개발일지

2025350003 최동주

5/29: 기초 환경 세팅

영상을 보며 기초적인 환경 세팅을 완료하였다. 3주차 과제와 축제 등 여러 일정때문에 늦게 시작하게 되었다. js와 jsx 관련한 기초적인 문법들을 돌아보고 개발을 시작하게 되었다. jsx가 html의 문법을 중간에 끼어넣을 수 있다는 것이 너무 신기했다. 어떤 방식으로 개발을 해야 할지 감이 잡히지는 않지만 일단은 이것저것 찾아보면서 해야겠다.

우선 기본으로 제공되는 frontend의 src 내의 파일들을 통해 프론트엔드를 꾸미기로 했다. App.jsx의 App 내부 부분을 다음과 같이 바꾸었다.

```
function App() {
  // useState 변수 사용하는 자리 ex) const [count, setCount] = useState(0)
  return(
    <>
    <h1>CHAT</h1>
    <div className='chat-log'>
    </div><br></br>
    <div className='chat-input'>
      <input type='text'></input>
      <button>전송</button>
    </div>
    </>
)
}
```

아주 간단한 형식이다. 계획은 input과 버튼을 통해 메시지를 입력하면, 그것이 chat과 입력 사이 공간에 나타나는 방식으로 하고 싶다. 우선은 commit을 하겠다.

5/30: 회원가입 및 로그인 기능 구현

우선 채팅이 어떻게 이루어질지 고민을 해보았는데, 우선은 대화를 나누는 주체가 필요하기에 로그인 시스템을 구현해보고 생각을 해보기로 했다. 다만 3주차에서는 php와 mysql을 사용해서 구현했지만, 이번엔 js와 mongodb를 이용해야 하기에 어떻게 해야 할지 감이 안잡혀서 mern을 통해 로그인과 회원가입을 만드는 블로그와 유튜브들을 참고하기로 했다.

블로그들을 보니 router라는 기능을 사용하는것 같아 임시 코드로 짜보니 router 설치가 안되어 있다고 떠 설치를 했다.

또한 index.js에서 사용할 mongbodb의 환경 역시 세팅했다.
이후 env파일을 추가해주었고

```
require("dotenv").config();

mongoose.connect(process.env.MONGO_URI)
.then(() =>console.log('DB 연결 성공'))
.catch(err =>console.error( err));
```

위와 같이 db와 연결하는 간단한 코드를 넣었다.

```
const Users =new mongoose.Schema({
    id:{
        type:String,
        required:true,
        unique:true
    pw:{
        type:String,
        required:true
}
});
```

이후 이렇게 유저에 관한 스키마를 생성하였다.
백엔드 서버를 실행시키니 console에 db연결 성공이라고 뜬다. 우선 commit을 하겠다.

```
const userdata =mongoose.model('userdata',Users);
module.exports =userdata;
```

이후 해당 코드를 만들어 userdata라는 컬렉션을 생성했으며, 다른 파일에서도 사용 할 수 있도록 export하였다.

이제 로그인과 회원가입 기능을 구현하면 된다.

우선, login.jsx와 siginin.jsx를 추가해주었고, router기능을 활용해 app.jsx에 다음과 같은 코드를 추가하였다.

```
    <Router>
      <div>
        <Link to="/login">로그인</Link>
```

```
        <Link to="/signin">회원가입</Link>
        <Routes>
          <Route path="/login"element={<Login />}/>
          <Route path="/signin"element={<Signin />}/>
        </Routes>
      </div>
    </Router>
```

각각의 jsx는 비어있지만, 추후에 login과 sigin 컴포넌트를 랜더링 할 수 있도록 미리 적어두었다. 우선은 commit하겠다.

