

Spring Boot - Oauth2 로그인 붙여보기 (구글, 네이버, 카카오)

Spring Boot의 Spring Security 기능과 yml을 활용하여 Google, Naver, 카카오 연동 Oauth2 로그인을 붙여보는 내용을 진행해 보도록 한다.

1. 기본 세팅 : 구글 로그인

먼저 구글 로그인을 설정해보도록 한다.

start.spring.io에서

devtool, lombok, MySQL, Spring Data JPA, thymeleaf 디펜던시를 추가한 후 프로젝트를 생성하였다.(기존 Spring Security Test Project 사용)

프로젝트를 연 뒤 기본 properties파일을 yml로 변경하고 아래와 같이 설정을 변경한다.

```
server:
  port: 80

spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/article?serverTimezone=Asia/Seoul
    username: root
    password: 1111

jpa:
  hibernate:
    ddl-auto: validate #create update none
  show-sql: true
```

구글 클라우드 콘솔(<https://console.cloud.google.com/>)로 이동하여 테스트용 프로젝트를 설정해보자.



Google Cloud 리소스, 문서, 제품

새 프로젝트

projects 할당량이 8개 남았습니다. 할당량 증가를 요청하거나 프로젝트를 삭제하세요. [자세히 알아보기](#)

[MANAGE QUOTAS](#)

프로젝트 이름 *
My Project 33796

프로젝트 ID: modern-photon-396107입니다. 나중에 변경할 수 없습니다. [수정](#)

위치 *
조직 없음 [찾아보기](#)

상위 조직 또는 폴더

[만들기](#) [취소](#)

최초 접속 시 약관 동의가 나오면 약관 동의 후, 화면 좌상단의

프로젝트 선택 > 새 프로젝트

를 선택하여 적당한 이름(articleBoard)을 입력하고 프로젝트를 생성한다. 시간이 조금 소요되고 프로젝트가 생성되면

탐색 메뉴(좌상단 햄버거 메뉴) > API 및 서비스 > 사용자 인증 정보

로 이동하여 상단의

사용자 인증 정보 만들기 > OAuth 클라이언트 ID를 실행

한다. 자동으로 OAuth 동의 화면 생성이 진행된다면, 필수 정보들을 입력 한 뒤 넘어가도록 하자.

동의 화면 생성이 끝났으면 다시 사용자 인증 정보로 이동하여

사용자 인증 정보 만들기 > OAuth 클라이언트 ID

를 실행한 뒤, 본인이 작성할 성격에 맞게 설정한 뒤 승인된 리디렉션 URI에는

<http://localhost:8088/login/oauth2/code/google>
(혹은 본인이 사용할 다른 포트번호)

을 입력하여 설정을 마무리한다.

스프링 부트 공식문서에 나와있는 내용이다.

<https://docs.spring.io/spring-security/reference/servlet/oauth2/login/core.html>

💡 Tip

The default redirect URI template is `{baseUrl}/login/oauth2/code/{registrationId}`. The `registrationId` is a unique identifier for the `ClientRegistration`.

프로젝트 페이지의 Additional information 항목을 보면 클라이언트 ID와 클라이언트 보안 비밀번호를 확인할 수 있는데, 이 정보를 아래와 같이 application.yml 파일에 설정해주자.

Downloaded (1 of 1 enabled)



Spring Boot Assistant

222.17.2 PENG FEI



This plugin adds auto-completion support for Spring Boot configuration files (application.yml, etc.).

Features:

plugins 에 Spring Boot Assistant를 추가해서 yml 등 Configuration 파일 편집이 용이하도록 한다.

```
spring:
  security:
    oauth2:
      client:
        registration:
          google:
            client-id: {클라이언트 ID}
            client-secret: {보안 비밀번호}
            scope: profile,email
```

구글의 경우 Spring Security에서 기본적으로 프로바이더를 제공해주기 때문에 카카오,

네이버와 다르게 provider 설정을 할 필요는 없다.

```
spring:
  security:
    oauth2:
      client:
        registration:
          google: # /oauth2/authorization/google 이 주소를 동작하게 한다.
            clientId: 1073085804262-5ik2cjma4soc692vkkct5v5qanqtse4r.apps.googleusercontent.com
            clientSecret: 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
            scope:
              - email
              - profile
```

yaml 설정이 완료되었으면 컨트롤러와 도메인, 레포지토리, 서비스, dto, Security Config를 작성한다.

