



YouTube

NAVER 카페

구멍가게 코딩단

코드로 배우는 리액트

9. 리액트 소셜 로그인

9장. 리액트 소셜 로그인

- 현재 앱과 웹에서는 전통적인 회원가입 방식 대신 소셜 로그인(구글, 네이버, 애플, 페이스북 등)이 흔히 사용됨
- 리액트와 API 서버를 활용하여 카카오 서비스의 소셜 로그인을 호출
- API 서버에서 사용자의 인증 정보를 기존 회원 데이터와 연동

개발목표

1. 소셜 로그인 방식의 이해
2. 카카오 로그인을 위한 설정
3. 리액트에서 사용자 인증
4. API 서버의 사용자 처리

Index

9.1 소셜 로그인과 OAuth2.0

9.2 카카오 연동 설정

9.3 리액트에서 카카오 로그인

9.4 API 서버에서 Access Token 처리

9.5 자동 회원 추가 및 회원정보의 반환

9.6 회원정보 수정

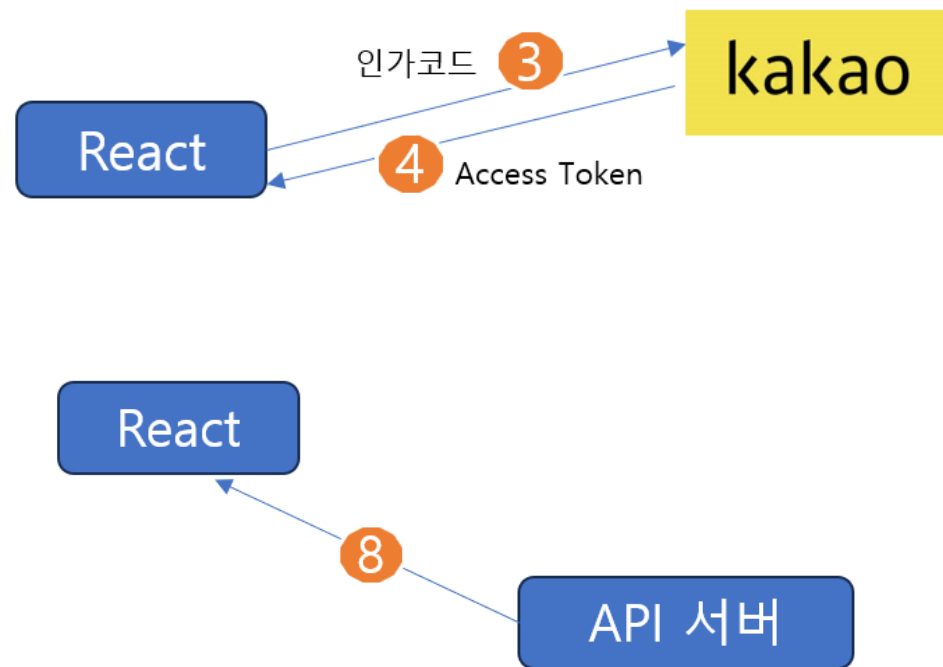
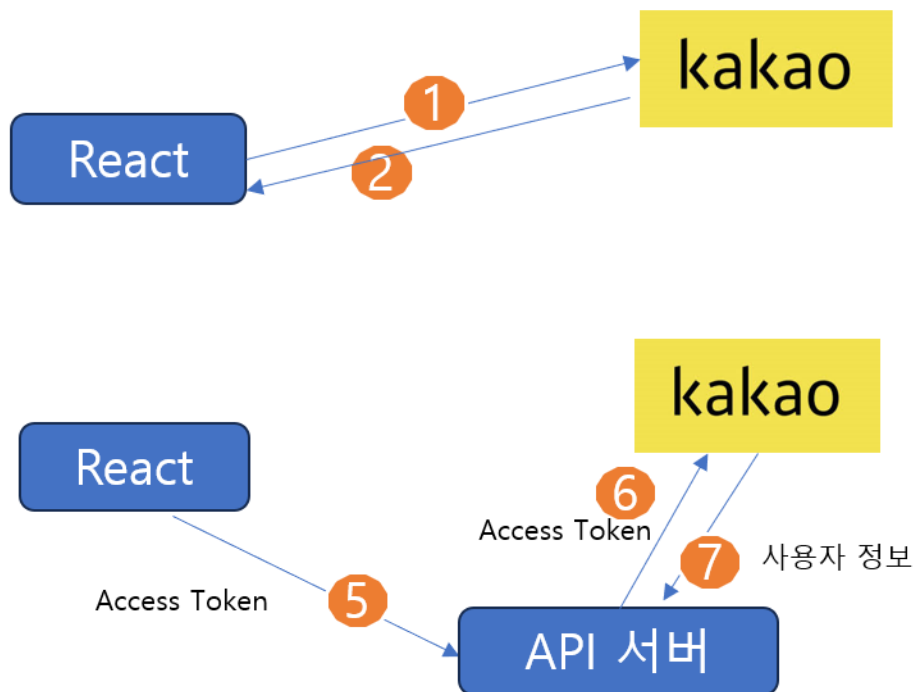
OAuth2.0

→ 다양한 상황에 맞게 사용자의 권한을 부여하는 방식을 제공

- 권한부여 승인 코드 방식(Authorization Code Grant)
- 암묵적 승인 방식(Implicit Grant)
- 자원 소유자 자격증명 승인 방식(Resource Owner Password Credentials Grant)
- 클라이언트 자격증명 승인 방식 (Client Credentials Grant)

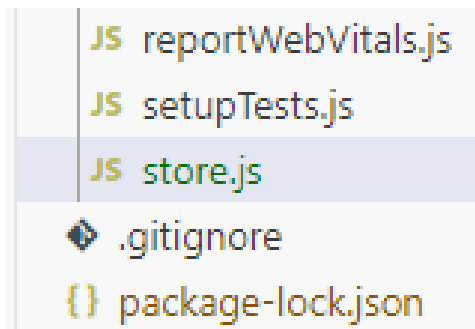
구현 방식

→ 카카오톡이나 네이버 서비스는 '권한부여 승인 코드 방식'



