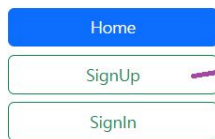
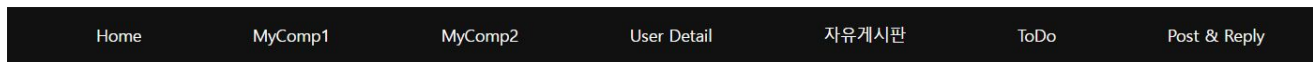


4_1. 웹서비스 아키텍처 (React.js와 Node.js연동 실습)

백성애 2025-02-15



SignUp.jsx - 회원가입 컴포넌트 구성하기



SignUp.jsx

Signup

이름:

이름을 입력해야 해요

Email:

Email을 입력해야 해요

비밀번호:

비밀번호를 입력해야 해요

비밀번호 확인:

비밀번호가 일치하지 않아요

☐ 이용약관에 동의합니다.
이용약관 동의에 체크해야 해요

회원가입 다시쓰기

LoginModal.jsx - 로그인 모달 폼 구현하기

Side 또는 Header

부분에 “SignIn”메뉴를

추가하고

해당 메뉴를 클릭시

LoginModal 컴포넌트를

띄워보자.

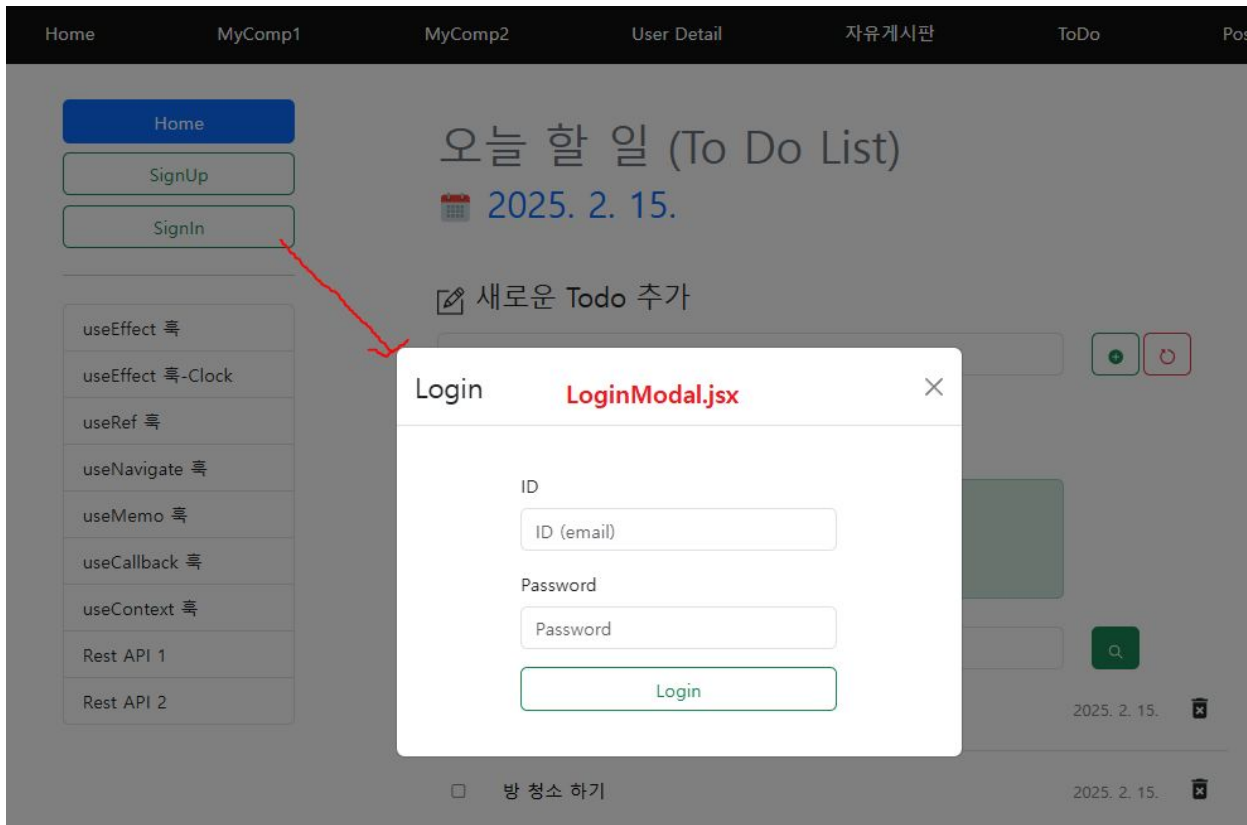
그러기 위해선 최상위 부모

App.js에서 모달을 띄우기

위한 state=> showLogin을

주고 이를 LoginModal에 props

로 전달 하자



로그인 모달 띄우기 위한 App.js에 state추가

Side 컴포넌트에
SignIn버튼이 있으므로
해당 버튼을 클릭할 때
showLogin state값을
변경하기 위해
onShowLogin이란
props로
onChangeShowLogin함
수를 전달하자

```
23 import SignUp from './components/member/SignUp';
24 import LoginModal from './components/member/LoginModal';
25 import { useState } from 'react';
26 function App() {
27   const [showLogin, setShowLogin] = useState(false);
28   const onChangeShowLogin = (bool) => {
29     //alert(bool);
30     setShowLogin(bool);
31   };
32   return (
33     <div className="container py-5">
34       <BrowserRouter future={{ v7_relativeSplatPath: false }}>
35         {/* BrowserRouter로 앱 전체를 감싸야 라우팅 기능을 사용할 수 있다 */}
36         <Container>
37           <Row>
38             <Col>
39               <Header />
40             </Col>
41           </Row>
42           <Row>
43             <Col xs={12} sm={3} md={4} lg={4} className="d-none d-sm-block mt-3">
44               {/* d-none: 모든 화면에서 숨겨지고 d-sm-block => small사이즈부터 보여준다 */}
45               <Side onShowLogin={onChangeShowLogin} />
46             </Col>
47             <Col xs={12} sm={9} md={8} lg={8}>
48               <LoginModal show={showLogin} setShow={setShowLogin} /> {/* 모달 상태 전달 */}
49             </Col>
50           </Row>
51         </Container>
52       </BrowserRouter>
53     </div>
54   );
55 }
```

Side.jsx 에 로그인버튼 (SignIn) 이벤트 처리

```
1 // Side.jsx
2 import React from 'react';
3 import { Stack, Button, ListGroup } from 'react-bootstrap';
4 import { Link } from 'react-router-dom';
5
6 export default function Side({ onShowLogin }) {
7   return (
8     <Stack gap={2} className="col-md-6 mx-auto">
9       { /* <Link to="/">Home</Link> */ }
10      <Button variant="primary" as={Link} to="/">
11        Home
12      </Button>