● 로그인 폼에 구글 로그인 링크 붙이기

```
<a href="/oauth2/authorization/google">구글로그인</a>
<a href="/joinForm">회원가입하기</a>
```

● 처리 절차

- ① 코드받기(인증 완료)
 - ② Access Token 발급 - 권한설정
 - ③ 사용자 프로필 정보 가져오기
 - ④ 회원가입(전체 권한 위임 or Database 저장)
- ▶ 구글 로그인이 완료 되면 액세스토큰 + 사용자프로필 정보를 함께 받는다.

```
.oauth2Login(oAuth -> oAuth
    .loginPage("/user/login"))
```

● OAuth2에게 로그인 정보 가로 챌 위치 알려주기

위 내용을 테스트 해 보면 구글 로그인 페이지가 정상적으로 보이고 로그인까지 진행

된다.

● entity - User.java 수정

```
@Data
@Entity
@Builder
@RequiredArgsConstructor
@AllArgsConstructor
public class User {
    @Id // primary key
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String username;
    private String password;
    private String email;
    private UserRole userRole;
    @CreationTimestamp
    private LocalDateTime createDate;
    private String provider;
    private String providerId;
}
```

● UserDto.java 수정

```
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class UserDto{
    private String username;
    private String password;
    private String email;
    private String provider;
```

```
private String providerId; }
```

● PrincipalOauth2UserService 만들기

- 기존에 구글 로그인 정보가 있으면 바로 로그인 처리
- 처음 로그인하는 사용자라면 회원가입(DB테이블에 신규 저장) 후 자동 로그인 처리
- 유저 정보를 생성 한 후 PrincipalDetails 신규 정보를 생성해 줌.

```
<google>
attributes={
    sub=107869079993331253150,
    name=H M K,
    given_name=H M,
    family_name=K,
    picture=https://lh3.googleusercontent.com/a/AACHTtdG,
    email=sm01079200887@gmail.com,
    email_verified=true,
    locale=ko
}

<naver>
{
    resultcode=00,
    message=success,
    response = {
        id=pvdq1FSG3VZID7Cp3JuWfAFi-3xir6A-WPIP5f8kXlo,
        email=vizolmin@naver.com,
        name=김형민
    }
}
```

```
<kakao>
{
  id=2632890179,
  connected_at=2023-01-22T08:17:54Z,
  properties = {nickname=김형민},
  kakao_account = {
    profile_nickname_needs_agreement=false,
    profile={nickname=김형민},
    has_email=true,
    email_needs_agreement=false,
    is_email_valid=true,
    is_email_verified=true,
    email=kiti6000@daum.net
  }
}
```

● interface OAuth2UserInfo

```
public interface OAuth2UserInfo {
    String getProviderId();
    String getProvider();
    String getEmail();
    String getName();
}
```

● GoogleUserInfo

```
public class GoogleUserInfo implements OAuth2UserInfo{
    private Map<String, Object> attributes;

    public GoogleUserInfo(Map<String, Object> attributes) {
        this.attributes = attributes;
    }
}
```

```

@Override
public String getProviderId() {
    return (String)attributes.get("sub") ;
}

@Override
public String getProvider() {
    return "google";
}

@Override
public String getEmail() {
    return (String) attributes.get("email");
}

@Override
public String getName() {
    return (String) attributes.get("name");
}
}

```

● NaverUserInfo

```

public class NaverUserInfo implements OAuth2UserInfo{
    private Map<String, Object> attributes;

    public NaverUserInfo(Map<String, Object> attributes) {
        this.attributes = attributes;
    }

    @Override
    public String getProviderId() {
        return (String) attributes.get("id");
    }
}

```



```
@Override
public String getProvider() {
    return "naver";
}

@Override
public String getEmail() {
    return (String) attributes.get("email");
}

@Override
public String getName() {
    return (String) attributes.get("name");
}
}
```

● KakaoUserInfo

```
public class KakaoUserInfo implements OAuth2UserInfo{
    private Map<String, Object> attributes;

    public KakaoUserInfo(Map<String, Object> attributes) {
        this.attributes = attributes;
    }

    @Override
    public String getProviderId() {
        return attributes.get("id").toString();
    }

    @Override
    public String getProvider() {
        return "kakao";
    }
}
```

```

@Override
public String getEmail() {
    return (String) ((Map)attributes.get("kakao_account")).get("email");
}

@Override
public String getName() {
    return (String) ((Map)attributes.get("properties")).get("nickname");
}
}