스토어 설정

→ src 폴더내에 store.js 파일을 추가



```
JS reportWebVitals.js
JS setupTests.js
JS store.js
.gitignore
package-lock.json
```

```
import { configureStore } from '@reduxjs/toolkit'

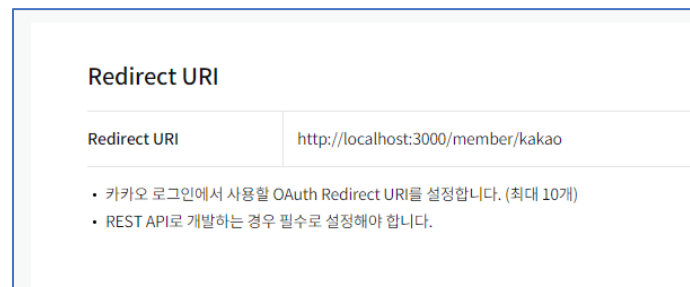
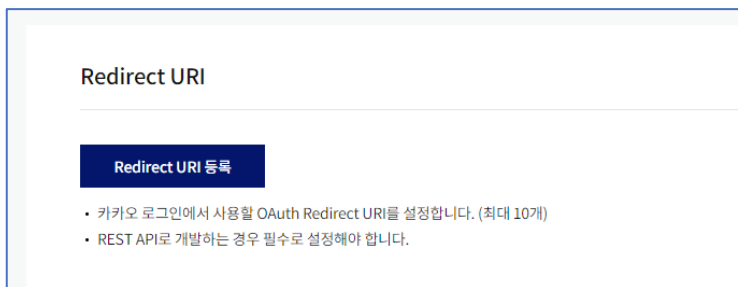
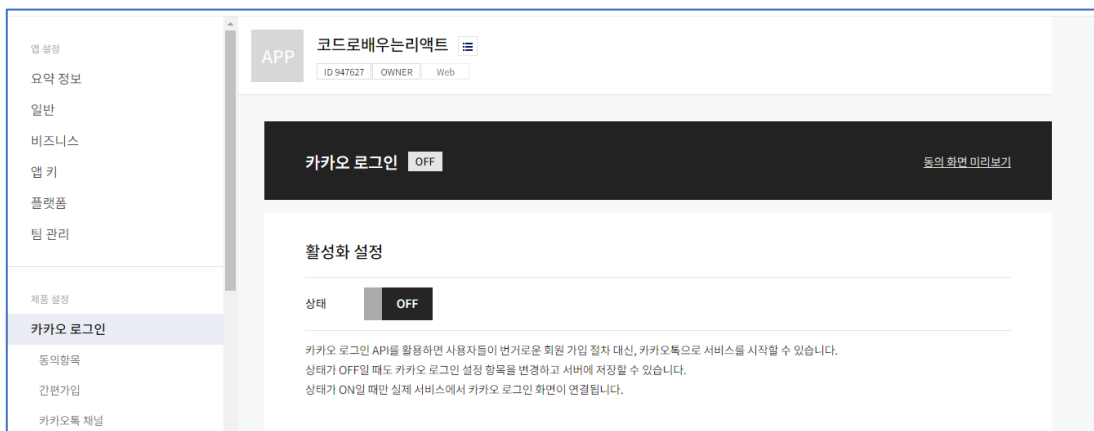
export default configureStore({
  reducer: { }
})
```

카카오 연동 설정

- ① '카카오 개발자' 사이트(<https://developers.kakao.com/>)에서 '내 애플리케이션'을 추가
- ② 로그인 후에 '애플리케이션 추가하기' 메뉴를 선택
- ③ 애플리케이션 추가 시에는 필요한 기본적인 정보를 입력/저장
- ④ '앱 키'들이 생성되면 이 중에서 'REST API키'를 사용
- ⑤ 카카오 로그인을 수행하는 플랫폼을 설정할 때는 Web을 선택
- ⑥ Web 플랫폼 등록에서는 서버 주소와 포트를 지정하는데, 로컬 환경은 'http://localhost:3000'으로 지정

카카오 연동 설정

→ 로그인 동의 설정



카카오 연동 설정

→ 로그인 시 동의 항목 설정

내 애플리케이션 > 제품 설정 > 카카오 로그인 > 동의항목

개인정보			
항목 이름	ID	상태	
닉네임	profile_nickname	● 사용 안함	설정
프로필 사진	profile_image	● 사용 안함	설정
카카오계정(이메일)	account_email	● 사용 안함	설정
이름	name	○ 권한 없음	
성별	gender	● 사용 안함	설정
연령대	age_range	● 사용 안함	설정
생일	birthday	● 사용 안함	설정
출생 연도	birthyear	○ 권한 없음	

동의 항목 설정

항목
카카오계정(이메일) / account_email

동의 단계

- 필수 동의 (필수 필요)
카카오 로그인 시 사용자가 필수로 동의해야 합니다.
- 선택 동의
사용자가 동의하지 않아도 카카오 로그인을 완료할 수 있습니다.
- 이용 중 동의
카카오 로그인 시 동의를 받지 않고, 항목이 필요한 시점에 동의를 받습니다.
- 사용 안함
사용자에게 동의를 요청하지 않습니다.

카카오 계정으로 정보 수집 후 제공

☒ 사용자에게 값이 없는 경우 카카오 계정 정보 입력을 요청하여 수집

동의 목적 [필수]

아이디대신 사용

개발자 앱 동의 항목 관리 화면에 입력하는 사실이 실제 서비스 내용과 다를 경우 API 서비스의 거부 사유가 될 수 있습니다.

취소 저장

동의 항목 설정

항목
닉네임 / profile_nickname

동의 단계

- 필수 동의
카카오 로그인 시 사용자가 필수로 동의해야 합니다.
- 선택 동의
사용자가 동의하지 않아도 카카오 로그인을 완료할 수 있습니다.
- 이용 중 동의
카카오 로그인 시 동의를 받지 않고, 항목이 필요한 시점에 동의를 받습니다.
- 사용 안함
사용자에게 동의를 요청하지 않습니다.

동의 목적 [필수]

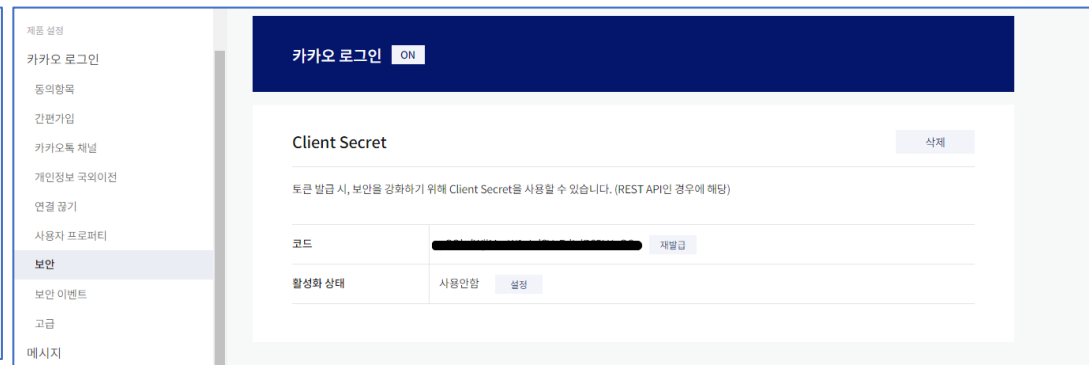
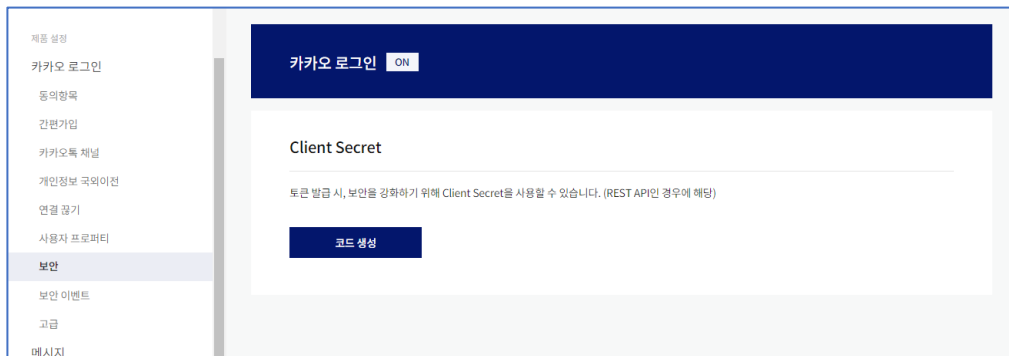
아이디대신 사용

개발자 앱 동의 항목 관리 화면에 입력하는 사실이 실제 서비스 내용과 다를 경우 API 서비스의 거부 사유가 될 수 있습니다.

