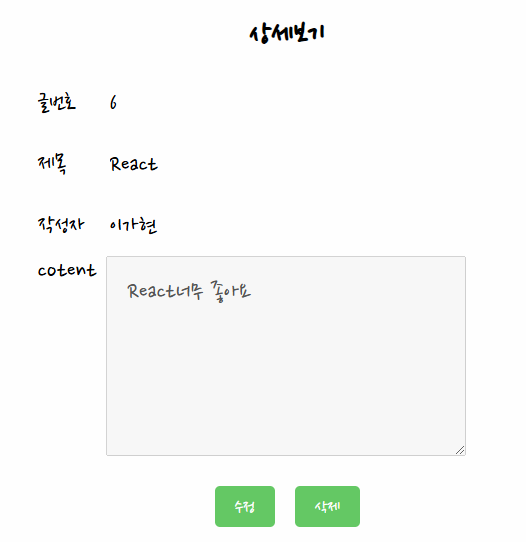
Details.jsx파일



Details.css 파일



실행화면



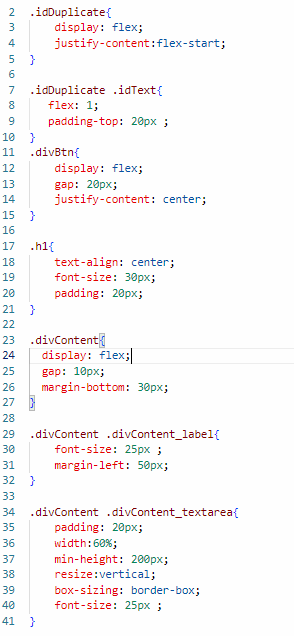
**4) 수정하기**

board > UpdateForm.jsx | UpdateForm.css

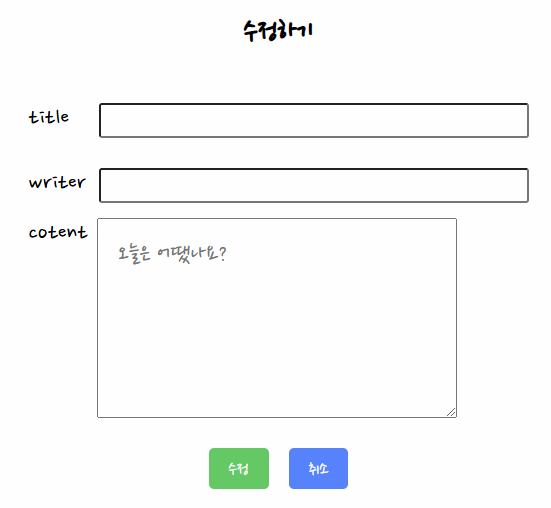
UpdateForm.jsx파일

****

**UpdateForm.css파일**

****

**실행화면**

****

**주요기능 만들기**

**- 회원가입(JoinForm.jsx파일)**

**☞ 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,          })          .then((res)=>{              console.log(res);          })          .catch((err)=>{           //실패            leterrMessage = 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);         });        } |

**.**

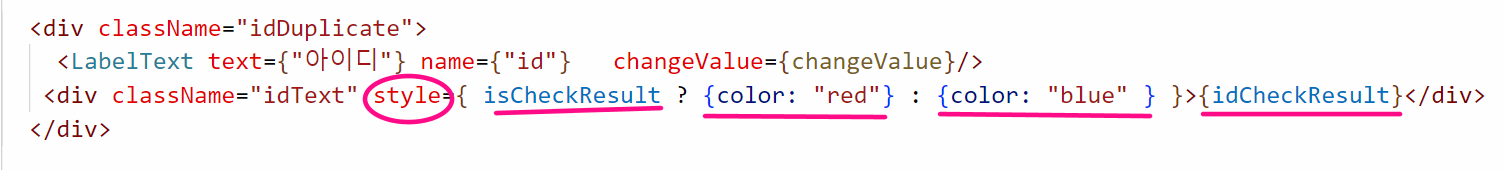
**☞ 중복체크 결과를 아이디 text박스 옆에 출력한다.**

|  |
| --- |
| // **중복체크 결과 값을 저장 할 idCheckResult** - 중복입니다. or 사용가능합니다.  const [idCheckResult ,setIdCheckResult] = useState("");    // **아이디 중복여부에 따른 css를 적용하기 위해 상태 변수 -** true이면 중복, false이면 사용가능  const [isCheckResult ,setIsCheckResult] = useState(false); |

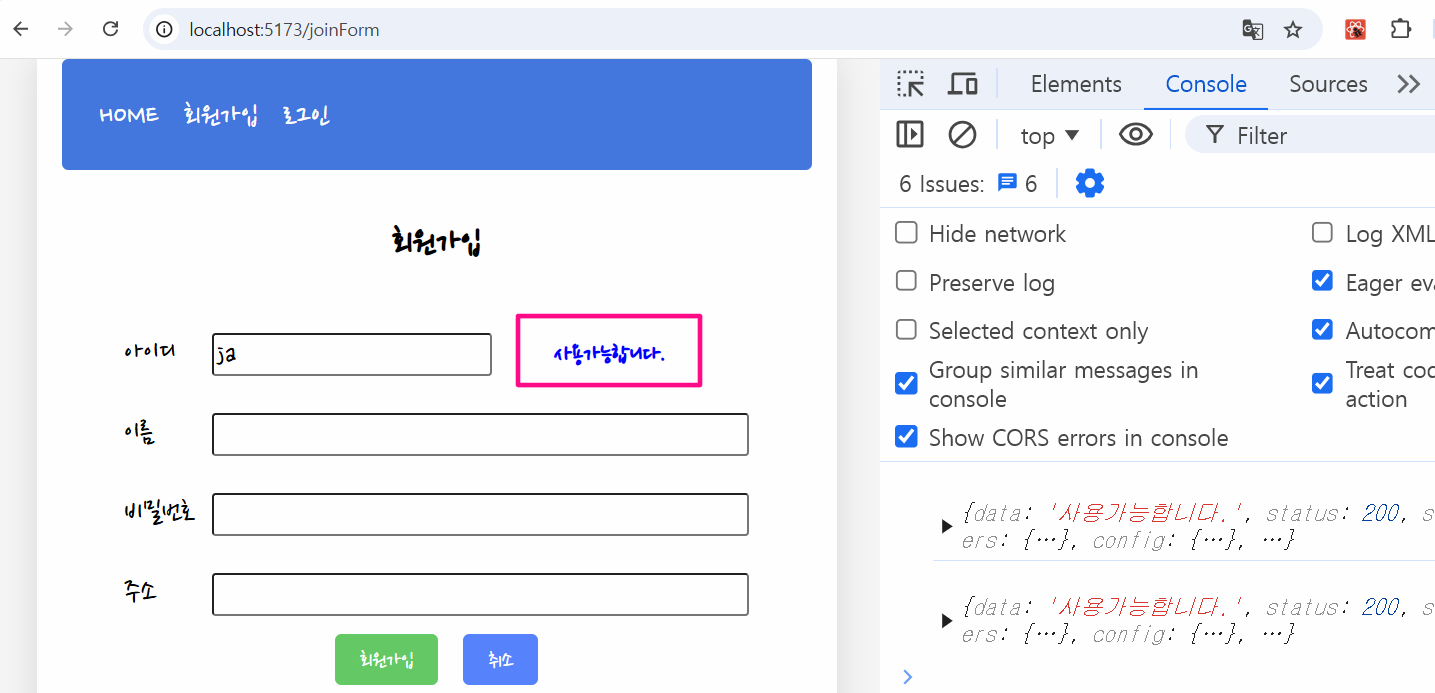
**☞ axios결과를 받아서** setIdCheckResult , setIsCheckResult 값변경