```

● PrincipalOAuth2UserService

```

@Service
public class PrincipalOAuth2UserService extends DefaultOAuth2UserService {

    @Autowired
    private UsersRepository userRepository;

    // userRequest 는 code를 받아서 accessToken을 응답 받은 객체
    // 함수 종료 시 @AuthenticationPrincipal Annotation 생성
    @Override
    public OAuth2User loadUser(OAuth2UserRequest userRequest) throws
        OAuth2AuthenticationException {
        System.out.println("getClientRegistration : "
            + userRequest.getClientRegistration());
        System.out.println("getAccessToken : " +
            userRequest.getAccessToken().getTokenValue());

        OAuth2User oAuth2User = super.loadUser(userRequest);
        // google의 회원 프로필 조회
        return processOAuth2User(userRequest, oAuth2User);
    }
}

```

● processOAuth2User

```
private OAuth2User processOAuth2User(OAuth2UserRequest userRequest,
                                      OAuth2User oAuth2User) {

    // Attribute를 파싱해서 공통 객체로 묶는다. 관리가 편함.
    OAuth2UserInfo oAuth2UserInfo = null;

    if (userRequest.getClientRegistration().getRegistrationId().equals("google")) {
        System.out.println("구글 로그인 요청");
        oAuth2UserInfo = new GoogleUserInfo(oAuth2User.getAttributes());
    } else if
        (userRequest.getClientRegistration().getRegistrationId().equals("naver")){
        System.out.println("네이버 로그인 요청");
        Map<String, Object> response = (Map<String, Object>)
            oAuth2User.getAttributes().get("response");
        oAuth2UserInfo = new NaverUserInfo(response);
    } else if
        (userRequest.getClientRegistration().getRegistrationId().equals("kakao")){
        System.out.println("카카오 로그인 요청");
        oAuth2UserInfo = new
            KakaoUserInfo((Map)oAuth2User.getAttributes());
    }
    else {
        System.out.println("우리는 구글과 페이스북, 카카오만 지원해요");
    }

    Optional<User> userOptional =

        userRepository.findByProviderAndProviderId(oAuth2UserInfo.getProvider(),
            oAuth2UserInfo.getProviderId());

    User user;
    if (userOptional.isPresent()) {
        user = userOptional.get();
    }
}
```

```

        // user가 존재하면 update 해주기
        user.setEmail(oAuth2UserInfo.getEmail());
        userRepository.save(user);
    } else {
        // user의 패스워드 해석불가하기 때문에 OAuth 유저는 일반적인
        // 로그인할 수 없음.
        user = User.builder()
            .username(oAuth2UserInfo.getProvider() + "_" +
                oAuth2UserInfo.getProviderId())
            .email(oAuth2UserInfo.getEmail())
            .role("ROLE_USER")
            .password(String.valueOf(UUID.randomUUID()))
            .provider(oAuth2UserInfo.getProvider())
            .providerId(oAuth2UserInfo.getProviderId())
            .build();
        userRepository.save(user);
    }

    return new PrincipalDetails(user, oAuth2User.getAttributes());
}
}

```

● PrincipalDetails 처리(일반 로그인 처리와 OAuth2 처리를 함께 함.)

```

@Getter
@ToString
public class PrincipalDetails implements UserDetails, OAuth2User{

    private User user;
    private Map<String, Object> attributes;

    // 일반시큐리티로그인시사용
    public PrincipalDetails(User user) {
        this.user = user;
    }
}

```

```
}

// OAuth2.0 로그인시사용
public PrincipalDetails(User user, Map<String, Object> attributes) {
    this.user = user;
    this.attributes = attributes;
}

public User getUser() {
    return user;
}

@Override
public String getPassword() {
    return user.getPassword();
}

@Override
public String getUsername() {
    return user.getUsername();
}

@Override
public boolean isAccountNonExpired() {
    return true;
}

@Override
public boolean isAccountNonLocked() {
    return true;
}

@Override
public boolean isCredentialsNonExpired() {
```

```

return true;
}

@Override
public boolean isEnabled() {
return true;
}

/**
 * UserDetails 구현
 * 해당유저의 권한 목록리턴
 */

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    Collection<GrantedAuthority> collet = new ArrayList<>();
    collet.add()->{ return user.getRole();});
    return collet;
}

// 리소스 서버로부터 받는 회원정보
@Override
public Map<String, Object> getAttributes() {
return attributes;
}

// User의PrimaryKey
@Override
public String getName() {
String sub = attributes.get("sub").toString();
return sub;
}
}