취소 저장

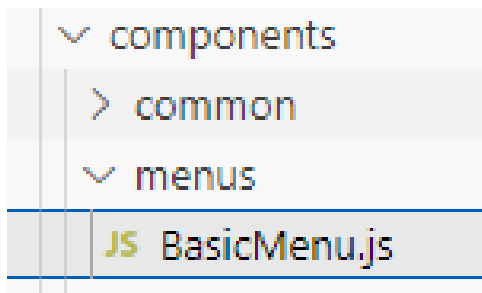
카카오 연동 설정

→ 애플리케이션에서 사용할 비밀 키를 생성



로그인 페이지와 로그인

→ BasicMenu.js 컴포넌트



```
import { useSelector } from "react-redux";
import { Link } from "react-router-dom";

const BasicMenu = () => {
  const loginState = useSelector(state => state.loginSlice)

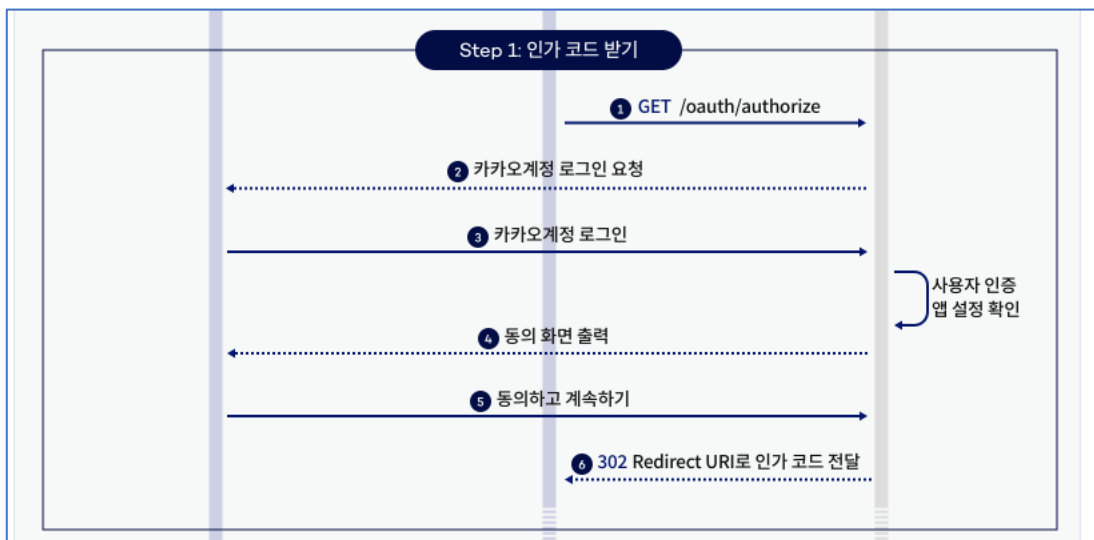
  return (
    <nav id='navbar' className=" flex bg-blue-300">
      ...생략

      <div className="w-1/5 flex justify-end bg-orange-300 p-4 font-medium">
        <div className="text-white text-sm m-1 rounded" >Login</div>
      </div>
    </nav>
  );
}

export default BasicMenu;
```

인가 코드의 처리

→ 동의 화면과 로그인 화면 인가 코드를 처리

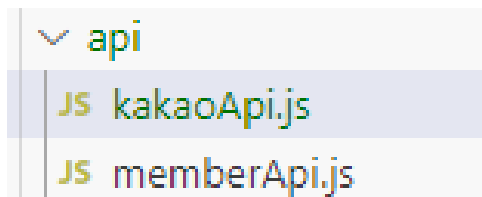


쿼리 파라미터

이름	타입	설명	필수
client_id	String	앱 REST API 키 [내 애플리케이션] > [앱 키]에서 확인 가능	O
redirect_uri	String	인가 코드를 전달받을 서비스 서버의 URI [내 애플리케이션] > [카카오 로그인] > [Redirect URI]에서 등록	O
response_type	String	code로 고정	O

인가 코드의 처리

→ api 폴더에 kakaoApi.js 파일을 추가



```
const rest_api_key = `0a1d9f042...435bd01b571b1e00` //REST키값
const redirect_uri = `http://localhost:3000/member/kakao`

const auth_code_path = `https://kauth.kakao.com/oauth/authorize`

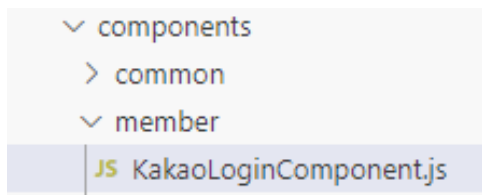
export const getKakaoLoginLink = () => {

  const kakaoURL =
`${auth_code_path}?client_id=${rest_api_key}&redirect_uri=${redirect_uri}&response_type=code`;

  return kakaoURL
}
```

인가 코드의 처리

→ components/member > KakaoLoginComponent.js 추가

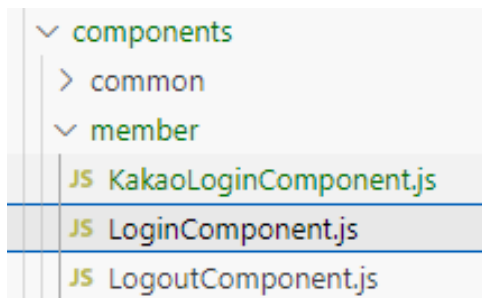


```
import { Link } from "react-router-dom";
import { getKakaoLoginLink } from "../../api/kakaoApi";

const KakaoLoginComponent = () => {
  const link = getKakaoLoginLink()
  return (
    <div className="flex flex-col">
      <div className="text-center text-blue-500">로그인시에 자동 가입처리 됩니다</div>
      <div className="flex justify-center w-full">
        <div
          className="text-3xl text-center m-6 text-white font-extrabold w-3/4 bg-yellow-500
shadow-sm rounded p-2">
          <Link to={link}>KAKAO LOGIN</Link>
        </div>
      </div>
    </div>
  )
}
export default KakaoLoginComponent;
```