이후 각종 블로그 글들과 유튜버들의 mern 강의를 참고하여 1차적인 signin.jsx와 index.js 코드를 완성시켰다.

```
importuseStatefrom "react";
function Signin() {
    const[id, setid] =useState("");
    const[pw, setpw] =useState("");
    const signin =async() =>{
        const res =await fetch("http://localhost:5001/signin", {
            method:"POST",
            headers:"Content-Type":"application/json"},
            body:JSON.stringify({ id, pw}),
});
        alert(await res.text());
    return(
        <div>
            <h2>회원가입</h2>
                        <input   type="text"placeholder="아이디"onChange={e
=>setid(e.target.value)}/>
                    <input  type="password"placeholder="비밀번호"onChange={e
=>setpw(e.target.value)}/>
            <button onClick={signin}>회원가입</button>
        </div>
}
export default Signin;
```

```
app.post("/signin", async(req, res) =>{
    try{
```

```
        constid, pw=req.body;
        const jungbok =await userdata.findOne({ id});
        if(jungbok) {
              return res.status(400).json({ message:"아이디가 이미 존재합니
다."});
}
        const signin =new userdata({ id, pw});
        await signin.save();
        res.status(201).json({ message:"회원가입에 성공하였습니다"});
} catch(error) {
        console.error(error);
        res.status(500).json({ message:"서버 오류 발생"});
}
});
```

각각 singin.jsx와 index.js의 추가된 코드이다. siginin.jsx에서는 usestate를 활용해 id,pw를 다룬다. sigin 함수에서는 fetch를 통해 5001/signin으로 post요청을 보낸다. ui에서는 버튼이 눌리면 함수가 실행이 될 수 있도록 하였다. index.jsx에서는 req.body를 통해 id와 pw의 데이터를 가져와 할당한다. 이후 userdata에서 id가 중복되는지 확인하고, 중복이 아닐 시 userdata에 id와 pw를 입력하고, 회원가입에 성공했다고 알림을 보낸다. 한번 돌려보니 회원가입에 성공했다고 뜨긴 한다. 잘 추가되었는지 확인하기 위해

```
userdata.find().then(users =>{
    console.log( users);
}).catch(error =>{
    console.error( error);
});
```

를 입력해 터미널을 확인해보니

```
[
  {
    _id: new ObjectId('683aa8911910d7b5201457b4'),
    id: 'fury48',
    pw: '1122',
    __v: 0
  }
]
```

라고 잘 출력이 됐다. 우선은 commit을 하겠다.

위의 방식으로 하니, app.jsx의 메인화면과 sigin.jsx의 ui가 겹쳐보인다.
원인이 무엇인가 찾아보니, app.jsx의 내용과 signin.jsx의 내용이 동시에 랜더링 되도록 코드가 작성되어있다. 이것을 해결할 방법을 찾아보니, 다른 몇몇 사람들은 app과 main이외에 여러 jsx파일을 만들고, 해당 컴포넌트들을 app에서 라우팅, main에서 랜더링 하는 방식으로 사용하는 것 같다. 따라서 나도 이러한 방식으로 옮겨가기로 했다.

우선 lobby.jsx에 기존 app의 코드를 전부 복붙한뒤, app의 코드를

```
function App() {
  // useState 변수 사용하는 자리 ex) const [count, setCount] = useState(0)
  return(
    <Router>
      <Routes>
        <Route path="/"element={<Lobby />}/>
        <Route path="/signin"element={<Signin />}/>
      </Routes>
    </Router>
);
}
```

로 바꾸어주니

chunk-DQRVZFIR.mjs:188 Uncaught Error: You cannot render a inside another . You should never have more than one in your app.

중복 router에러가 생겼다. 왜그런가 보니 lobby에 기존의 app의 코드를 복붙하며 router코드도 그대로 들어가서 그랬다. 그것을 고치니 잘 나온다. commit을 하겠다.

이후 rout 바로 위에 헤더처럼

```
<h1>CHAT</h1>
      <div>
        <Link to="/lobby">로비 </Link>
        <Link to="/signin">회원가입 </Link>
        <Link>로그인</Link>
      </div>
```

을 추가해주었다. 이렇게 하면 모든 페이지에서 공통된 제목과 하이퍼링크를 가질 수 있기 때문이다. localhost를 확인해보니 잘 나온다. commit을 하겠

다.