```

● Config - SecurityConfig 설정 추가

```
public class SecurityConfig{

@Autowired private PrincipalOAuth2UserService principalOAuth2UserService;

---- 중간 생략

.oauth2Login(oAuth -> oAuth
.loginPage("/loginForm")
.defaultSuccessUrl("/")
.userInfoEndpoint(userInfo-> userInfo
.userService(principalOAuth2UserService)))
```

2. 네이버 로그인

구글 클라우드 센터와 비슷하게 진행된다. 네이버 개발자 센터로 이동하여 개발자 등록이 필요하면 등록 한 후, 상단 메뉴의 Application > 어플리케이션 등록을 진행하자.

어플리케이션 이름은 적당한 이름을 사용하고, 사용 API는 네이버 로그인을 선택한다. 해당을 선택하면 제공 정보 선택 박스가 나오면, 회원이름, 연락처 이메일 주소, 별명, 프로필 사진을 필수에 체크한 뒤 로그인 오픈 API 서비스 환경에 PC 웹을 선택한다.

서비스 URL은 구글과 마찬가지로 `http://localhost:8088` (혹은 본인의 다른 url), Callback URL은 `http://localhost:8088/login/oauth2/code/naver` 를 입력하면 된다.

어플리케이션 등록이 완료되면 개요 탭 > 애플리케이션 정보 항목에서 Client ID와 Client Secret 정보를 확인할 수 있는데, 구글과 마찬가지로 yml에 설정을 추가로 등록한다. 단, 네이버는 Spring Security에서 프로바이더를 기본적으로 제공하지 않기 때문에 yml에 프로바이더 정보를 같이 입력해야 한다.

```
spring:
  security:
    oauth2:
      client:
        registration:
          google: # /oauth2/authorization/google 이 주소를 동작하게 한다.
            clientId:
1073085804262-5ik2cjma4soc692vkkct5v5qanqtse4r.apps.googleusercontent.com
            clientSecret: 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
            scope:
              - email
              - profile

          naver:
            client-id: EAYpserNIN15cnzowC2O
            client-secret: 8gEQNfcAoH
```



```
scope:
  - name
  - email
client-name: Naver
authorization-grant-type: authorization_code
redirect-uri: http://localhost/login/oauth2/code/naver
#네이버는 수정가능하나 규칙을 맞추자
```

provider:

naver:

```
authorization-uri: https://nid.naver.com/oauth2.0/authorize
#로그인창 보이기
token-uri: https://nid.naver.com/oauth2.0/token
user-info-uri: https://openapi.naver.com/v1/nid/me
#프로필 정보 받을 곳
user-name-attribute: response
#회원정보를 json으로 받아올 때 response라는 키값으로 리턴.
```

● 로그인 폼에 네이버 로그인 링크 붙이기

```
<a href="/oauth2/authorization/google">구글로그인</a>
<a href="/oauth2/authorization/naver">네이버로그인</a>
<a href="/joinForm">회원가입하기</a>
```

네이버의 경우 로그인에 대한 응답으로 response라는 이름의 컬렉션을 던지기 때문에 내부에서 한번 풀어주는 과정을 거친다.

프로바이더에 naver를 추가로 등록하면 해당 내용을 자동으로 Spring Security가 인식하여 로그인 화면의 Oauth2부분에 네이버 로그인을 추가해준다.

기능을 테스트해보면 잘 넘어가는 것을 확인할 수 있을 것이다.

● 구글과 네이버 로그인 정보를 얻기 위해 Repository에 Query Method를 추가한다.

```

public interface UsersRepository extends JpaRepository<User, Long> {
    // SELECT * FROM user WHERE username = ?1
    User findByUsername(String username);

    // SELECT * FROM user WHERE provider = ?1 and providerId = ?2
    Optional<User> findByProviderAndProviderId
        (String provider, String providerId);
}

```

● 프로바이더를 구분하기 위한 Interface 생성(OAuth2UserInfo)

```

public interface OAuth2UserInfo {
    String getProviderId();
    String getProvider();
    String getEmail();
    String getName();
}