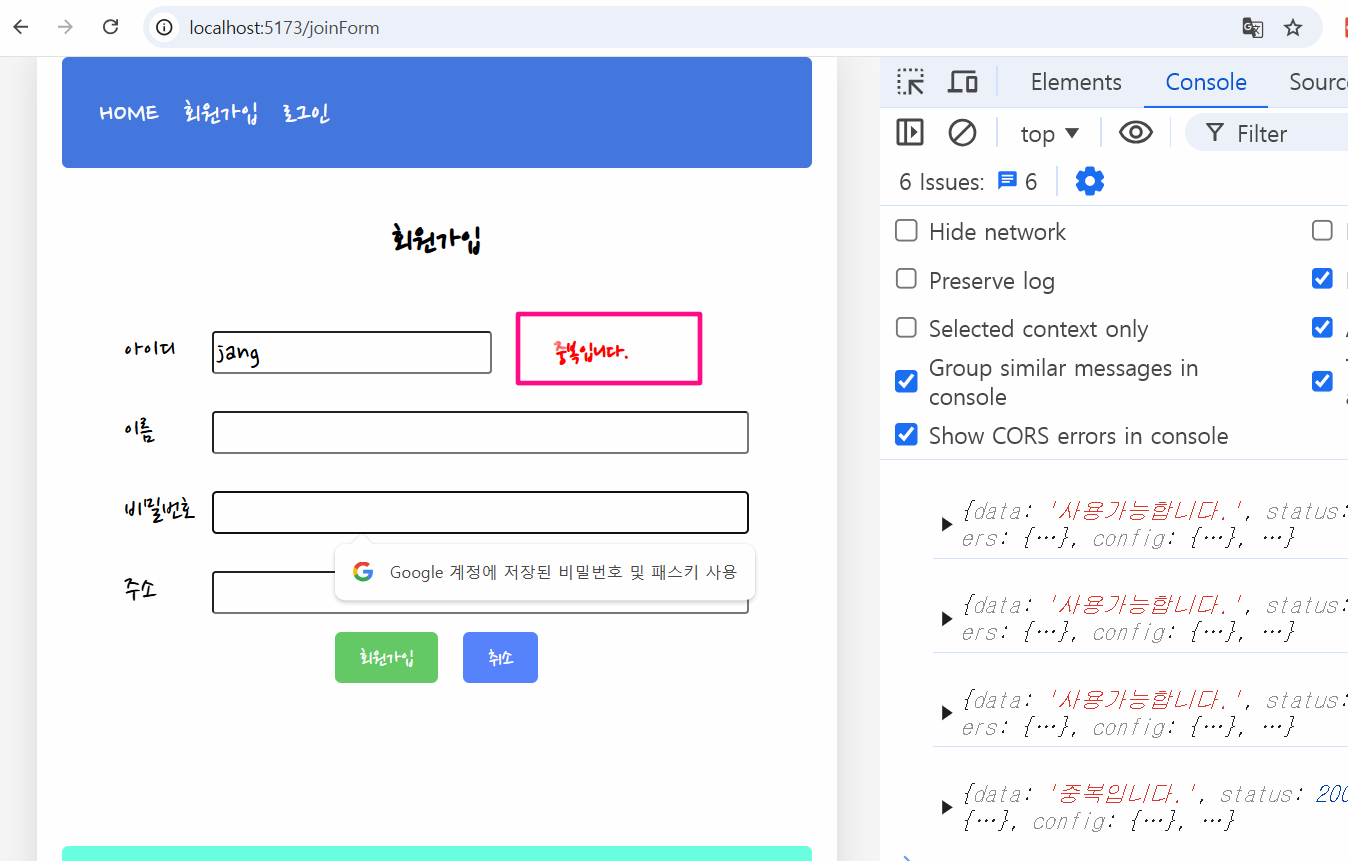
|  |
| --- |
| .then((res)=>{      setIdCheckResult(res.data);     res.data==="중복입니다." ?  setIsCheckResult(true) :setIsCheckResult(false);   }) |

**☞ 아이디 text박스 옆에 결과를 출력한다.**

****

**실행결과**

****

****

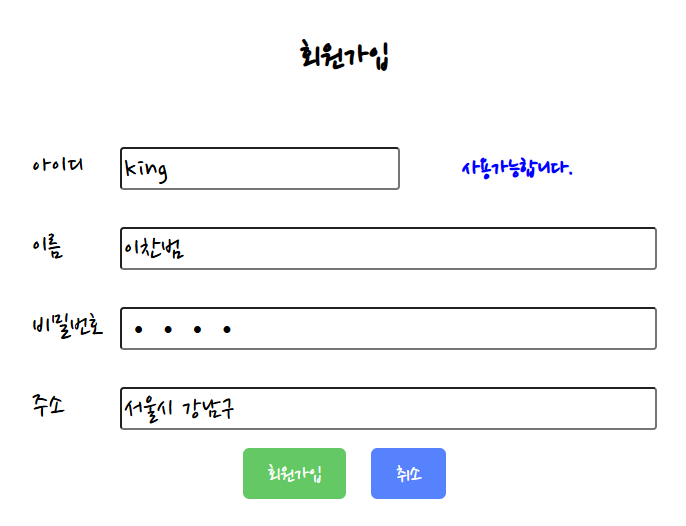
**☞ 회원가입 클릭**

**: 가입을 성공하면 “/” Home으로 이동한다.**

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

****

**실행결과**

****

**로그인 하기**

**: backend spring security를 이용하여 인증을 진행 할 예정이다.**

때문에 입력한 **아이디와 비밀번호는 Controller가 아닌 Spring security의 filter에 전달된다**.