13      <Button variant="outline-success" as={Link} to="/signup">
14        SignUp
15      </Button>
16      <Button
17        variant="outline-success"
18        onClick={() => {
19          onShowLogin(true);
20        }}
21      >
22        SignIn
23      </Button>
24      <hr></hr>
25      <ListGroup>
26        <ListGroup.Item as={Link} to="/hook1">
27          useEffect 훅
```

LoginModal.jsx

```
1 import { Modal, Button, Row, Col, Form } from 'react-bootstrap';
2 const LoginModal = ({ show, setShow }) => {
3   return (
4     <>
5       {/* 로그인 모달 */}
6       <Modal show={show} onHide={() => setShow(false)} centered>
7         <Modal.Header closeButton>
8           <Modal.Title>Login</Modal.Title>
9         </Modal.Header>
10        <Modal.Body>
11          <Row className="LoginForm">
12            <Col className="p-4 mx-auto" xs={10} sm={10} md={8}>
13              <Form>
14                <Form.Group className="mb-3">
15                  <Form.Label>ID</Form.Label>
16                  <Form.Control type="text" id="email" name="email" placeholder="ID (email)" />
17                </Form.Group>
18                <Form.Group className="mb-3">
19                  <Form.Label>Password</Form.Label>
20                  <Form.Control type="password" id="pwd" name="pwd" placeholder="Password" />
21                </Form.Group>
22                <div className="d-grid gap-2">
23                  <Button type="submit" variant="outline-success">
24                    Login
25                  </Button>
26                </div>
27              </Form>
28            </Col>
29          </Row>
30        </Modal.Body>
31      </Modal>
32    </>
33  );
34 }
35 export default LoginModal;
```

MySQL - Database 구축

회원 테이블 생성 DDL문장

```
create database eduDB;  
use eduDB;  
show tables;  
drop table if exists members;
```

```
create table if not exists members(  
    id int primary key auto_increment,  
    name varchar(30) not null,  
    email varchar(50) not null unique,  
    passwd varchar(20) not null,  
    indate date default (current_date()),  
    refreshtoken varchar(512) default null  
);  
desc members;  
select * from members;
```

refreshToken의 경우 회원 인증시 사용하는 컬럼으로
원칙상

별도의 테이블을 만들어 관리하는 것이 좋다.

하지만 이럴 경우 복잡도가 증가한다.

우리는 학습을 위해 좀 단순화 시켜 members테이블에
포함하도록 한다.