```

● GoogleUserInfo

```

public class GoogleUserInfo implements OAuth2UserInfo{
    private Map<String, Object> attributes;

    public GoogleUserInfo(Map<String, Object> attributes) {
        this.attributes = attributes;
    }

    @Override
    public String getProviderId() {
        return (String)attributes.get("sub") ;
    }

    @Override
    public String getProvider() {
        return "google";
    }
}

```

```

    }

    @Override
    public String getEmail() {
        return (String) attributes.get("email");
    }

    @Override
    public String getName() {
        return (String) attributes.get("name");
    }
}

```

● NaverUserInfo

```

public class NaverUserInfo implements OAuth2UserInfo{
    private Map<String, Object> attributes;

    public NaverUserInfo(Map<String, Object> attributes) {
        this.attributes = attributes;
    }

    @Override
    public String getProviderId() {
        return (String) attributes.get("id");
    }

    @Override
    public String getProvider() {
        return "naver";
    }

    @Override
    public String getEmail() {

```

```
        return (String) attributes.get("email");
    }

    @Override
    public String getName() {
        return (String) attributes.get("name");
    }
}
```

● PrincipalOAuth2UserService

▶ **loadUser** 함수의 역할

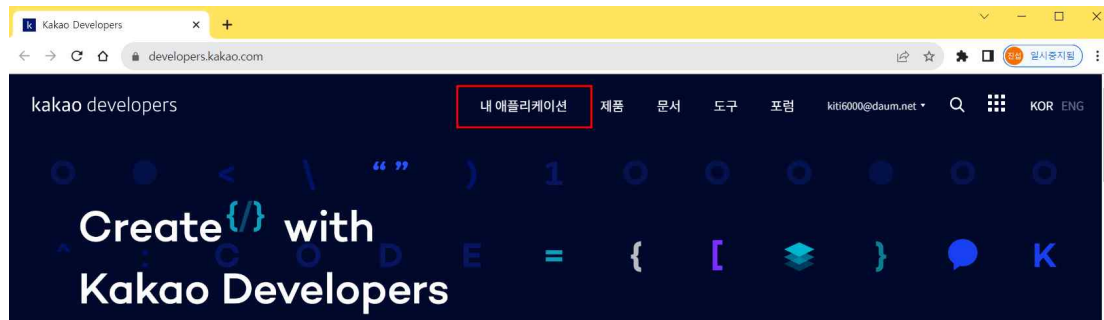
구글 로그인 버튼 클릭 -> 구글 로그인 -> 로그인 성공 -> code를 리턴
(OAuth-Client Library) -> Access Token 요청

userRequest 정보 -> loadUser함수 -> 프로바이더로부터 회원 프로필 받음

3. 카카오 로그인

kakao를 이용하여 로그인을 구현하기 위해서는 먼저 사용하려는 서비스 애플리케이션 정보를 등록해야 함.


1. kakao developer 접속(<https://developers.kakao.com/>)



2. 애플리케이션 추가하기

애플리케이션 추가하기를 누르면 아래와 같은 화면이 나오고,
정보들을 모두 입력하고 저장을 눌러주면 다음과 같이 애플리케이션이 추가 됨.

애플리케이션 추가하기

앱 아이콘	<div></div> <div>파일 선택 JPG, GIF, PNG 권장 사이즈 128px, 최대 250KB</div>
앱 이름	<input type="text" value="wooriboard"/>
사업자명	<input type="text" value="woori"/>

- 입력된 정보는 사용자가 카카오 로그인을 할 때 표시됩니다.
- 정보가 정확하지 않은 경우 서비스 이용이 제한될 수 있습니다.

☒ 서비스 이용이 제한되는 카테고리, 금지된 내용, 금지된 행동 관련 운영정책을 위반하지 않는 앱입니다.

취소

저장

● 애플리케이션 생성확인

전체 애플리케이션 (1)

애플리케이션 이름



애플리케이션 추가하기



wooriboard

ID 947854

OWNER

3. 사이트 도메인 등록

애플리케이션에 사용될 사이트 도메인을 등록. →

생성한 애플리케이션을 클릭한 뒤 좌측 메뉴에 플랫폼을 선택하고 Web 플랫폼 등록 버튼을 클릭.

추가 참고사항으로 개발을 하다 보면 등록되어야 할 도메인이 로컬 도메인, 개발 도메인, 운영 도메인까지 최소 3개가 됨.

http://localhost:8080

그럴 땐 다음과 같이 개행을 통해 입력을 하고, 첫 번째 작성한 도메인이 기본 도메인이 되기 때문에 기본 도메인을 운영 도메인으로 설정 함.