인가 코드의 처리

→ 기존의 LoginComponent에 KakaoLoginComponent를 import 해서 추가



```
import { useState } from "react"
import useCustomLogin from "../../hooks/useCustomLogin"
import KakaoLoginComponent from "../KakaoLoginComponent"

...생략

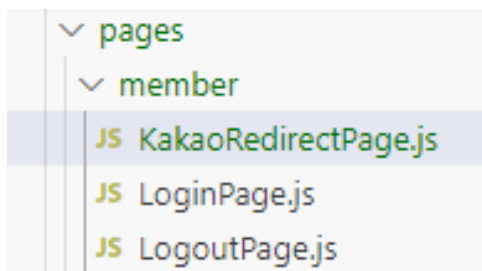
const LoginComponent = () => {
  ...생략

  return (
    <div className = "border-2 border-sky-200 mt-10 m-2 p-4">
      ...생략
      </div>
      </div>
      </div>
      <KakaoLoginComponent/>
    </div>
  );
}

export default LoginComponent;
```

인가 코드의 처리

→ 인가 코드의 페이지 처리



```
import { useSearchParams } from "react-router-dom";

const KakaoRedirectPage = () => {

  const [searchParams] = useSearchParams()

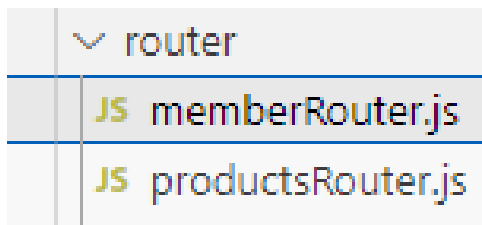
  const authCode = searchParams.get("code")

  return (
    <div>
      <div>Kakao Login Redirect</div>
      <div>{authCode}</div>
    </div>
  )
}

export default KakaoRedirectPage;
```


인가 코드의 처리

→ 인가 코드의 페이지 처리 : KakaoRedirectPage에 대한 라우팅 설정



```
import { Suspense, lazy } from "react";
...생략

const KakaoRedirect = lazy(() => import("../pages/member/KakaoRedirectPage"))

const memberRouter = () => {

  return [
    ...생략
    {
      path: "kakao",
      element: <Suspense fallback={Loading}><KakaoRedirect/></Suspense>,
    }
  ]
}

export default memberRouter
```

Access Token 받기

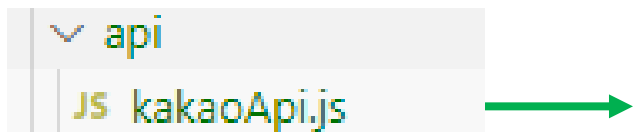
→ <https://kauth.kakao.com/oauth/token>을 호출

본문 

이름	타입	설명	필수
grant_type	String	authorization_code로 고정	O
client_id	String	앱 REST API 키 [내 애플리케이션] > [앱 키]에서 확인 가능	O
redirect_uri	String	인가 코드가 리다이렉트된 URI	O
code	String	인가 코드 받기 요청으로 얻은 인가 코드	O
client_secret	String	토큰 발급 시, 보안을 강화하기 위해 추가 확인하는 코드 [내 애플리케이션] > [보안]에서 설정 가능 ON 상태인 경우 필수 설정해야 함	X

Access Token 받기

→ Access Token 호출



```
import axios from "axios";

const rest_api_key = `a09fff59f94e18.....ba7c87d00` //REST키값
const redirect_uri = `http://localhost:3000/member/kakao`

const auth_code_path = `https://kauth.kakao.com/oauth/authorize`

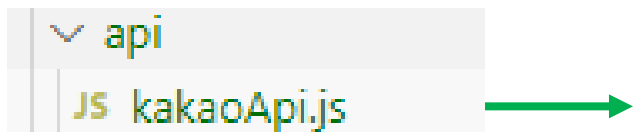
const access_token_url = `https://kauth.kakao.com/oauth/token` //추가

export const getKakaoLoginLink = () => {
  const kakaoURL =
    `${auth_code_path}?client_id=${rest_api_key}&redirect_uri=${redirect_uri}&response_type=code`;

  return kakaoURL
}
```

Access Token 받기

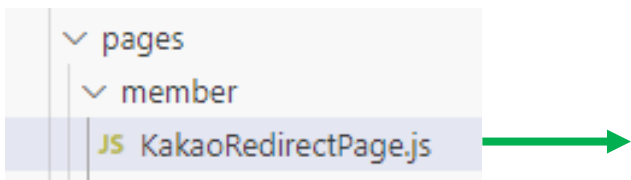
→ Access Token 호출



```
export const getAccessToken = async (authCode) => {  
  const header = {  
    headers: {  
      "Content-Type": "application/x-www-form-urlencoded",  
    },  
  }  
  
  const params = {  
    grant_type: "authorization_code",  
    client_id: rest_api_key,  
    redirect_uri: redirect_uri,  
    code: authCode  
  }  
  
  const res = await axios.post(access_token_url, params, header)  
  const accessToken = res.data.access_token  
  
  return accessToken  
}
```

Access Token 받기

→ KakaoRedirectPage : 인가 코드가 변경되었을 때 `getAccessToken()`을 호출하도록 변경



```
import { useEffect } from "react";
import { useSearchParams } from "react-router-dom";
import { getAccessToken } from "../../api/kakaoApi";


const KakaoRedirectPage = () => {
  const [searchParams] = useSearchParams()
  const authCode = searchParams.get("code")
  useEffect(() => {
    getAccessToken(authCode).then(data => {
      console.log(data)
    })
  }, [authCode])


  return (
    <div>
      <div>Kakao Login Redirect</div>
      <div>{authCode}</div>
    </div>
  )
}

export default KakaoRedirectPage;
```

사용자 정보를 가져오기

→ 'https://kapi.kakao.com/v2/user/me'

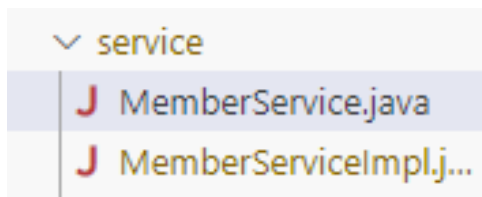
사용자 정보 가져오기 

기본 정보 

메서드	URL	인증 방식
GET/POST	https://kapi.kakao.com/v2/user/me	액세스 토큰 서비스 앱 어드민 키

MemberService의 개발

→ MemberService/MemberServiceImpl 추가



```
package org.zerock.mallapi.service;

import org.springframework.transaction.annotation.Transactional;
import org.zerock.mallapi.dto.MemberDTO;

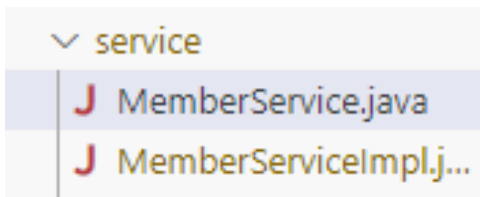
@Transactional
public interface MemberService {

    MemberDTO getKakaoMember(String accessToken);

}
```

MemberService의 개발

→ MemberService/MemberServiceImpl 추가



```
@Service
@RequiredArgsConstructor
@Log4j2
public class MemberServiceImpl implements MemberService {

    private final MemberRepository memberRepository;

    @Override
    public MemberDTO getKakaoMember(String accessToken) {

        String email = getEmailFromKakaoAccessToken(accessToken);

        log.info("email: " + email );

        return null;
    }
}
```