별도로 refresh_tokens 테이블을 만든다면 아래 ddl문장을
참고하자

```
CREATE TABLE refresh_tokens (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    user_id INT NOT NULL,  
    token VARCHAR(512) NOT NULL,  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    FOREIGN KEY (user_id) REFERENCES members(id)  
    ON DELETE CASCADE  
);
```



회원 데이터 삽입후 조회

```
insert into members(name,email,passwd) values('홍철 수','hong@a.b.c','111');
```




```
insert into members(name,email,passwd) values('김철수','kim@a.b.c','111');
```



```
select * from members;
```


Result Grid

Filter Rows:

Edit:




Export/Import:



Wrap Cell Content: 

	id	name	email	passwd	indate	refreshtoken
▶	1	홍철수	hong@a.b.c	111	2025-02-15	NULL
	2	김철수	kim@a.b.c	111	2025-02-15	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL

members 16 ×

JWT를 이용한 로그인 인증 처리

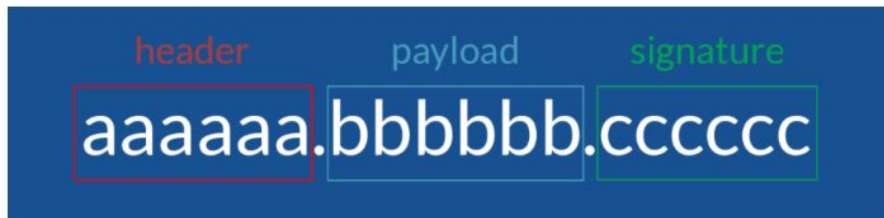
JWT란?

JWT는 **Json Web Token**의 약자로 일반적으로 **클라이언트와 서버 사이에서 통신할 때 사용자 인증과 정보교환을 위해 사용되는 토큰**을 의미한다

JWT는 JSON 형식의 데이터를 **Base64 URL-safe** 방식으로 인코딩한 문자열로, **서버와 클라이언트 간에 안전하게 정보를 전달**하는 데 사용된다

JWT의 구성요소

JWT는 **헤더(header)**, **페이로드(payload)**, **서명(signature)** 세 파트로 나뉘져 있으며, 아래와 같은 형태로 구성되어 있다.



JWT

Algorithm

HS256

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  "alg": "HS256",  "typ": "JWT"}
```

PAYLOAD: DATA

```
{  "sub": "1234567890",  "name": "John Doe",  "iat": 1516239022}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
) ☐ secret base64 encoded
```

1 Header (헤더)

어떤 알고리즘과 타입을 사용할지 정의하는 부분이야.

예제:

```
json
{
  "alg": "HS256", // HMAC SHA256 알고리즘 사용
  "typ": "JWT"    // 토큰 타입이 JWT임
}
```

2 Payload (페이로드, 내용)

사용자의 정보나 추가 데이터를 포함하는 부분이야.

하지만 암호화되지 않아서 노출되면 안 되는 정보는 넣으면 안 돼!

예제:

```
json
{
  "userId": 123,
  "role": "admin",
  "exp": 1698000000 // 만료 시간 (Unix Timestamp)
}
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJyMywiZXhwIjoxNjk4MDAwMDAwfQ.Sf1KxwRJSMeKKI
```

JWT는 **헤더 (Header)**, **페이로드 (Payload)**, **서명 (Signature)** 세 부분으로 구성된다.
각각을 **.(점)**으로 구분한 형태이다.

3 Signature (서명)

토큰의 무결성을 보장하기 위해 **비밀 키를 사용해서 생성된 서명**이야.

헤더 + 페이로드를 **비밀 키**로 **해싱해서 만들어**.

예제 (HMAC SHA256 알고리즘 사용)