Web 플랫폼 등록

사이트 도메인

JavaScript SDK, 카카오톡 공유, 카카오맵, 메시지 API 사용시 등록이 필요합니다.

여러개의 도메인은 줄바꿈으로 추가해주세요. 최대 10까지 등록 가능합니다. 추가 등록은 포럼(데브톡)으로 문의주세요.

예시: (O) https://example.com (X) https://www.example.com

https://운영도메인
https://개발도메인
http://localhost:8088

4. Redirect URI 등록하기

Redirect URI는 로그인이 성공적으로 이루어졌을 때 카카오에서 결과값을 담은 데이터를 넘겨주는 우리가 개발할 서비스 경로이다. 먼저 도메인을 등록한 정보 아래에 있는 등록 하러 가기 링크를 클릭.

Web		삭제	수정
사이트 도메인	http://localhost		
• 카카오 로그인 사용 시 Redirect URI를 등록해야 합니다. 등록하러 가기			

링크를 클릭하면 나오는 화면에서는 먼저 활성화 설정 상태를 ON으로 변경.

활성화 설정

상태	ON
----	----

그리고 스크롤을 아래로 내려서 Redirect URI 등록 버튼을 클릭하여 다음과 같이 등록 → 등록되는 URI는 위에서 등록했던 도메인을 기준으로 추가 경로 입력

http://localhost/kakao/login

Redirect URI 부분도 도메인 등록할 때와 동일.

로컬, 개발, 운영까지 최소 3개의 URI이 등록될 것이며 이럴 땐 다음과 같이 개행을 통해 입력.

http://운영도메인/kakao/login
http://개발도메인/kakao/login
http://localhost/kakao/login

5. 개인정보 동의 항목 설정

개인정보 동의 항목 같은 경우는 사용자가 로그인을 했을 때 개발자가 활용할 수 있

는 정보들을 등록하는 곳

좌측 메뉴에 동의 항목을 선택한 뒤 활용하기 원하는 정보들을 다음과 같이 설정

제품 설정	
카카오 로그인	
동의항목	
간편가입	
카카오톡 채널	
개인정보 국외이전	
연결 끊기	
사용자 프로퍼티	
보안	
보안 이벤트	
고급	

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

6. 애플리케이션 앱 키 확인

모든 설정이 끝났으면 좌측 메뉴에서 요약 정보를 선택하여 애플리케이션 앱 키들을 확인.

요약 정보	앱 키
일반	
비즈니스	
앱 키	

네이티브 앱 키	ad495ecda3c16d1931b7ca88fa4c846e
REST API 키	3b5261251a7bb4e09841331127aafc3b

7. 카카오 로그인 API 요청하기

```
kauth.kakao.com/oauth/authorize?client_id={REST_API_KEY}&redirect_uri={REDIRECT_URI}&response_type=code
```

```
kauth.kakao.com/oauth/authorize?client_id=3b5261251a7bb4e09841331127aafc3b&redirect_uri=http://localhost/kakao/login&response_type=code
```

- REST_API_KEY는 아까전에 얻은 REST KEY 키를 기입하고,
- REDIRECT_URI는 code를 반환할 URI를 기입하면 된다.



wooriboard
woori

✓ 전체 동의하기

전체동의를 선택목적에 대한 동의를 포함하고 있으며, 선택 목적에 대한 동의를 거부해도 서비스 이용이 가능합니다.



kiti6000@daum.net

[계정 변경](#)

wooriboard 서비스 제공을 위해 회원번호와 함께 개인정보가 제공됩니다. 보다 자세한 개인정보 제공항목은 동의 내용에서 확인하실 수 있습니다. 해당 정보는 동의 철회 또는 서비스 탈퇴 시 지체없이 파기됩니다.