MemberService의 개발

→ MemberService/MemberServiceImpl 추가

```
private String getEmailFromKakaoAccessToken(String accessToken){

    String kakaoGetUserURL = "https://kapi.kakao.com/v2/user/me";

    if(accessToken == null){
        throw new RuntimeException("Access Token is null");
    }
    RestTemplate restTemplate = new RestTemplate();

    HttpHeaders headers = new HttpHeaders();
    headers.add("Authorization", "Bearer " + accessToken);
    headers.add("Content-Type","application/x-www-form-urlencoded");
    HttpEntity<String> entity = new HttpEntity<>(headers);

    UriComponents uriBuilder = UriComponentsBuilder.fromHttpUrl(kakaoGetUserURL).build();
```

MemberService의 개발

→ MemberService/MemberServiceImpl 추가

```
ResponseEntity<LinkedHashMap> response
    = restTemplate.exchange(uriBuilder.toString(), HttpMethod.GET, entity, LinkedHashMap.class);

log.info(response);

LinkedHashMap<String, LinkedHashMap> bodyMap = response.getBody();

log.info("-----");

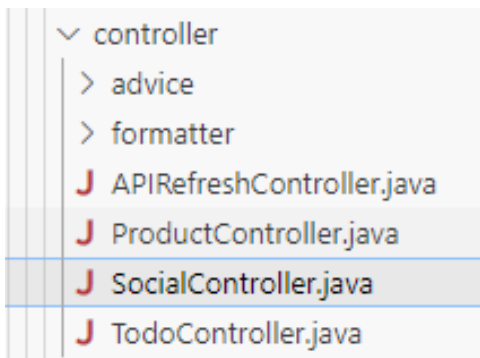
log.info(bodyMap);

LinkedHashMap<String, String> kakaoAccount = bodyMap.get("kakao_account");

log.info("kakaoAccount: " + kakaoAccount);
return kakaoAccount.get("email");
}
```

MemberService의 개발

→ SocialController의 개발



```
@RestController
@Log4j2
@RequiredArgsConstructor
public class SocialController {

    private final MemberService memberService;

    @GetMapping("/api/member/kakao")
    public String[] getMemberFromKakao(String accessToken) {

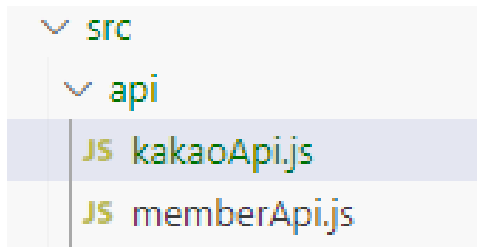
        log.info("access Token ");
        log.info(accessToken);

        memberService.getKakaoMember(accessToken);

        return new String[]{"AAA", "BBB", "CCC"};
    }
}
```

MemberService의 개발

→ 리액트의 호출 테스트 : kakaoApi.js



```
import axios from "axios";
import { API_SERVER_HOST } from "../todoApi";

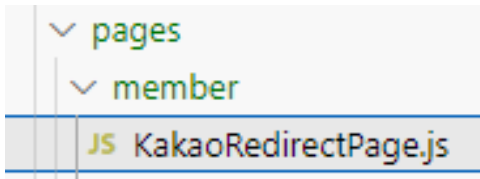
...생략
export const getMemberWithAccessToken = async(accessToken) => {

  const res = await
  axios.get(`${API_SERVER_HOST}/api/member/kakao?accessToken=${accessToken}`)

  return res.data
}
```

MemberService의 개발

→ 리액트의 호출 테스트
: KakaoRedirectPage.js



```
import { useEffect } from "react";
import { useSearchParams } from "react-router-dom";
import { getAccessToken, getMemberWithAccessToken } from "../../api/kakaoApi";

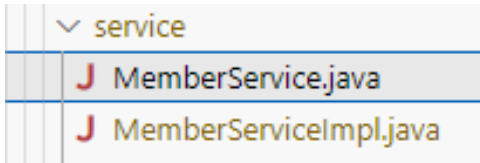
const KakaoRedirectPage = () => {
  const [searchParams] = useSearchParams()
  const authCode = searchParams.get("code")
  useEffect(() => {
    getAccessToken(authCode).then(accessToken => {
      console.log(accessToken)
      getMemberWithAccessToken(accessToken).then(memberInfo => {
        console.log("-----")
        console.log(memberInfo)
      })
    })
  }, [authCode])

  return (
    <div>
      <div>Kakao Login Redirect</div>
      <div>{authCode}</div>
    </div>
  )
}

export default KakaoRedirectPage;
```

MemberService 회원 처리

→ Member 엔티티 객체를 MemberDTO 객체로 변환하는 entityToDTO()를 추가



```
@Transactional
public interface MemberService {

    MemberDTO getKakaoMember(String accessToken);

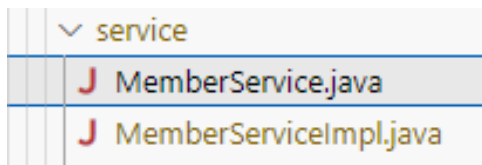
    default MemberDTO entityToDTO(Member member) {

        MemberDTO dto = new MemberDTO(
            member.getEmail(),
            member.getPw(),
            member.getNickname(),
            member.isSocial(),
            member.getMemberRoleList().stream()
                .map(memberRole -> memberRole.name()).collect(Collectors.toList()));

        return dto;
    }
}
```

MemberService 회원 처리

→ Member 엔티티 객체를 MemberDTO 객체로 변환하는 entityToDTO()를 추가



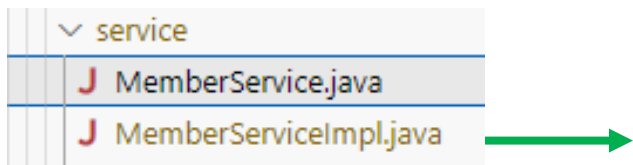
```
@Service
@RequiredArgsConstructor
@Log4j2
public class MemberServiceImpl implements MemberService {

    private final MemberRepository memberRepository;

    private final PasswordEncoder passwordEncoder;
    ...생략
```

MemberService 회원 처리

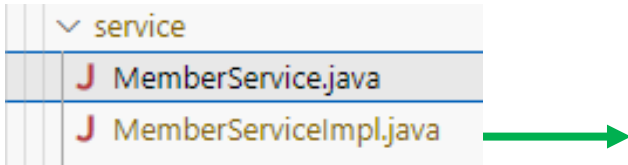
→ Member 엔티티 객체를 MemberDTO 객체로 변환하는 entityToDTO()를 추가



```
private String makeTempPassword() {  
    StringBuffer buffer = new StringBuffer();  
  
    for(int i = 0; i < 10; i++){  
        buffer.append( (char) ( (int)(Math.random()*55) + 65 ));  
    }  
    return buffer.toString();  
}
```