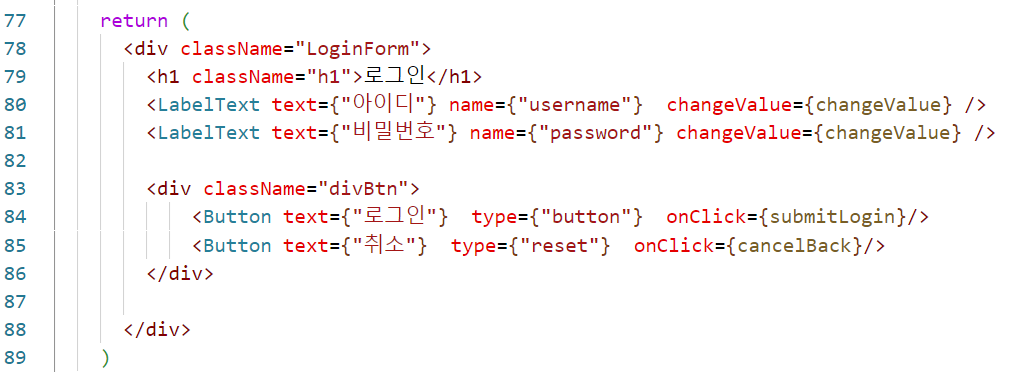
**Security는**

**Parameter 정보 -> 아이디 : username , 비번 : password**

**parameter정보를 json이 아닌 form data로 전송 해야한다.**

**요청주소 -> /login**

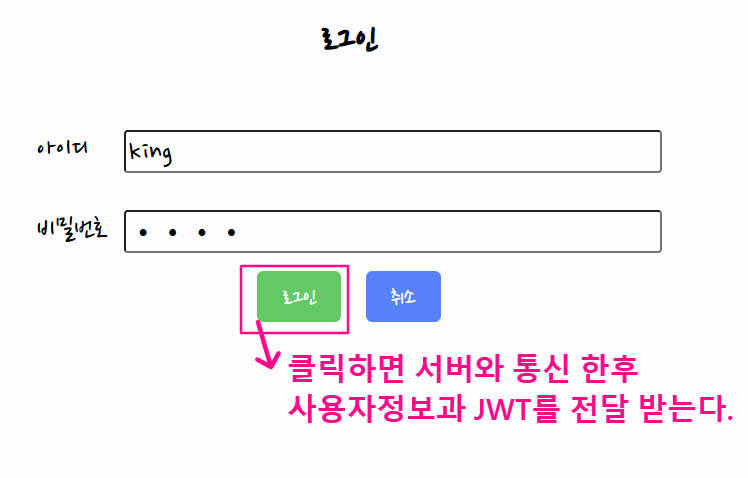
**☞ 로그인 폼 디자인 – LoginForm.jsx**

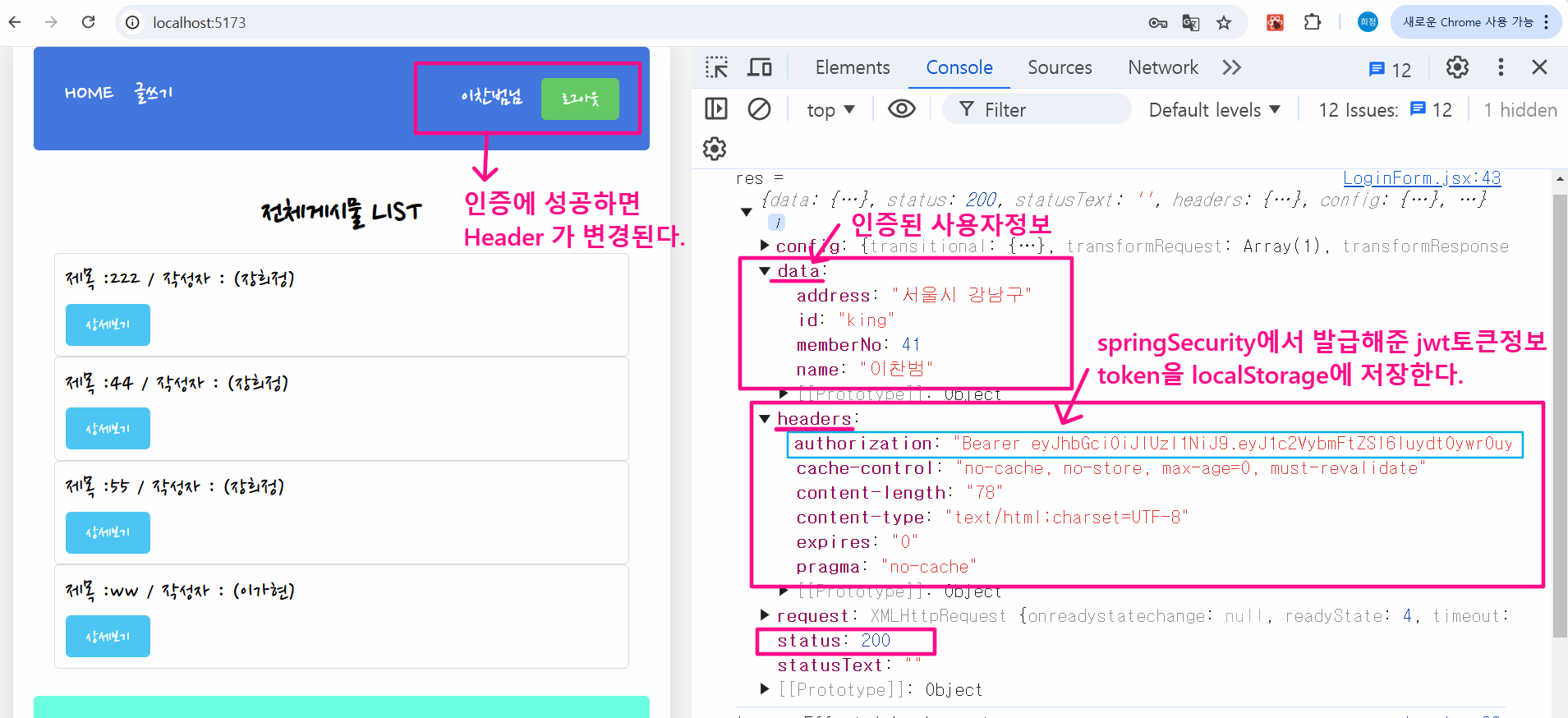
****

**☞ 로그인 기능구현**

|  |
| --- |
| // 인증에필요한 username, password 상태관리를 위한useState  const [ member, setMember ] = useState({      username :'' ,      password:'',  });  // input에 값이 입력될 때 상태값 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();//submit이벤트막음      const formData = new FormData(); //폼전송으로보내기위한작업      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.");    } |

**실행결과**

****

****

**☞** 로그인이 성공하면 인증된 사용자의 정보를 1)localStorage에 저장하고

App.jsx에 있는 isLogin 변수를 true로 변경하기 위해 useContext로 Context에 있는 LoginContext를 가져와서 2)handelLoginChange 함수를 호출한다.

3)Home.jsx로 이동한다.

|  |
| --- |
| //인증된 정보 localStorage에 저장  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);     //App.js에 있는 isLogin 변수를 true 변경한다.     handleLoginChange(true);    //Home.jsx로 이동한다.      navigator("/"); |