```
SCSS
HMACSHA256(
  base64UrlEncode(header) + "." + base64UrlEncode(payload),
  secretKey
)
```

JWT 인증 과정 (예제)

1 사용자가 로그인 요청

→ 아이디, 비밀번호 입력 후 서버에 요청

2 서버에서 JWT 생성

→ 사용자의 정보를 담은 JWT를 생성하고 클라이언트에 반환

3 클라이언트에서 JWT 저장

→ `localStorage`, `sessionStorage`, `cookie` 등에 저장

4 요청할 때 JWT 포함

→ 이후 API 요청마다 JWT를 **Authorization** 헤더에 담아서 전송

http

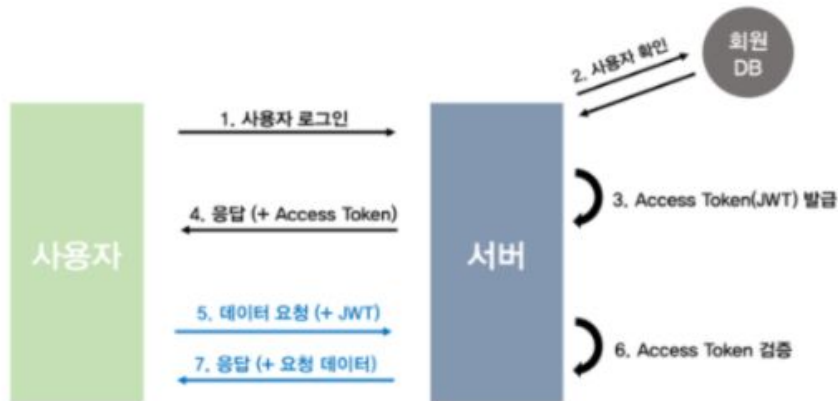
Authorization: Bearer <JWT>

5 서버에서 JWT 검증 후 응답

→ 서명이 유효하면 요청을 처리하고, 아니면 401 Unauthorized 응답

JWT 동작 원리

1. 사용자가 id와 password를 입력하여 로그인 요청을 한다.
2. 서버는 회원DB에 들어가 있는 사용자인지 확인을 한다.
3. 확인이 되면 서버는 로그인 요청 확인 후, secret key를 통해 토큰을 발급한다.
4. 이것을 클라이언트에 전달한다.
5. 서비스 요청과 권한을 확인하기 위해서 헤더에 데이터(JWT) 요청을 한다.
6. 데이터를 확인하고 JWT에서 사용자 정보를 확인한다.
7. 클라이언트 요청에 대한 응답과 요청한 데이터를 전달해준다.



<https://velog.io/@hahan/JWT%EB%9E%80-%EB%AC%B4%EC%97%87%EC%9D%B8%EA%B0%80>

이와 같이 토큰 기반 인증방식은 사용자의 인증이 완료된 이후에 토큰을 발급한다.

클라이언트쪽에서는 전달받은 토큰을 저장해두고 서버에 요청을 할 때마다 해당 토큰을 서버에 함께 전달한다. 그 이후 서버는 토큰을 검증하고 응답하는 방식으로 작동한다.

JWT와 Session 기반 인증 차이점

JWT vs Session 기반 인증

비교항목	JWT 기반 인증	세션 기반 인증
서버 부담	서버에서 세션 저장 필요 없음	세션 저장해야 해서 부담 증가
확장성	여러 서버에서 사용 가능 (무상태)	서버에 세션 저장해야 해서 확장 어려움
보안성	토큰 노출 시 위험 → HTTPS 필수	서버에서 세션 관리 가능
인증 속도	빠름 (토큰만 검증)	느림 (DB 조회 필요)
로그아웃 처리	어려움 (강제 만료 어려움)	가능 (세션 삭제하면 됨)

서버단

토큰 생성 함수 구성

```
...  
const jwt = require('jsonwebtoken'); // JWT 라이브러리  
  
//JWT(JSON Web Token) 토큰 생성 함수  
const generateToken = (user, secret, expiresIn) => {  
  return jwt.sign(user, secret, { expiresIn });  
  /*secret은 JWT(JSON Web Token)를 암호화하여 서명하는 데 사용되는 비밀 키  
  secret은 서명을 생성하는 데 사용되며, 이 서명을 통해 토큰이 변조되지 않았음을 보장한다  
  내부적으로 암호화( HMAC SHA256 같은... ) 알고리즘을 사용해 서명을 생성한다  
  https://jwt.io/ 사이트에서 확인해보자  
  */  
};  
...
```

서버단

- 로그인요청 처리

로그인 요청이 들어왔을 때 DB에서
사용자가 맞는지 확인한 뒤 맞다면
accessToken과 refreshToken을
생성하여 회원정보(name,email)와
함께 token들을 응답으로 보낸다

```
//로그인 요청시
app.post('/api/login', async (req, res) => {
  const { email, passwd } = req.body;
  console.log(`email ${email}, passwd ${passwd}`);
  const sql = 'SELECT id, name, email FROM members WHERE email=? AND passwd=?';
  try {
    const [results] = await pool.query(sql, [email, passwd]);
    if (results.length > 0) {
      const user = results[0];
      // access token과 refresh token 생성
      const accessToken = generateToken(
        { id: user.id, email: user.email, name: user.name },
        process.env.ACCESS_SECRET,
        '15m' // access token 유효시간 15분
      );
      const refreshToken = generateToken(
        { id: user.id, email: user.email },
        process.env.REFRESH_SECRET,
        '1d' // refresh token 유효시간 (1일)
      );
      // 새로 생성된 refreshToken을 members 테이블에 업데이트
      const updateSql = 'UPDATE members SET refreshToken = ? WHERE email = ?';
      await pool.query(updateSql, [refreshToken, email]);
      res.json({
        result: 'success',
        data: { email: user.email, name: user.name },
        accessToken,
        refreshToken, // 클라이언트에 반환
      });
    } else {
      res.status(401).json({ result: 'failure', message: 'Invalid email or password' });
    }
  } catch (err) {
    console.error('Error executing query:', err);
    res.status(500).json({ error: 'Database error' });
  }
});
```


서버단 - accessToken 검증 미들웨어