6/1: 로그인 구현

이제 기본적인 ui와 페이지이동, db구축과 회원가입을 통한 컬렉션에 유저정보 저장을 구현하였으니, 이제 로그인 및 로그인 상태 유지를 완성시키려 한다.

리서칭을 해보니, 예상대로 토큰을 활용한 방식을 사용하는 것 같다. npm install jsonwebtoken을 입력해 설치했다. 이후 기존의 sigin.jsx와 다양한 자료들을 참고하며 짜본 코드는 다음과 같았다.

```
importuseStatefrom "react";
importuseNavigatefrom "react-router-dom";
function Login() {
    const[id, setid] =useState("");
    const[pw, setpw] =useState("");
    const nav =useNavigate();
    const login =async() =>{
        const res =await fetch("http://localhost:5001/login", {
            method:"POST",
            headers:"Content-Type":"application/json"},
            body:JSON.stringify({ id, pw}),
});
        const data =await res.json();
        if(res.status ==200) {
            localStorage.setItem("token", data.token);
            alert("로그인 되었습니다.");
            nav("/");
} else{
            alert(data.message);
}
    return(
        <div>
                        <input   type="text"placeholder="아이디"onChange={e
=>setid(e.target.value)}/>
                    <input  type="password"placeholder="비밀번호"onChange={e
=>setpw(e.target.value)}/>
            <button onClick={login}>로그인</button>
        </div>
```

```
}
export default Login;
```
login.jsx의 코드이다. 로그인 완료시 메인으로 페이지을 옮기기 위해 usenavigate를 import 해왔다. 이후 signin에서 그랬듯 post방식으로 5001 포트에 json을 보냈다. 이후 성공적이라는 의미로 응답상태가 200이 되면 토큰을 localstorage에 저장하여 로그인 정보를 유지시킨다.

```
app.post("/login", async(req, res) =>{
    try{
        constid, pw=req.body;
        const user =await userdata.findOne({ id, pw});
        if(!user) {
            return res.status(400).json({ message:"아이디 또는 비밀번호가 올
바르지 않습니다."});
}
        const token =jwt.sign({ id:user.id});
        res.status(200).json({ message:"로그인 성공!", token});
} catch(error) {
        console.error(error);
        res.status(500).json({ message:"서버 오류 발생"});
}
});
```
index.js에 추가시킨 코드이다. 역시 signin의 url 코드를 따왔고, 수정 및 추가하여 완성시켰다. jwt사인을 통해 아주 간단하게 토큰을 만들었으며, 비밀키와 유지 시간등도 추가 가능하다고 보았지만 우선은 구현을 목적으로 최대한 간단하게 만들었다. 이후 실행을 해보니
secretOrPrivateKey must have a value 오류가 떳다... 어쩔수 없이 비밀키를 추가해보니 로그인이 되었다고 뜬다. commit을 하겠다.

만약 로그인이 되었다면 로그인이 된 아이디를 불러와 활용해야 한다. 이것에 대해 서칭을 해보니 jwt는 기본적으로 암호화가 되어있어 이것을 decode해야 사용할 수 있다고 뜬다. 그래서 app.jsx에 `import{jwtDecode} from "jwt-decode";`
를 추가했다.
이후

```
const token =localStorage.getItem("token");
  if(token) {
```

```
    const decoded =jwtDecode(token);
    console.log(decoded.id);
}
```
를 추가해 개발자도구에서 콘솔을 확인해보니 로그인 후 정상적으로 id가 출력되는 것을 확인했다. commit하겠다.

6/2: 채팅 개발

이제 채팅을 구현하면 된다. 우선 소켓 연결을 위해 다음과 같은 코드를 작성했다.

```
//소켓 연결
const server =http.createServer(app)
const io =new Server(server, {
    cors:{
        origin:"*"
}
})
io.on("connection", (socket)=>{
    console.log('Connected to Server');
    socket.on('disconnect',()=>{
        console.log('Disconnected');
})
```
이후 lobby.jsx에서 채팅을 입력하면 표시가 되도록 구현하고자 한다.