MemberService 회원 처리

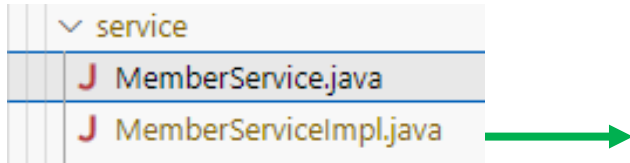
→ Member 엔티티 객체를 MemberDTO 객체로 변환하는 entityToDTO()를 추가



```
private Member makeSocialMember(String email) {  
  
    String tempPassword = makeTempPassword();  
  
    log.info("tempPassword: " + tempPassword);  
  
    String nickname = "소셜회원";  
  
    Member member = Member.builder()  
        .email(email)  
        .pw(passwordEncoder.encode(tempPassword))  
        .nickname(nickname)  
        .social(true)  
        .build();  
  
    member.addRole(MemberRole.USER);  
  
    return member;  
}
```

MemberService 회원 처리

→ Member 엔티티 객체를 MemberDTO 객체로 변환하는 entityToDTO()를 추가



```
@Override
public MemberDTO getKakaoMember(String accessToken) {

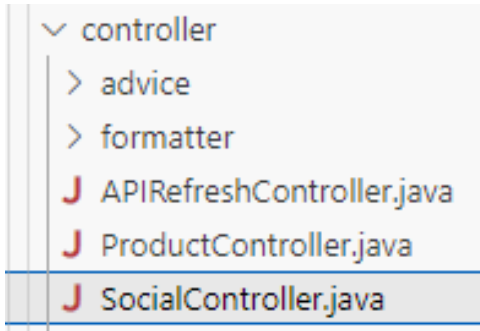
    String email = getEmailFromKakaoAccessToken(accessToken);
    log.info("email: " + email );
    Optional<Member> result = memberRepository.findById(email);

    //기존의 회원
    if(result.isPresent()){
        MemberDTO memberDTO = entityToDTO(result.get());
        return memberDTO;
    }

    //회원이 아니었다면 닉네임은 '소셜회원' 으로 패스워드는 임의로 생성
    Member socialMember = makeSocialMember(email);
    memberRepository.save(socialMember);
    MemberDTO memberDTO = entityToDTO(socialMember);
    return memberDTO;
}
```

MemberService 회원 처리

→ 컨트롤러의 결과 처리



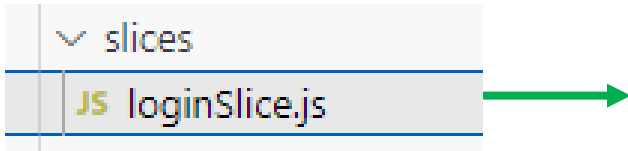
```
@RestController
@Log4j2
@RequiredArgsConstructor
public class SocialController {

    private final MemberService memberService;

    @GetMapping("/api/member/kakao")
    public Map<String, Object> getMemberFromKakao(String accessToken) {
        log.info("access Token : " + accessToken);
        MemberDTO memberDTO = memberService.getKakaoMember(accessToken);
        Map<String, Object> claims = memberDTO.getClaims();
        String jwtAccessToken = JWTUtil.generateToken(claims, 10);
        String jwtRefreshToken = JWTUtil.generateToken(claims, 60*24);
        claims.put("accessToken", jwtAccessToken);
        claims.put("refreshToken", jwtRefreshToken);
        return claims;
    }
}
```

MemberService 회원 처리

→ 리액트의 로그인 처리



```
import { createAsyncThunk, createSlice } from "@reduxjs/toolkit";
import { loginPost } from "../api/memberApi";

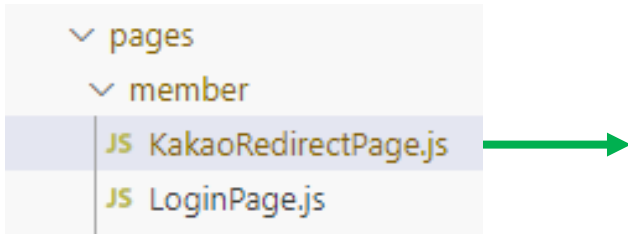
import { getCookie, removeCookie, setCookie } from "../util/cookieUtil";
...생략

const loginSlice = createSlice({
  name: 'LoginSlice',
  initialState: loadMemberCookie() || initState, //쿠키가 없다면 초깃값사용
  reducers: {
    login: (state, action) => {
      console.log("login.....")
      const payload = action.payload // {소셜로그인 회원이 사용}
      setCookie("member", JSON.stringify(payload), 1) //1일
      return payload
    },
    logout: (state, action) => {
      console.log("logout....")
      removeCookie('member')
      return {...initState}
    }
  },
  extraReducers: (builder) => {
    ...생략
  }
})

export const {login,logout} = loginSlice.actions
export default loginSlice.reducer
```

MemberService 회원 처리

→ KakaoRedirectPage.js



```
import { useEffect } from "react";
import { useSearchParams } from "react-router-dom";
import { getAccessToken, getMemberWithAccessToken } from "../../api/kakaoApi";
import { useDispatch } from "react-redux";
import { login } from "../../slices/loginSlice";

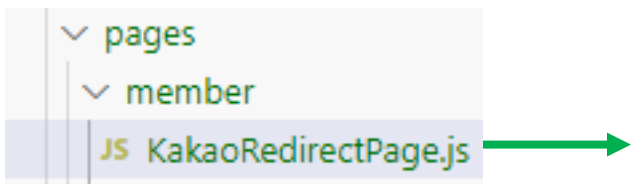
const KakaoRedirectPage = () => {
  const [searchParams] = useSearchParams()
  const dispatch = useDispatch()
  const authCode = searchParams.get("code")
  useEffect(() => {
    getAccessToken(authCode).then(accessToken => {
      console.log(accessToken)
      getMemberWithAccessToken(accessToken).then(memberInfo => {
        console.log("-----")
        console.log(memberInfo)
        dispatch(login(memberInfo))
      })
    })
  }, [authCode])

  return (
    <div>
      <div>Kakao Login Redirect</div>
      <div>{authCode}</div>
    </div>
  )
}

export default KakaoRedirectPage;
```

MemberService 회원 처리

→ 화면 이동 처리



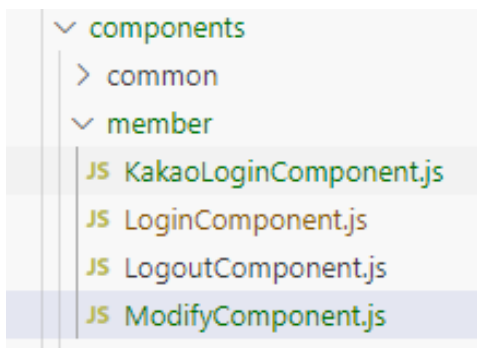
```
import { useEffect } from "react";
import { useSearchParams } from "react-router-dom";
import { getAccessToken, getMemberWithAccessToken } from "../../api/kakaoApi";
import { useDispatch } from "react-redux";
import { login } from "../../slices/loginSlice";
import useCustomLogin from "../../hooks/useCustomLogin";

const KakaoRedirectPage = () => {
  const [searchParams] = useSearchParams()
  const {moveToPath} = useCustomLogin()
  const dispatch = useDispatch()
  const authCode = searchParams.get("code")
  useEffect(() => {
    getAccessToken(authCode).then(accessToken => {
      getMemberWithAccessToken(accessToken).then(memberInfo => {
        dispatch(login(memberInfo))
        if(memberInfo && !memberInfo.social){//소셜 회원이 아니라면
          moveToPath("/")
        }else {
          moveToPath("/member/modify")
        }
      })
    })
  }, [authCode])
  return (
    <div>
      <div>Kakao Login Redirect</div>
      <div>{authCode}</div>
    </div>
  )
}

export default KakaoRedirectPage;
```