```
// 인증이 필요한 API 예제
app.get('/api/protected', verifyAccessToken, (req, res) => {
  res.json({ user: req.user });
});
```

인증이 필요한 API 요청 (/api/protected)

- 클라이언트는 요청 헤더에 **accessToken**을 포함해서 요청 보냄
- GET /api/protected
- Authorization: Bearer ACCESS_TOKEN

verifyAccessToken 미들웨어가 **accessToken**을 검증하여 유효하면 요청을 통과하고 **user**정보 반환.

유효하지 않으면 403응답

→ 클라이언트가 **refreshToken**을 이용해서 새 **accessToken** 요청.

```
const verifyAccessToken = (req, res, next) => {
  const token = req.headers['authorization']?.split(' ')[1]; // ? (optional chaining)
  //클이 요청을 보낼 때 요청 헤더에 아래와 같은 형태로 보낸다 토큰값만 추출하기 위해 공백으로 분리
  //Authorization: Bearer 토큰값
  //"Bearer abcdefg12345".split(' ') => 결과: ["Bearer", "abcdefg12345"]

  console.log('검증 미들웨어 토큰: ', token);

  if (!token) return res.status(401).json({ message: 'Access Token required' });

  jwt.verify(token, process.env.ACCESS_SECRET, (err, decoded) => {
    if (err) {
      console.log('토큰 검증 실패 verify fail');
      return res.status(403).json({ message: 'Invalid Access Token' });
    }
    req.user = decoded; // 토큰 정보 저장
    console.log(req.user); //// JWT 토큰의 payload에 포함된 사용자 정보를 req.user에 저장

    next();
  });
};
```

인증 실패시 → 새 accessToken 발급 (/api/refresh)

accessToken이 만료되었거나 유효하지 않으면, 클라이언트는 refreshToken을 보내서 새로운 accessToken 요청.

```
POST /api/refresh
```

```
Content-Type: application/json
```

```
{  
  "refreshToken": "EXISTING_REFRESH_TOKEN"  
}
```

서버는:

1. refreshToken 검증
2. DB에서 해당 사용자 정보 조회
3. 새 accessToken 발급 후 클라이언트에 반환.

클라이언트는:

- 새로운 accessToken을 받아서 API 요청을 다시 시도.

서버단 -refreshToken 검증

accessToken만료
시
클라이언트쪽에
서는
refreshToken을
서버쪽에 보내
검증을 받는다.

검증에 성공하면
서버는 다시
새로운
accessToken을
발급한다

```
app.post('/api/refresh', async (req, res) => {  
  const { refreshToken } = req.body;  
  
  if (!refreshToken) return res.status(401).json({ message: 'Refresh Token 필요합니다' });  
  
  // refreshToken 검증  
  jwt.verify(refreshToken, process.env.REFRESH_SECRET, async (err, decoded) => {  
    if (err) {  
      return res.status(403).json({ message: 'Invalid Refresh Token' });  
    }  
  
    // DB에서 해당 user 정보 확인  
    const sql = 'SELECT id, email FROM members WHERE id = ?';  
    const [results] = await pool.query(sql, [decoded.id]);  
  
    if (results.length === 0) {  
      return res.status(403).json({ message: 'User not found' });  
    }  
  
    const user = results[0];  
  
    // 새 AccessToken 발급  
    const newAccessToken = generateToken({ id: user.id, email: user.email }, process.env.ACCESS_SECRET, '15m');  
  
    res.json({ accessToken: newAccessToken });  
  });  
});
```

서버단 - 로그아웃 요청 (/api/logout)

클라이언트가 로그아웃을 원하면,
`refreshToken`을 서버에서 제거 (DB에서 `NULL`로 업데이트).

이후 해당 `refreshToken`은 더 이상 유효하지 않음.

- 즉, 다시 `accessToken`을 갱신할 수 없음 → 재로그인 필요.