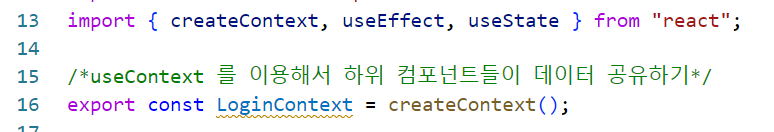
**App.jax파일**

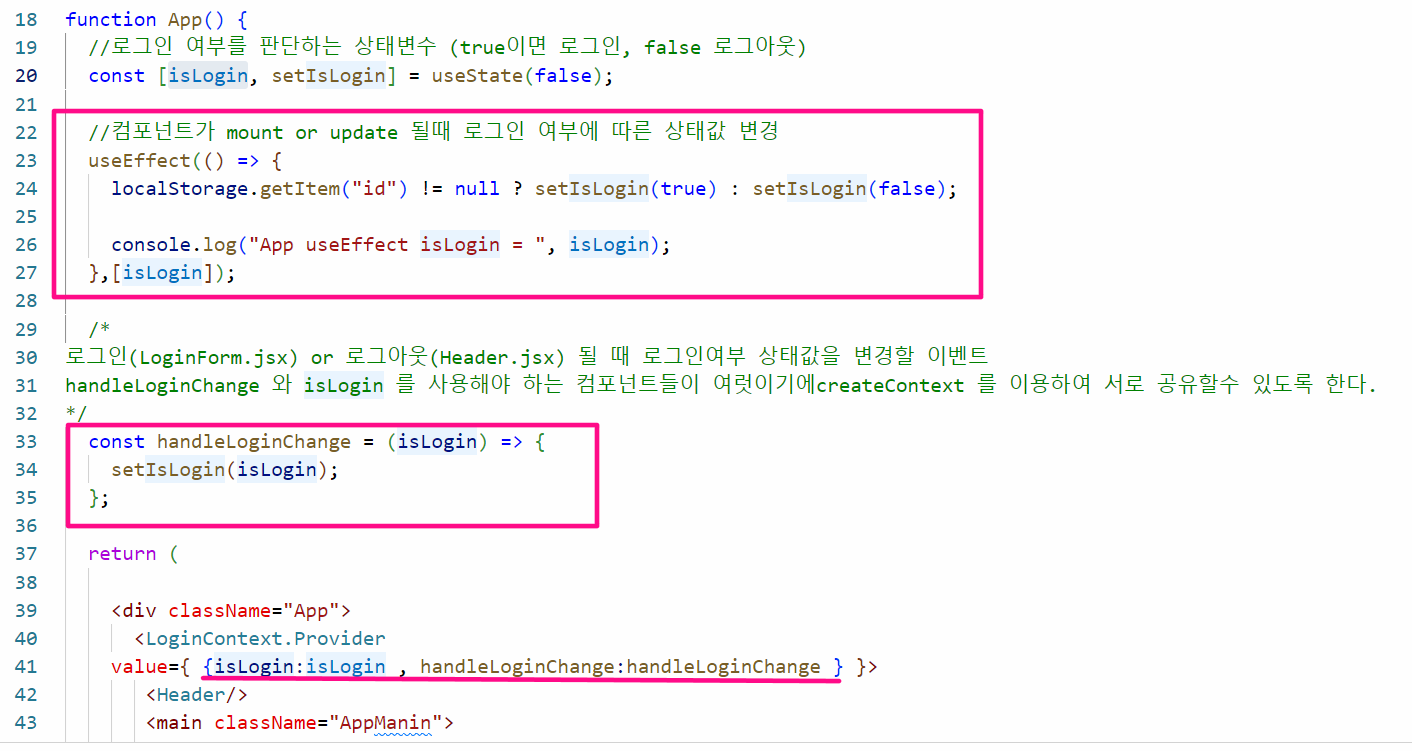
**먼저, App.js문서에서**

1. 로그인 여부를 체크 할 상태변수 isLogin 과

상황에 따라 상태변수의 값을 변경 할 함수=handleLoginChange를 만든다.

1. main.jsx 요청되면 -> App.jsx 로딩 ->useEffect()를 이용하여localStorage에 인증된 정보가 있는지 확인하여 상태변수 isLogin 을 변경한다.(있으면 true, 없으면 false)
2. createContext()를 이용하여 하위 컴포넌트들이 isLogin, handleLoginChange를 공유 하도록 한다.

****

****

**Header.jsx파일**

****

**인증 안 된 경우**

****

**-인증 된 경우**

****

**☞LoginForm.jsx 수정**

**: useContext를 이용하여 전달된 Context를 사용한다**

|  |
| --- |
| // {isLogin:isLogin , handleLoginChange:handleLoginChange }  const { handleLoginChange} = useContext(LoginContext); |

**☞로그인이 성공했을 때 콜백함수에서 인증여부 상태변수의 값을 변경한다.**

|  |
| --- |
| //App.js에 있는 isLogin 변수를 true 변경한다.   handleLoginChange(true); |

**그리고, useNavigate()를 이용하여 “/” Home으로 이동한다.**

|  |
| --- |
| //페이지 이동하는 방법  const navigator = useNavigate();   //Home.jsx로 이동한다.    navigator("/"); |

**☞Header.jsx 문서에서 useCcontext 선언**

|  |
| --- |
| const {isLogin , handleLoginChange} = useContext(LoginContext); |

**☞Header.jsx 문서에서 로그아웃 되었을 때 이벤트**

**: 모든 Storage에 저장된 정보를 삭제한 후**

handleLoginChange **함수에 false를 전달하여 호출한다**

**그리고 , /로 이동한다.**

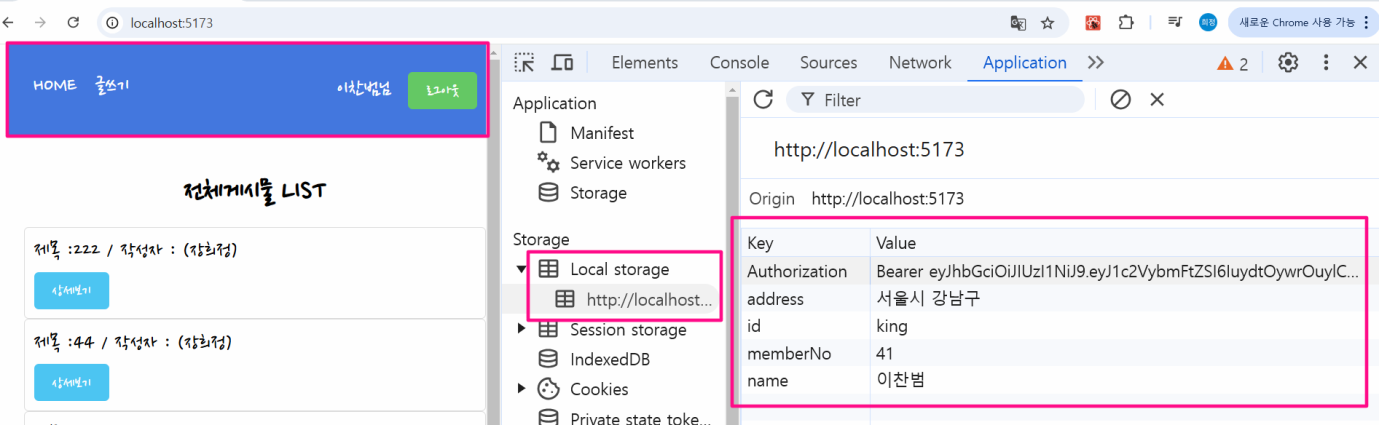
|  |
| --- |
| const navigator = useNavigate();    const logoutCheck = ()=>{      localStorage.clear();//모든 localStorage의 정보 지우기      handleLoginChange(false); //App.jsx에 선언된 state가 변경되면서 Header컴포넌트 리랜더링      navigator("/");    } |

**☞Header.jsx 문서에서 인증여부에 따른 메뉴 출력**

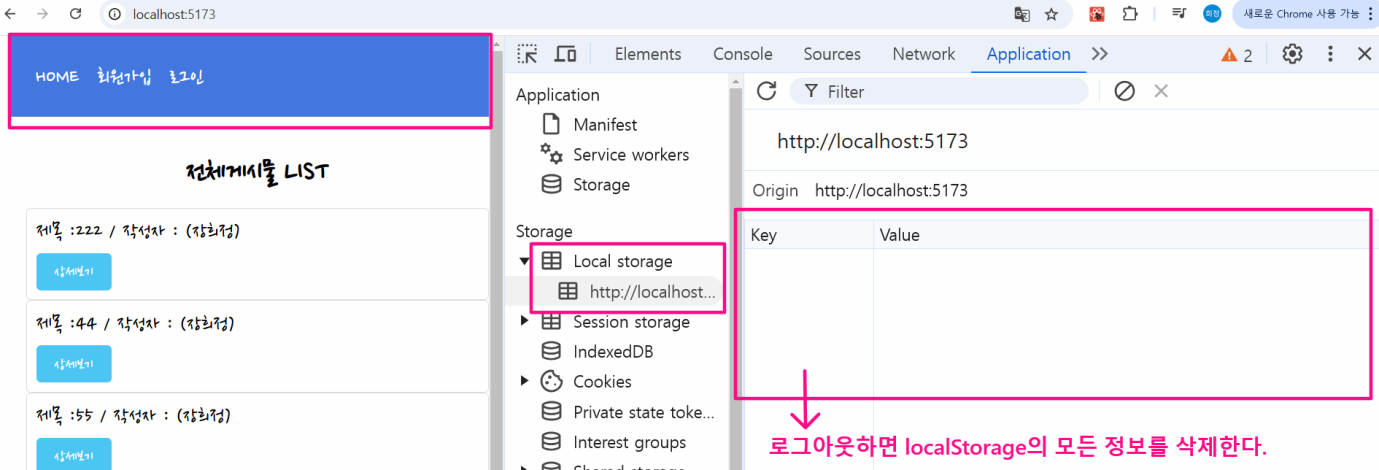
|  |
| --- |
| return (        <>          <div className="Header">              <Link to="/" >HOME</Link>              { isLogin && <Link to="/saveForm">글쓰기</Link>}              { !isLogin && <Link to="/joinForm">회원가입</Link>}                { isLogin ? (                 <div className="login">                  <span>{ localStorage.getItem("name") }님</span>                  <Button text={"로그아웃"} type={"button"} onClick={logoutCheck} />                 </div>              )  **:**                <Link to="/loginForm">로그인</Link>             }            </div>        </>      ) |