회원정보 수정 화면 처리

→ 화면 이동 처리



```
import { useEffect } from "react";
import { useState } from "react";
import { useSelector } from "react-redux";

const initState = { email: '', pw: '', nickname: '' }

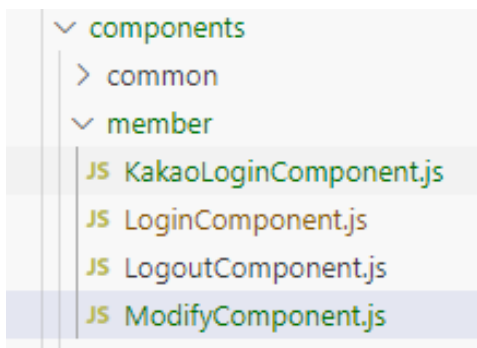
const ModifyComponent = () => {
  const [member, setMember] = useState(initState)
  const loginInfo = useSelector(state => state.loginSlice)

  useEffect(() => {
    setMember({...loginInfo, pw: 'ABCD'})
  }, [loginInfo])

  const handleChange = (e) => {
    member[e.target.name] = e.target.value
    setMember({...member})
  }
}
```

회원정보 수정 화면 처리

→ 화면 이동 처리



```
import { useEffect } from "react";
import { useState } from "react";
import { useSelector } from "react-redux";

const initState = { email: '', pw: '', nickname: '' }

const ModifyComponent = () => {
  const [member, setMember] = useState(initState)
  const loginInfo = useSelector(state => state.loginSlice)

  useEffect(() => {
    setMember({...loginInfo, pw: 'ABCD'})
  }, [loginInfo])

  const handleChange = (e) => {
    member[e.target.name] = e.target.value
    setMember({...member})
  }
}
```


회원정보 수정 화면 처리

→ 화면 이동 처리

```
return (  
  <div className="mt-6">  
    <div className="flex justify-center">  
      <div className="relative mb-4 flex w-full flex-wrap items-stretch">  
        <div className="w-1/5 p-6 text-right font-bold">Email</div>  
        <input className="w-4/5 p-6 rounded-r border border-solid border-neutral-300 shadow-md"  
          name="email" type={'text'} value={member.email} readOnly />  
      </div>  
    </div>  
    <div className="flex justify-center">  
      <div className="relative mb-4 flex w-full flex-wrap items-stretch">  
        <div className="w-1/5 p-6 text-right font-bold">Password</div>  
        <input className="w-4/5 p-6 rounded-r border border-solid border-neutral-300 shadow-md"  
          name="pw" type={'password'} value={member.pw} onChange={handleChange} />  
      </div>  
    </div>  
  </div>  
)
```

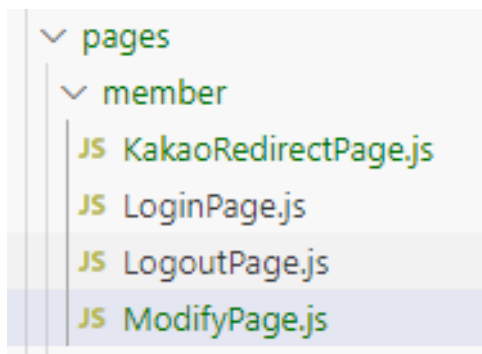
회원정보 수정 화면 처리

→ 화면 이동 처리

```
<div className="flex justify-center">
  <div className="relative mb-4 flex w-full flex-wrap items-stretch">
    <div className="w-1/5 p-6 text-right font-bold">Nickname</div>
    <input className="w-4/5 p-6 rounded-r border border-solid border-neutral-300 shadow-md"
      name="nickname" type={'text'} value={member.nickname} onChange={handleChange} >
    </input>
  </div>
</div>
<div className="flex justify-center">
  <div className="relative mb-4 flex w-full flex-wrap justify-end">
    <button type="button" className="rounded p-4 m-2 text-xl w-32 text-white bg-blue-500" > Modify </button>
  </div>
</div>
</div>
);
}
export default ModifyComponent;
```

회원정보 수정 화면 처리

→ pages/member/ModifyPage 추가



```
import ModifyComponent from "../../components/member/ModifyComponent";
import BasicLayout from "../../layouts/BasicLayout";

const ModifyPage = () => {
  return (
    <BasicLayout>
      <div className="text-3xl">Member Modify Page</div>

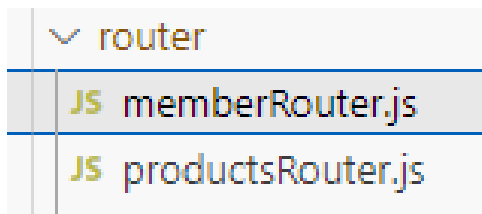
      <div className="bg-white w-full mt-4 p-2">
        <ModifyComponent></ModifyComponent>
      </div>

    </BasicLayout>
  );
}

export default ModifyPage;
```

회원정보 수정 화면 처리

→ pages/member/ModifyPage 추가



```
import { Suspense, lazy } from "react";

...생략

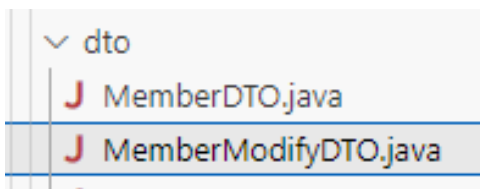
const MemberModify = lazy(() => import("../pages/member/ModifyPage"))

const memberRouter = () => {
  return [
    ...생략
    {
      path: "modify",
      element: <Suspense fallback={Loading}><MemberModify/></Suspense>,
    },
  ]
}

export default memberRouter
```

API 서버의 회원정보 수정

→ pages/member/ModifyPage 추가



```
package org.zerock.mallapi.dto;

import lombok.Data;

@Data
public class MemberModifyDTO {

    private String email;

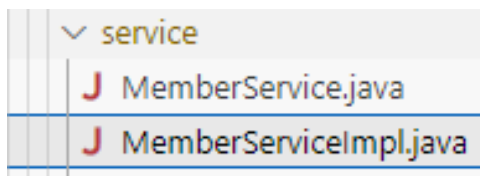
    private String pw;

    private String nickname;

}
```