```
app.post('/api/logout', async (req, res) => {
  const { email } = req.body;

  // refreshToken을 NULL로 업데이트
  const sql = 'UPDATE members SET refreshToken = NULL WHERE email = ?';
  try {
    await pool.query(sql, [email]);
    res.json({ message: 'Logged out successfully' });
  } catch (err) {
    console.error('로그아웃 처리 중 오류:', err);
    res.status(500).json({ message: '로그아웃 처리 실패' });
  }
});
```

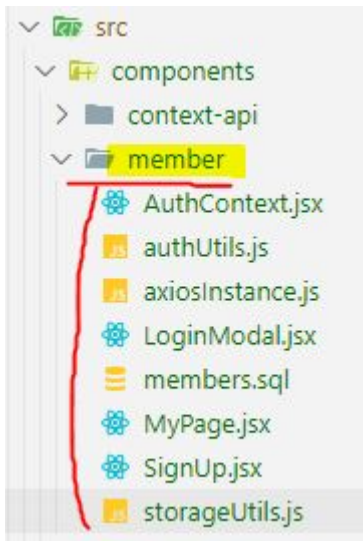
Database- members 테이블 확인

회원인증시 발급받은 refreshtoken이 저장된 것을 확인할 수 있다. (비밀번호를 암호화해서 넣어야 하나 암호화 처리는 나중에 하도록 하자)

Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:						
	id	name	email	passwd	indate	refreshtoken
▶	1	홍철수	hong@a.b.c	111	2025-02-15	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSV
	2	김철수	kim@a.b.c	111	2025-02-15	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiV
*	NULL	NULL	NULL	NULL	NULL	NULL

클라이언트단 - React

일단 인증과 관련 모듈 구성을 member 폴더 안에서 다 하고 테스트 완료후 재구성하도록 하자



react

storageUtils.js

best-app > src > components > member > storageUtils.js > ...

```
1 //토큰 가져오거나 저장하는 유틸
2 export const getAccessToken = () => {
3   return sessionStorage.getItem('accessToken');
4 };
5
6 export const getRefreshToken = () => {
7   return localStorage.getItem('refreshToken');
8 };
9
10 export const setAccessToken = (token) => {
11   sessionStorage.setItem('accessToken', token); //세션스토리지에 저장 15분
12 };
13
14 export const setRefreshToken = (token) => {
15   localStorage.setItem('refreshToken', token); //로컬스토리지에 저장 1일
16 };
17
18 export const removeTokens = () => {
19   localStorage.removeItem('accessToken');
20   localStorage.removeItem('refreshToken');
21 };
22
```

react

authUtils.js

best-app > src > components > member > authUtils.js > ...

```
1  import axios from 'axios';
2  import { getRefreshToken } from './storageUtils'; // localStorage에서 refreshToken 가져오는 함수
3
4  export const refreshAccessToken = async () => {
5      const refreshToken = getRefreshToken();
6
7      if (refreshToken) {
8          try {
9              const response = await axios.post('http://localhost:7777/api/refresh', { refreshToken });
10
11              const { accessToken } = response.data;
12
13              // 새로운 accessToken을 반환
14              return accessToken;
15          } catch (error) {
16              console.error('refresh token error: ', error);
17              return null;
18          }
19      }
20
21      console.log('No refresh token found');
22      return null;
23  };
24
25  // JWT 토큰의 만료 여부를 확인하는 함수
26  export const checkTokenExpiration = (token) => {
27      const payload = JSON.parse(atob(token.split('.')[1]));
28      const expirationTime = payload.exp * 1000; // exp는 초 단위로 저장됨
29      console.log('checkTokenExpiration: expirationTime=>' + expirationTime);
30
31      return expirationTime < Date.now();
32  };
33
```


react -axiosInstance.js

best-app > src > components > member > axiosInstance.js > ...

```
1 import axios from 'axios';
2 import { refreshToken, checkTokenExpiration } from './authUtils'; // refreshToken 갱신 함수
3 import { getAccessToken } from './storageUtils'; // localStorage에서 accessToken 가져오는 함수
4 /**토큰 관리-----
5 [1] 토큰 갱신 전략: accessToken이 만료되었을 때 refreshToken을 사용하여
6     새로운 accessToken을 발급받아 로그인 상태를 유지합니다.
7 [2] axios 인터셉터를 사용하여 서버의 응답을 가로채고, 만약 accessToken이 만료되었다면
8     refreshToken을 사용해 새 토큰을 요청하도록 합니다.
9 [3] 로그인 상태 종료 처리: accessToken이 만료된 경우, 로그아웃 상태로 처리하고
10     로그인 페이지로 리디렉션하거나, 새 accessToken을 받아서 기존 요청을 재시도합니다.
11 * -----
12 * axiosInstance를 사용하여 각 컴포넌트에서 별도로 토큰 유효성 검사를 처리하지 않고, 중앙에서 관리해보자
13 * 모듈화된 **axiosInstance**를 사용하여, 모든 API 요청 시 accessToken 만료 체크 및 갱신이 자동으로 처리된다.
14 * 인터셉터가 글로벌하게 설정되어 있기 때문에, 별도의 요청 처리 코드에서 토큰 관리 로직을 신경 쓸 필요가 없다
15 * => axiosInstance를 이용해 요청을 보낼 때마다 인터셉터가 자동으로 토큰 유효 시간 체크를 하게 됨
16 *
17 * 인증이 필요없는 요청에서는 그냥 axios를 사용하고
18 * 인증이 필요한 요청에서는 axiosInstance를 사용하자
19 */
20 // axios 인스턴스 생성
21 const axiosInstance = axios.create({
22   timeout: 10000, // 타임아웃 설정
23 });
24 // 요청 인터셉터 설정
25 axiosInstance.interceptors.request.use(
```

react -axiosInstance.js