우선 1차적으로 구현한 코드는 다음과 같다.

```
const[chat,setChat] =useState([]);
  const[user,setUser] =useState(localstorage.getItem(token));
  const[input,setInput] =useState("");
  useEffect(()=>{
    socket.on("chat",(message)=>{
      setChat((prevChat) =>[...prevChat,message]);
});
    return() =>{
      socket.off("chat");
},[]);
  const sendChat =() =>{
    if(input.trim()) {
      socket.emit("chat",{user,message:input});
      setInput("");
}
```

```
};
  return(
    <>
    <br></br>
    <div className='chat-log'>
      {chat.map((msg,index)=>(
        <p key={index}>
          <strong>{msg.user}</strong>: {msg.message}
        </p>
))}
    </div>
    <div className='chat-input'>
                                                                <input
type='text'value={input}onChange={(e)=>setInput(e.target.value)}></input>
      <button onClick={sendChat}>전송</button>
    </div>
    </>
);
}
```

토큰을 통해 유저의 정보를 가져온 후, useState를 이용해 유저의 채팅정보와 입력값을 관리하도록 하였다. 이후 useEffect를 사용하여 서버에서 채팅 메시지를 수신할 때 마다 setChat(prevChat) => [...prevChat,message]를 호출해 기존의 채팅목록에 새 메시지를 추가하도록 하였다. sendChat함수에서는 socket.emit을 통해 서버로 입력한 메시지를 전송하며, 입력 후 입력input을 비운다.

이후 jsx hmtl?에서는 chat log 아래에서 chat.map((msg,index))를 통해서 chat 배열을 반복적으로 랜더링해 채팅을 업데이트 하도록 하였다. input을 통해 채팅을 입력받고, 버튼을 누를시 sendChat을 호출하도록 하였다.

이러고 실행을 해보니, 두개의 문제가 발생했다.

Failed to load resource: the server responded with a status of 404 (Not Found)

GET http://localhost:5001/socket.io/?EIO=4&transport=polling&t=dh72rmdx 404 (Not Found)

왜그런가 서칭을 해보니, index.js에서 app.listen이 맨 아래에 있지 않아 그런것 같다. 또한 socket.io를 사용할 시 app.listen보다는 server.listen을

사용하는 것이 좋다고 하여 수정하고 돌려보니 오류는 사라졌다. 이후 채팅을 입력하니
Qw1hji123980213udhiidsjk128di1293012390DSAD: Hello
이렇게 뜬다. 생각을 해보니, JWT는 기본적으로 인코딩이 되는데, 그것을 디코딩하지 않고 바로 token으로 가져와 아이디가 인코딩되어있는 채로 출력이 된 것 같다. 이것을 해결하고자 코드를 다음과 같이 일부 수정하였다.

```
const socket =io("http://localhost:5001");
const token =localStorage.getItem("token");
let Id =null;
function Lobby() {



    if(token) {
      const decoded =jwtDecode(token);
      Id =decoded.id;
}
  // useState 변수 사용하는 자리 ex) const [count, setCount] = useState(0)
  const[chat,setChat] =useState([]);
  const[user,setUser] =useState(Id);
  const[input,setInput] =useState("");
  useEffect(()=>{
```

ID라는 전역변수를 선언한 이후, token의 id를 decoding한 값을 ID에 저장하게 하였다. 이렇게 하니 정상적으로 이름이 출력됐다!

이후 인터넷 익스플로러와 크롬을 통해 돌려보니, 정상적으로 채팅이 올라오는 것을 확인하였다. commit을 하겠다!

회원가입을 하고 난 후 자동으로 로그인 화면으로 이동하면 좋겠다는 생각이 들어

```
        if(res.status ==201) {
            alert("로그인 되었습니다.");
            nav("/login");
} else{
            alert(data.message);
}
```

해당 코드를 추가하니 회원가입 이후 잘 이동하였다. commit을 하겠다.