**실행결과**

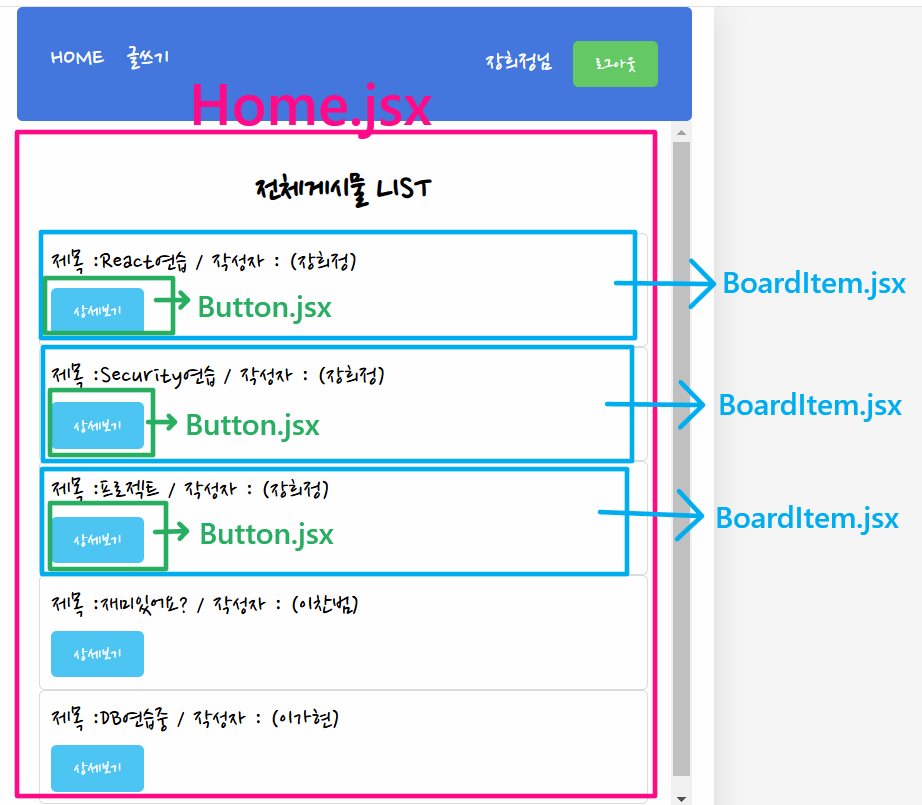
**로그인성공 했을 때**

****

**로그아웃 했을 때**

****

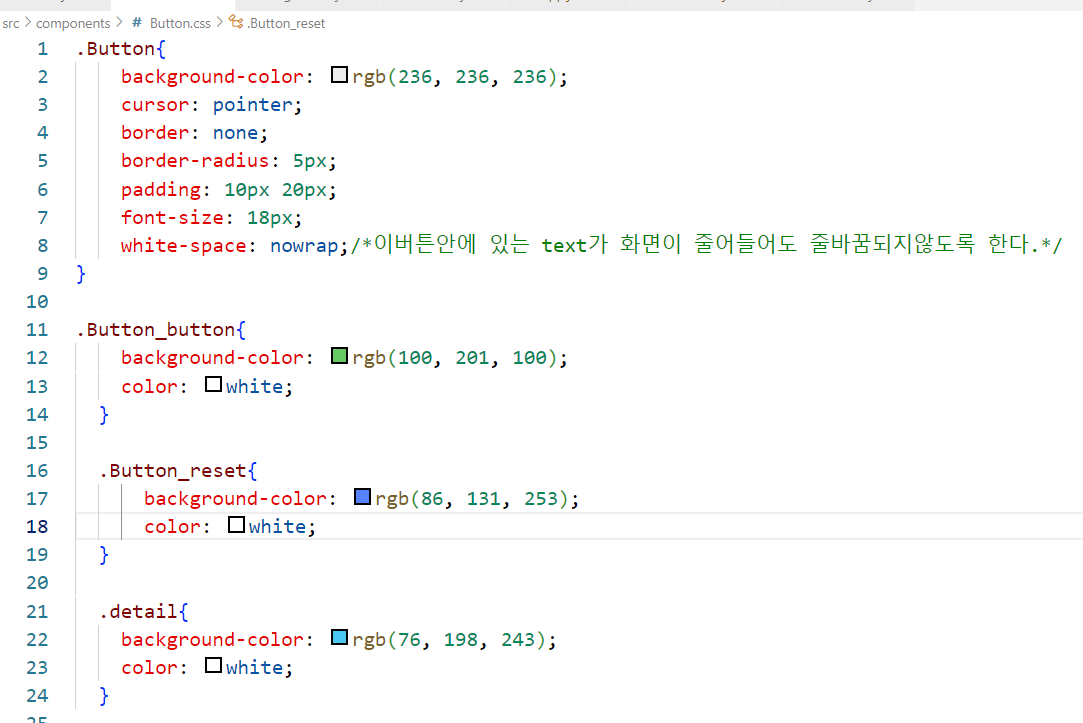
**☞ 전체 게시물 보기**

****

**src> components > Button.jsx**

****

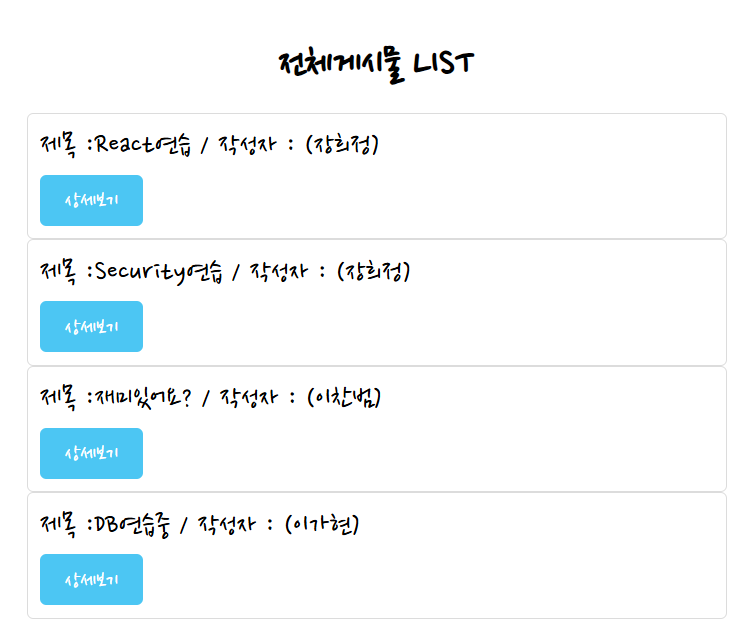
**src> components > Button.css**

****

**src> pages > Home.jsx(전체소스)**

|  |
| --- |
| const Home =()=>{    const { handleLoginChange} = useContext(LoginContext);     //DB 목록을 저장해서 관리 할 useState     const [boards , setBoards] = useState([]);     const nav = useNavigate();       //서버에서 데이터 조회   - axios       useEffect(()=>{          axios          .get("http://localhost:9000/boards", {          headers: {          Authorization : localStorage.getItem("Authorization"),          },})          .then((res)=>{              //console.log(res);             setBoards(res.data);//state에서 관리되는 변수가 변경되어 update(re-rendering)          })          .catch((err)=>{              console.log(err)              if(err.response.status===403){                 localStorage.clear();//모든 세션의 정보 지우기                 handleLoginChange(false);                 alert("로그인하고 이용해주세요.");                 nav("/loginForm");              }else{                  alert(err.response.data.detail) ;              }         });       }, []);      return (        <div className="Home">         <h1>전체게시물 LIST</h1>          {            boards.map( (board)=> <BoardItem key={board.id} board={board}/> )          }        </div>      )  }    export default Home; |

**실행결과**

****

**Home.jsx에 코드를 추가 해보면서 코드을 완성해보자.**

BoardItem 컴포넌투는 데이터베이스에 있는 게시물의 개수 만큼 반복되야한다.(map함수이용)

그러기 위해서는 DB 테이블에 있는 레코드를 조회 해야 한다.

Home.jsx 에서 React Hook의 종류인 useEffect를 이용하여 DB에 들어있는 데이터를 비 동기통신으로(axios) 조회한다.

**([ ] 안넣는다면?) =>최초실행 + 상태값변경 때마다 실행됨**

**☞axios를 이용하여 모든 게시물 조회**

**- Home.jsx import**

|  |
| --- |
| import { useContext, useEffect, useState } from "react";  import BoardItem from "../../components/BoardItem";  import "./Home.css";  import axios from 'axios';  import { useNavigate } from "react-router-dom";  import { LoginContext } from "../../App"; |

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

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

|  |
| --- |