```
20 // axios 인스턴스 생성
21 const axiosInstance = axios.create({
22   timeout: 10000, // 타임아웃 설정
23 });
24 // 요청 인터셉터 설정
25 axiosInstance.interceptors.request.use(
26   (config) => {
27     const accessToken = getAccessToken(); // localStorage.getItem('accessToken');
28     console.log('요청 인터셉터 실행됨....accessToken: ', accessToken);
29     if (accessToken) {
30       if (checkTokenExpiration(accessToken)) {
31         // 만료된 경우 리프레시 토큰으로 새로 발급받고 요청 헤더에 넣기
32         return refreshAccessToken().then((newAccessToken) => {
33           config.headers['Authorization'] = `Bearer ${newAccessToken}`;
34           return config;
35         });
36       }
37       config.headers['Authorization'] = `Bearer ${accessToken}`;
38     }
39     return config;
40   },
41   (error) => Promise.reject(error)
42 );
43
```

react-axiosInstance.js

```
43
44 // 응답 인터셉터 설정 (토큰 갱신)
45 axiosInstance.interceptors.response.use(
46   (response) => response, // 정상적인 응답 처리
47   async (error) => {
48     // 요청이 401(Unauthorized)로 실패할 경우
49     if (error.response && error.response.status === 401) {
50       // `accessToken` 만료된 경우 refreshToken을 이용해 갱신
51       const refreshToken = localStorage.getItem('refreshToken');
52
53       if (refreshToken) {
54         try {
55           // 리프레시 토큰을 사용해 새로운 액세스 토큰 요청
56           const { data } = await axios.post('http://localhost:7777/api/refresh', { refreshToken });
57
58           // 새로운 액세스 토큰 저장
59           const { accessToken } = data;
60           localStorage.setItem('accessToken', accessToken);
61
62           // 실패했던 요청을 새로운 액세스 토큰과 함께 재요청
63           error.config.headers['Authorization'] = `Bearer ${accessToken}`;
64           return axiosInstance(error.config); // 원래 요청을 재시도
65         } catch (refreshError) {
66           // 리프레시 토큰도 만료되었거나 오류가 발생하면 로그아웃 처리
67           localStorage.removeItem('accessToken');
68           localStorage.removeItem('refreshToken');
69           window.location.href = '/login'; // 로그인 페이지로 리디렉션
70           return Promise.reject(refreshError);
71         }
72       } else {
73         // refreshToken이 없으면 로그아웃 처리
74         localStorage.removeItem('accessToken');
75         localStorage.removeItem('refreshToken');
76         window.location.href = '/login'; // 로그인 페이지로 리디렉션
77         return Promise.reject(error);
78       }
79     }
80     // 그 외의 오류는 그대로 처리
81     return Promise.reject(error);
82   }
83 );
```

react

-LoginModal.jsx

```
1 import { Modal, Button, Row, Col, Form } from 'react-bootstrap';
2 import { useNavigate } from 'react-router-dom';
3 import { useState, useRef } from 'react';
4 import axios from 'axios';
5 import axiosInstance from './axiosInstance';
6 // import { useUser } from './AuthContext';
7 import { useContext } from 'react';
8 import { AuthContext } from './AuthContext';
9
10 const LoginModal = ({ show, setShow }) => {
11   const navigate = useNavigate();
12   const idRef = useRef(null);
13   const pwRef = useRef(null);
14
15   //로그인 폼에서 사용자가 입력한 값을 받을 state
16   const [loginUser, setLoginUser] = useState({ email: '', passwd: '' });
17   ///////////////////////////////////////////////////
18   const { loginAuthUser } = useContext(AuthContext); //인증받은 로그인 유저를 받는다
19   ///////////////////////////////////////////////////
20
21   const onChangeInput = (e) => {
22     console.log(e.target.name, e.target.value);
23
24     const { name, value } = e.target;
25     let tmp = { ...loginUser, [name]: value };
26     console.log('tmp: ', tmp);
27
28     setLoginUser(tmp);
29
30     /**폼에는 여러 개의 입력 필드(email, password 등)가 있는데,
31      * onChangeInput 하나만으로 다 처리하려면 어떤 필드가 변경됐는지
32      * 동적으로 알아야 한다. 그래서 [name]: value를 써서 입력 필드의 name을 키로 사용
33      * - 키(속성 이름) → [name]처럼 대괄호([])로 감싸야 동적으로 설정됨
34      * - 값(value) → 그냥 value 쓰면 됨
35      */
36   };
37 }
```

react-LoginModal.jsx