6/4: 로그아웃 시스템 구현

로그아웃 시스템을 구현하고자 한다. 우선 app.jsx에 헤더가 있기에 로비 메뉴 로그인 옆에 로그아웃을 넣겠다. 우선은 다음과 같이 코드를 작성하였다.

```
 const logout =() =>{
    localStorage.removeItem("token");
    window.location.href ="/lobby";
};
  return(
    <>
    <Router>
      <h1>CHAT</h1>
      <div>
        <Link to="/lobby">로비 </Link>
        <Link to="/login">로그인 </Link>
        <Link to="/signin">회원가입 </Link>
        {token && <button onClick={logout}>로그아웃</button>}
```

이렇게 해보니, 로그아웃 버튼이 있었고, 클릭을 하니 토큰이 삭제되었다.

그러나 문제는, 다시 로그인을 하였을 때는 로그아웃 버튼이 뜨지 않는 것이다. 뭐가 문제인가 생각을 해보니, 지속적으로 렌더링이 되는 와중 토큰을 usestate로 관리하지 않아 토큰의 현황이 엡데이트 되지 않아 그러는 것 같다. 따라서 다음과 같이 코드를 수정하였다.

```
const[token, setToken] =useState(localStorage.getItem("token"));
  useEffect(() =>{
  if(token) {
    const decoded =jwtDecode(token);
    console.log(decoded.id);
}
}, [token]);

  const logout =() =>{
    localStorage.removeItem("token");
    setToken(null);
    window.location.href ="/lobby";
};
```

이렇게 하니 정상적으로 작동하였다. commit을 하겠다.

또한 하나의 문제점이 더 발생하였다. 이미 회원가입이 되어있는 아이디로 회

원가입을 다시 시도하니,

POST http://localhost:5001/signin 400 (Bad Request)
이라 뜬다. 분석을 해보니, 만약 중복이 탐지될 시 400을 반환하지만 해당 반환을 받아주는 코드가 없어 그런것 같다. 따라서 코드를 다음과 같이 수정하였다.

```
if(res.status ==201) {
            alert("로그인 되었습니다.");
            nav("/login");
} else if(res.status ==400){
            alert("이미 존재하는 아이디입니다.")
} else{
            alert(data.message);
}
```

이렇게 수정하니, 정상적으로 alert가 뜬다. commit을 하겠다.

6/5일
만약 로그인이 안 되었을 시에는 현재는 채팅에 그냥 빈칸으로 뜨는데, 이때 익명으로 뜨게 수정을 하겠다.

```
    const[Id, setId] =useState("");
    const[chat,setChat] =useState([]);
    const[user,setUser] =useState("");
    const[input,setInput] =useState("");

  useEffect(() =>{
    const token =localStorage.getItem("token");
    if(token) {
      const decoded =jwtDecode(token);
      setId(decoded.id);
} else{
      setId("익명");
}
}, []);
  useEffect(() =>{
    setUser(Id);
}, [Id]);
```

다음과 같이 lobby.jsx를 수정하였다. Id를 전역변수로 한번 선언하는 것이 아

닌, usestate를 통해 꾸준히 관리해주고, useEffect를 통해 지속적으로 업데이트가 되어 렌더링이 되도록 하였다. 실행을 해보니 정상적으로 작동한다. commit을 하겠다.

밤새 도커 환경을 설정하려고 싸운 끝에 겨우 도커 환경을 설정하였다. docker-compose up —build를 입력하면 정상적으로 각 컨테이너들이 실행이 된다. 이를 위 몇몇 코드들 및 config를 수정하기도 했다. 그러나 왜인지 모르겠지만 기존에는 없던 여러 버그들이 터지기 시작했다. 제일 큰 것은, 로그인을 해서 lobby로 이동이 되게 되면 왜인지 모르게 로그아웃 버튼이 뜨지을 않는다. 원래는 이러지 않았는데, 어디서 문제가 된것인지 감이 안잡힌다. 우선은 commit을 하겠다.