| //서버에서 데이터 조회   - axios       useEffect(()=>{          axios          .get("http://localhost:9000/boards", {          headers: {          Authorization: localStorage.getItem("Authorization"),          },})          .then((res)=>{              //console.log(res);             setBoards(res.data);//state에서 관리되는 변수가 변경되어 update(re-rendering)          })          .catch((err)=>{              console.log(err)              if(err.response.status===403){                 localStorage.clear();//모든 세션의 정보 지우기                 handleLoginChange(false);                 alert("로그인하고 이용해주세요.");                 nav("/loginForm");              }else{                  alert(err.response.data.detail) ;              }         });       }, []); **//([ ] 안넣는다면?) =>최초실행 + 상태값변경 때마다 실행됨** |

-Home.jsxDB에서 조회된 boards의 정보를 map을 이용하여출력한다.

BoardItem 컴포넌트에 props를 이용하여 board의 정보를 전달한다.

|  |
| --- |
| return (        <div className="Home">         <h1>전체게시물 LIST</h1>          {            boards.map( (board)=> <BoardItem key={board.id} board={board}/> )          }        </div>      ) |

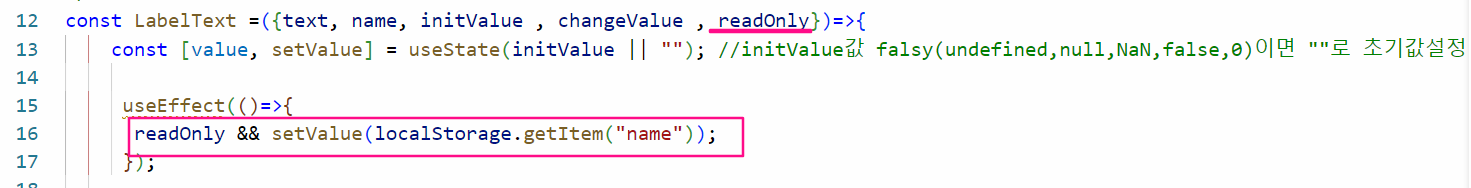
**-BoardItem.jsx 에서 props로 전달된 데이터를 받아서 출력**

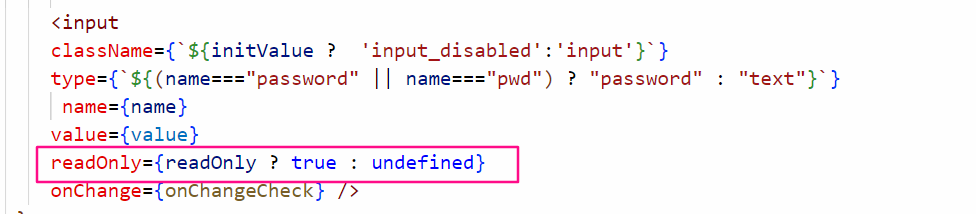
****

**☞ 글 쓰기 화면구성(SaveForm.jsx)**



**LabelText.jsx파일**

****

****

**실행화면**

****

**- text박스에 값이 입력될 때 state를 변경하기**

**: 작성자의 pk는 localStorage에서 조회**

|  |
| --- |
| const SaveForm =()=>{    //게시물 상태관리    const [board, setBoard] = useState({      title: "",      content: "",      memberNo: localStorage.getItem("memberNo"),    }); //제목,내용, 작성자pk=memberNo,    //input 에 값이 변경될 때    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";

**:** importaxiosfrom"axios";

|  |
| --- |
| //페이지 이동하는 방법    const navigator = useNavigate();   //등록하기 클릭    const submitBoard = (e) => {      axios({        method: "POST",        url: "http://localhost:9000/boards/board",        data: board,        headers: {          Authorization: localStorage.getItem("Authorization"),          }      })        .then((res) => {          console.log(res);          navigator("/");        })        .catch((err) => {          if (err.response.status === 403) {            alert("로그인된 사용자만이 글을 등록 할 수 있어요.");            navigator("/");          } else {           alert(err.response.data.detail);          }       });    }; |