API 서버의 회원정보 수정

→ pages/member/ModifyPage 추가



```
package org.zerock.mallapi.service;

import java.util.stream.Collectors;

import org.springframework.transaction.annotation.Transactional;
import org.zerock.mallapi.domain.Member;
import org.zerock.mallapi.dto.MemberDTO;
import org.zerock.mallapi.dto.MemberModifyDTO;

@Transactional
public interface MemberService {

    MemberDTO getKakaoMember(String accessToken);

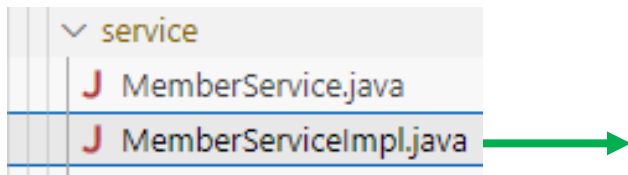
    void modifyMember(MemberModifyDTO memberModifyDTO);

    ...생략

}
```

API 서버의 회원정보 수정

→ pages/member/ModifyPage 추가



```
@Override
public void modifyMember(MemberModifyDTO memberModifyDTO) {

    Optional<Member> result =
memberRepository.findById(memberModifyDTO.getEmail());

    Member member = result.orElseThrow();

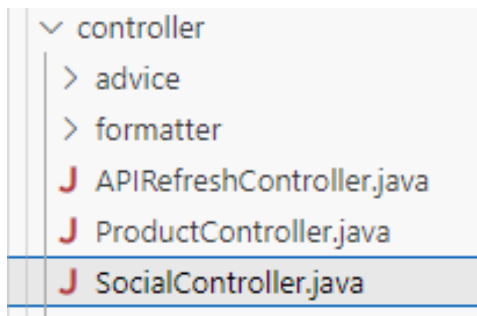
    member.changePw(passwordEncoder.encode(memberModifyDTO.getPw()));
    member.changeSocial(false);
    member.changeNickname(memberModifyDTO.getNickname());

    memberRepository.save(member);

}
```

API 서버의 회원정보 수정

→ pages/member/ModifyPage 추가



```
@PostMapping("/api/member/modify")
public Map<String,String> modify(@RequestBody MemberModifyDTO
memberModifyDTO) {

    log.info("member modify: " + memberModifyDTO);

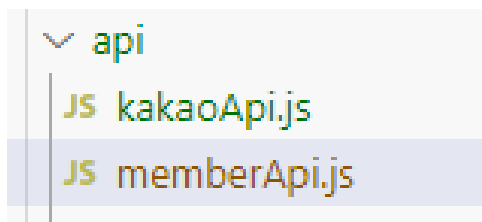
    memberService.modifyMember(memberModifyDTO);

    return Map.of("result","modified");

}
```


리액트와 API 연동

→ '/api/member/modify' 경로를 호출하는 코드를 memberApi.js에 추가



```
import axios from "axios"
import { API_SERVER_HOST } from "../todoApi"
import jwtAxios from "../util/jwtUtil"

const host = `${API_SERVER_HOST}/api/member`

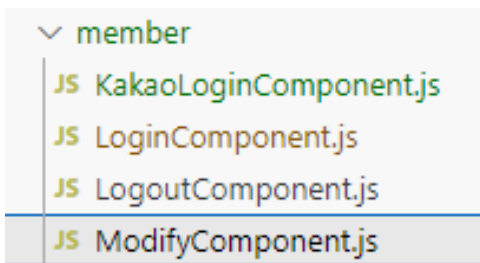
export const loginPost = async (loginParam) => {
  ...생략
}

export const modifyMember = async (member) => {
  const res = await jwtAxios.put(`${host}/modify`, member)

  return res.data
}
```

리액트와 API 연동

→ memberApi.js 수정



```
import { useEffect } from "react";
import { useState } from "react";
import { useSelector } from "react-redux";
import { modifyMember } from "../../api/memberApi";

...생략
const ModifyComponent = () => {

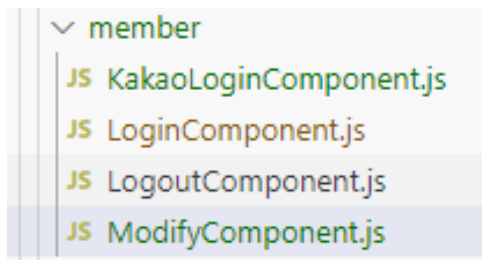
  ...생략
  const handleClickModify = () => {
    modifyMember(member)
  }
  return (
    <>
    <div className="text-xl">Member Modify</div>

    ...생략
    <div className="flex justify-center">
      <div className="relative mb-4 flex w-full flex-wrap justify-end">
        <button type="button" onClick={handleClickModify}
          className="rounded p-4 m-2 text-xl w-32 text-white bg-blue-500">
          Modify
        </button>
      </div>
    </div>
    </>
  );
}

export default ModifyComponent;
```

리액트와 API 연동

→ 수정 후 다시 로그인 하기



```
import { useEffect } from "react";
import { useState } from "react";
import { useSelector } from "react-redux";
import { modifyMember } from "../../api/memberApi";
import useCustomLogin from "../../hooks/useCustomLogin";
import ResultModal from "../../common/ResultModal";

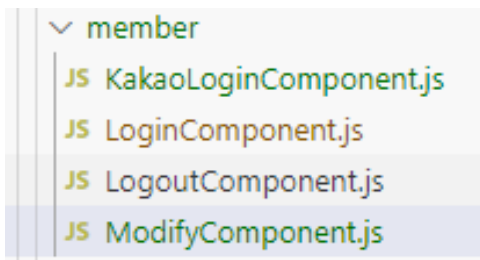
const initState = { email: '', pw: '', nickname: '' }

const ModifyComponent = () => {
  const [member, setMember] = useState(initState)
  const loginInfo = useSelector(state => state.loginSlice)
  const {moveToLogin} = useCustomLogin()
  const [result, setResult] = useState()

  useEffect(() => {
    setMember({...loginInfo, pw: 'ABCD'})
  }, [loginInfo])
```

리액트와 API 연동

→ 수정 후 다시 로그인 하기



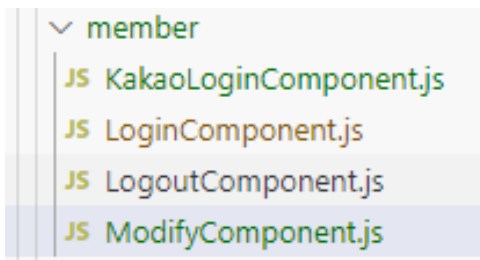
```
const handleChange = (e) => {
  member[e.target.name] = e.target.value
  setMember({...member})
}

const handleClickModify = () => {
  modifyMember(member).then(result => {
    setResult('Moodified')
  })
}

const colseModal = () => {
  setResult(null)
  moveToLogin()
}
```

리액트와 API 연동

→ 수정 후 다시 로그인 하기



```
return (  
  <div className="mt-6">  
    {result? <ResultModal title={'회원정보'} content={'정보수정완료'}  
      callbackFn={colseModal}></ResultModal>:<></>}  
    <div className="flex justify-center">  
      ...생략  
    </div>  
  </div>  
}  
  
export default ModifyComponent;
```

감사합니다.