```
36     const onSubmitHandler = (e) => {
37       e.preventDefault();
38       const { email, passwd } = loginUser;
39       if (!email) {
40         alert('아이디를 입력하세요');
41         idRef.current.focus();
42         return;
43       }
44       if (!passwd) {
45         alert('비밀번호를 입력하세요');
46         pwRef.current.focus(); //Optional Chaining 이용해도 된다.
47         return;
48       }
49       loginRequest();
50     };
```


react

-LoginModal.jsx

```
51 const loginRequest = async () => {
52   try {
53     //const response = await axios.post('/api/login', loginUser);
54     const url = 'http://localhost:7777/api/login';
55     //const url = '/api/login';
56     // const response = await axios.post(url, loginUser);
57     // alert(url);
58     const response = await axiosInstance.post(url, loginUser);
59     alert(JSON.stringify(response.data));
60     const {
61       result,
62       data: { email, name },
63       accessToken,
64       refreshToken,
65     } = response.data;
66
67     if (result === 'success') {
68       //1. 토큰 저장하기 (localStorage)
69       sessionStorage.setItem('accessToken', accessToken); //세션 스토리지는 브라우저 탭이나 창이 닫히면 사라짐
70       localStorage.setItem('refreshToken', refreshToken);
71       //2. 로그인 상태 업데이트==> 부모App의 왼쪽 Side.jsx쪽에 "test님 로그인 중" 출력하고
72       // 로그인 버튼을 로그아웃 버튼으로 변경하려는데
73       //setUser({ email, name });//<== 이 경우는 props로 로그인 상태를 이용할 경우, /
74       const authUser = { email, name };
75       loginAuthUser(authUser); //Context Api통해 공급받은 loginAuthUser통해 전역상태 업데이트
76       //3. 메인 페이지로 이동
77       navigate('/');
78       //4. 로그인 모달 닫기
79       setShow(false);
80     } else {
81       alert('아이디 또는 비밀번호가 일치하지 않습니다. 다시 시도하세요');
82       setLoginUser({ email: '', passwd: '' });
83     }
84   } catch (err) {
85     alert('Error: ' + err);
86   }
87 };
```

react

-LoginModal.jsx

```
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122

return (
  <>
    { /* 로그인 모달 */ }
    <Modal show={show} onHide={() => setShow(false)} centered>
      <Modal.Header closeButton>
        <Modal.Title>Login</Modal.Title>
      </Modal.Header>
      <Modal.Body>
        <Row className="LoginForm">
          <Col className="p-4 mx-auto" xs={10} sm={10} md={8}>
            <Form onSubmit={onSubmitHandler}>
              <Form.Group className="mb-3">
                <Form.Label>ID</Form.Label>
                <Form.Control
                  type="text"
                  ref={idRef}
                  id="email"
                  name="email"
                  onChange={onChangeInput}
                  value={loginUser.email}
                  placeholder="ID (email)"
                />
              </Form.Group>
              <Form.Group className="mb-3">
                <Form.Label>Password</Form.Label>
                <Form.Control
                  type="password"
                  ref={pwRef}
                  id="passwd"
                  name="passwd"
                  onChange={onChangeInput}
                  value={loginUser.passwd}
                  placeholder="Password"
                />
              </Form.Group>
            </Form>
          </Col>
        </Row>
      </Modal.Body>
    </Modal>
  </>
)
```

react-LoginModal.jsx

```
121 |         placeholder="Password"
122 |     />
123 | </Form.Group>
124 | <div className="d-grid gap-2">
125 |     <Button type="submit" variant="outline-success">
126 |         Login
127 |     </Button>
128 | </div>
129 | </Form>
130 | </Col>
131 | </Row>
132 | </Modal.Body>
133 | </Modal>
134 | </>
135 | );
136 | };
137 | export default LoginModal;
138 |
```