**실행화면**

****

**☞ BoardItem.jsx 파일에서 상세보기 클릭**

****

**Home.jsx에서**

**☞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 글번호에 해당하는 게시물 조회를 서버에 요청 한 후 게시물의 정보를 응답 받는다.

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

****

**-Detail.jsx파일 일부 소스**

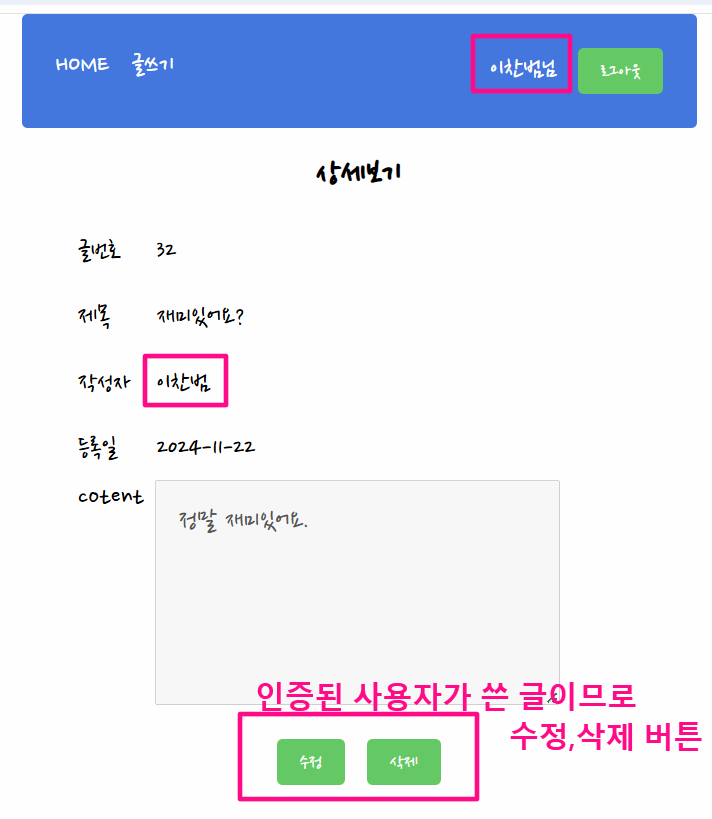
|  |
| --- |
| const Detail =()=>{     //파라미터를 받는다.     const { id } = useParams(); //  nav("/board/"+board.id) 요청에서 id 가져오기     const [board, setBoard] = useState({       id: "",       title: "",       content: "",       member: {},       regDate:""     });      const nav = useNavigate();    useEffect(() => {      axios        .get("http://localhost:9000/boards/" + id, {          headers: {            Authorization: localStorage.getItem("Authorization"),          },        })        .then((res) => {          console.log(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);      }      nav("/");    };    return (        <div className="Detail">         <h1>상세보기</h1>            <LabelText text={"글번호"}  initValue={board.id}/>          <LabelText text={"제목"}  initValue={board.title}/>          <LabelText text={"작성자"}  initValue={board.member.name}/>          <LabelText text={"등록일"}  initValue={board.regDate}/>            <div className="divContent">            <div className="divContent\_label">cotent</div>            <textarea className="divContent\_textarea" placeholder="오늘은 어땠나요?"             name="content"  disabled value={board.content}></textarea>         </div>          { board.member.name === localStorage.getItem("name") &&         <div className="divBtn">              <Button text={"수정"}  type={"button"}  onClick={()=>{nav("/updateForm/"+id)}}/>              <Button text={"삭제"}  type={"button"} onClick={deleteBoard} />          </div>        }          </div>      )    }      export default Detail; |

**src> components > LabelText.jsx**

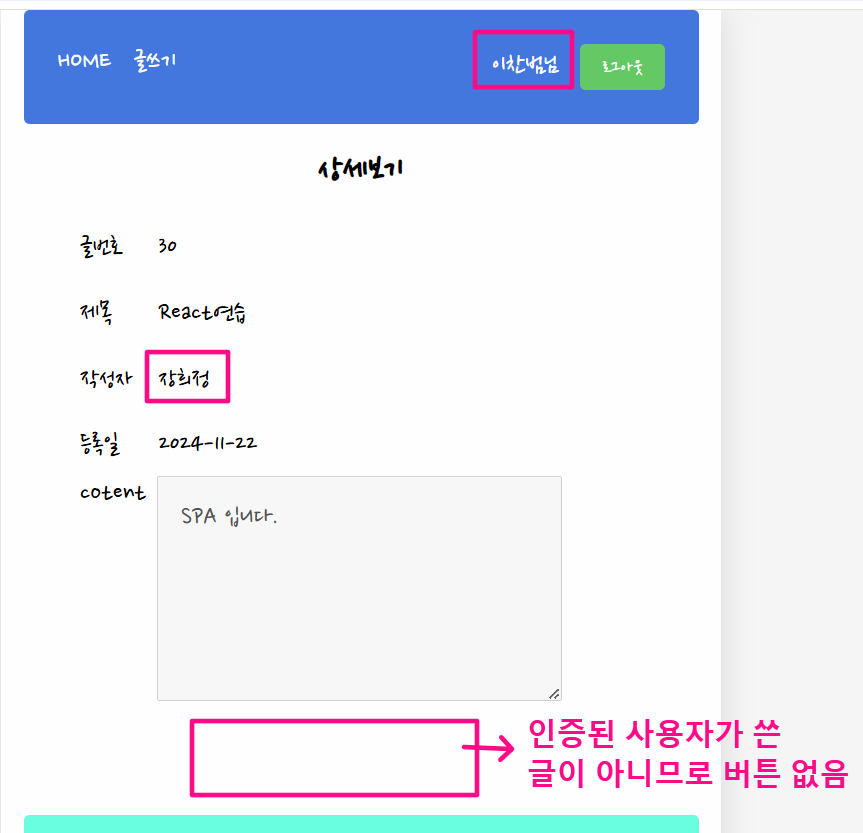
****

****

**- Detail.jsx자신이 쓴 글 클릭하면, 수정, 삭제 버튼생성**

****

**-Detail.jsx자신이 쓴 글이 이나면 수정, 삭제 버튼 없음.**

****

**-수정 and 삭제 이벤트**

|  |
| --- |
| //삭제 클릭  const deleteBoard = (id)=>{      }     //수정클릭     constupdateBoard = ()=>{       } |

**☞ 삭제하기**

: 삭제를 클릭하면 글번호를 인수로 받아 서버에 전송한다.

이때 headers를 이용하여 token을 함께 전송한다.

|  |
| --- |
| headers: {          Authorization: localStorage.getItem("Authorization"),   } |

: 삭제가 성공하면 “/” 으로 이동한다.

|  |
| --- |
| const deleteBoard = () => {      axios({          method:"DELETE",          headers: {          Authorization: localStorage.getItem("Authorization"),          },          url : "http://localhost:9000/boards/"+id,          })          .then((res)=>{          if(res.data ==="ok")               nav("/") ;          else           alert("삭제되지 않았습니다.");       })      .catch((err)=>{           console.log("err = " + err)           errFun(err);      });    };//삭제끝 |

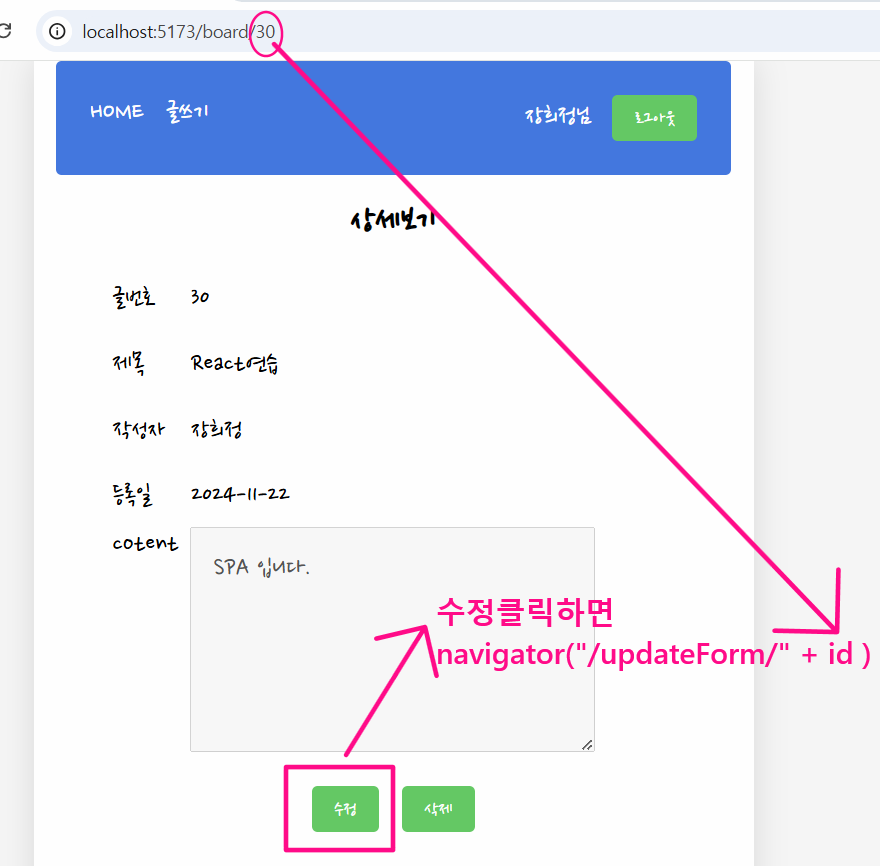
**☞ 수정하기**

: 수정을 클릭하면 UpdateForm.jsx컴포넌트로 이동한다.

이때 수정하려는 게시물의 글번호를 전달한다.

|  |
| --- |
| //수정하기     constupdateBoard = ()=>{         navigator("/updateForm/"+id);     } |

**실행화면**

****

**UpdateForm.jsx 실행화면**

****

**☞ UpdateForm.jsx에서**

: usePramas()를 이용하여 전달된 글번호를 받는다.

: useState()를 이용하여 게시물의 정보를 저장할 상태변수를 선언한다.

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

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

-import 문

|  |
| --- |
| import { useNavigate, useParams } from "react-router-dom";  import Button from "../../components/Button";  import LabelText from "../../components/LabelText";  import './UpdateForm.css';  import { useEffect, useState } from "react";  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**

|  |
| --- |
| return (        <>         <h1 className="h1">  수정하기 </h1>        <form>          <LabelText text={"title"} name={"title"} initValue={board.title}          changeValue={changeValue} />            <LabelText text={"writer"} name={"name"}  readOnly="yes"/>            <div className="divContent">            <div className="divContent\_label">cotent</div>            <textarea            className="divContent\_textarea"            placeholder="오늘은 어땠나요?"             name="content"             value={board.content}             onChange={changeValue}></textarea>         </div>          <div className="divBtn">              <Button text={"수정"}  type={"button"}  onClick={submitBoard}/>              <Button text={"취소"}  type={"reset"} />          </div>          </form>        </>      ) |

|  |
| --- |
| constchangeValue = (e)=>{     }     constsubmitBoard = (e)=>{  }; |

- text박스에 값이 입력될 때 board의 정보를 수정한다.

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

- 수정을 클릭하면 서버에 수정을 요청하고 성공하면

navigator("/boards/"+id); 로 이동한다.

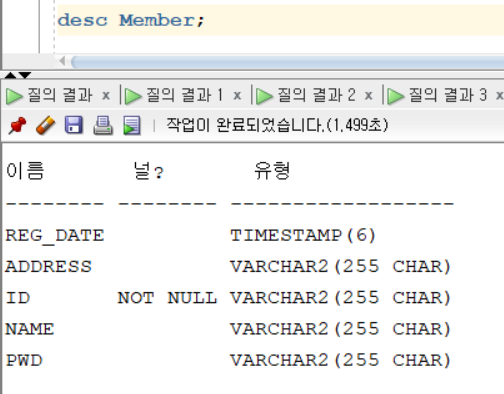
|  |
| --- |
| const submitBoard = (e)=>{      axios({          method:"PUT",          url : "http://localhost:9000/boards/"+id,          data : board,          headers: {          Authorization: localStorage.getItem("Authorization"),          }          })          .then((res)=>{             navigator("/board/"+id);          })          .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 LocalDateTimeregDate; } |

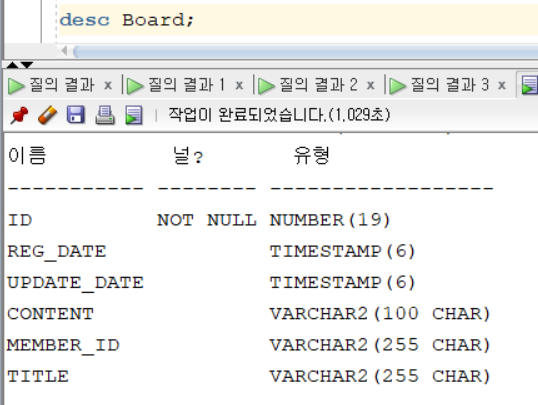
**Table 생성 결과**

****

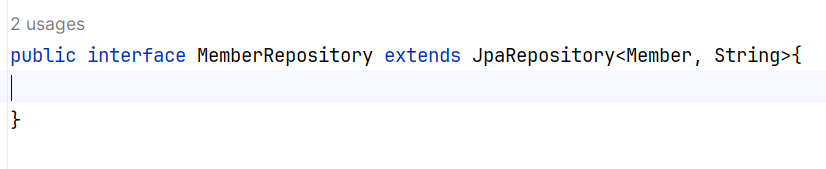
**☞ Board.java**

|  |
| --- |
| @AllArgsConstructor @NoArgsConstructor @Setter @Getter @ToString @Entity *//서버 실행시에 해당 객체로 테이블 매핑생성* @Builder public class Board {  @Id*//pk를 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 LocalDateTimeregDate;*//등록일* @UpdateTimestamp private LocalDateTimeupdateDate;*//수정일* } |

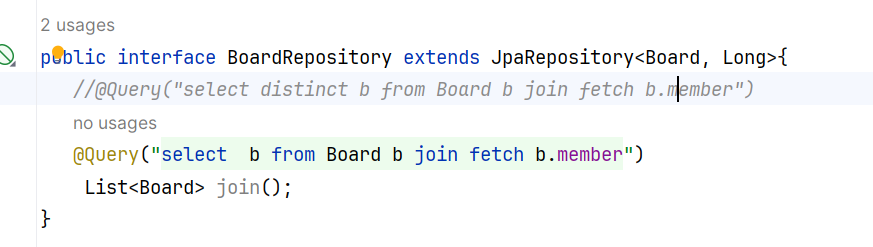
**Table결과**

****

**☞ MemberRepository.java**



**☞ MemberRepository.java**



**☞BoardReq.java**

|  |
| --- |
| @Getter @Setter @AllArgsConstructor @NoArgsConstructor @Builder public class BoardRes{ private Long id;*//글번호* private String title;*//제목* private String content;*//내용* private MemberResmember;*//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 BoardRestoBoardRes(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(BoardReqboardReq){ 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 MemberServiceImplimplements MemberService{ private final MemberRepositorymemberRepository;  @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.java문서의 filterChain메소드안에 추가**

|  |
| --- |
| *//CORS설정* http .cors((corsCustomizer->corsCustomizer.configurationSource(new CorsConfigurationSource() { @Override public CorsConfigurationgetCorsConfiguration(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*("Authorization"));  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.WebMvcConfigurer;  */\*\*  \* WebMvcConfigurer를 이용해서 @CrossOrigin 글로벌 설정 \* \*/* @Configuration @EnableWebMvc public class WebMvcConfigimplements WebMvcConfigurer{ @Override public void addCorsMappings(CorsRegistry registry) { registry.addMapping("/\*\*")  .allowedOrigins("http://localhost:3000")  .allowedMethods("OPTIONS","GET","POST","PUT","DELETE");  } } |