✓ [필수] 카카오 개인정보 제3자 제공 동의
닉네임

[보기](#)

동의하고 계속하기

● application.yml

kakao:

client-id: 3b5261251a7bb4e09841331127aafc3b

clientSecret: FHMAwpnlG92IIGN3E7pljFKDWr3JK9G

redirectUri: http://localhost/login/oauth2/code/kakao

client-authentication-method: client_secret_post

authorization-grant-type: authorization_code

client-name: Kakao

scope:

- profile_nickname
- account_email

provider:

naver:

authorization-uri: <https://nid.naver.com/oauth2.0/authorize>

#로그인창 보이기

token-uri: <https://nid.naver.com/oauth2.0/token>
user-info-uri: <https://openapi.naver.com/v1/nid/me>
#프로필 정보 받을 곳
user-name-attribute: response
#회원정보를 json으로 받아올 때 response라는 키값으로 리턴.

kakao:

authorization_uri: <https://kauth.kakao.com/oauth/authorize>
token_uri: <https://kauth.kakao.com/oauth/token>
user-info-uri: <https://kapi.kakao.com/v2/user/me>
user_name_attribute: id

● 로그인 링크 설정

```
<a href="/oauth2/authorization/google">구글로그인</a>  
<a href="/oauth2/authorization/naver">네이버로그인</a>  
<a href="/oauth2/authorization/kakao">카카오로그인</a>  
<a href="/joinForm">회원가입하기</a>
```

● 게시판 서비스에 카카오로그인 붙이기

▶ SecurityConfig

```
.oauth2Login(oAuth -> oAuth
    .loginPage("/loginForm")
    .defaultSuccessUrl("/articles/lists")
    .userInfoEndpoint(userInfo-> userInfo
        .userService(principalOAuth2UserService)))
```

▶ application.yml

```
server:
  port: 80

spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/exam_board?serverTimezone=Asia/Seoul
    username: root
    password: 1111

  jpa:
    hibernate:
      ddl-auto: validate #create update none
    show-sql: true

---

spring:
  security:
    oauth2:
      client:
        registration:
          google: # /oauth2/authorization/google 이 주소를 동작하게 한다.
            clientId:
```

1073085804262-5ik2cjma4soc692vkkct5v5qanqtse4r.apps.googleusercontent.com

clientSecret: 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

scope:

- email
- profile

naver:

client-id: EAYpserNIN15cnzowC2O

client-secret: 8gEQNfcAoH

scope:

- name
- email

client-name: Naver

authorization-grant-type: authorization_code

redirect-uri: http://localhost/login/oauth2/code/naver

#네이버는 수정가능하나 규칙을 맞추자

kakao:

client-id: 3b5261251a7bb4e09841331127aafc3b

clientSecret: FHMAwpnlG E92IIGN3E7pljFKDWr3JK9G

redirectUri: http://localhost/login/oauth2/code/kakao

client-authentication-method: client_secret_post

authorization-grant-type: authorization_code

client-name: Kakao

scope:

- profile_nickname
- account_email

provider:

naver:

authorization-uri: <https://nid.naver.com/oauth2.0/authorize>

#로그인창 보이기

token-uri: https://nid.naver.com/oauth2.0/token

```
user-info-uri: https://openapi.naver.com/v1/nid/me  
#프로필 정보 받을 곳  
user-name-attribute: response  
#회원정보를 json으로 받아올 때 response라는 키값으로 리턴.
```

kakao:

```
authorization_uri: https://kauth.kakao.com/oauth/authorize  
token_uri: https://kauth.kakao.com/oauth/token  
user-info-uri: https://kapi.kakao.com/v2/user/me  
user_name_attribute: id
```

▶ header.html

```
<a role="button" id="kakao-login" class="btn btn-outline-light me-2"  
    sec:authorize="isAnonymous()"  
    th:href="@{/oauth2/authorization/kakao}">  
      
</a>
```

▶ PrincipalOAuth2UserService

```
@Service  
public class PrincipalOAuth2UserService extends DefaultOAuth2UserService {  
  
    @Autowired  
    private UserAccountRepository userRepository;  
  
    // userRequest 는 code를 받아서 accessToken을 응답 받은 객체  
    // 함수 종료 시 @AuthenticationPrincipal Annotation 생성  
    @Override  
    public OAuth2User loadUser(OAuth2UserRequest userRequest) throws  
        OAuth2AuthenticationException {  
        OAuth2User oAuth2User = super.loadUser(userRequest);
```

```

// OAuth2 의 회원 프로필 조회

String email =
    (String) ((Map)oAuth2User.getAttribute("kakao_account")).get("email");
String nickname =
    (String) ((Map)oAuth2User.getAttribute("properties")).get("nickname");
String provider = userRequest.getClientRegistration().getClientId();
String username = provider + "_" + email;
String password = UUID.randomUUID().toString();

Optional<UserAccount> _user = userRepository.findByUserId(username);

if (_user.isEmpty()) {
    // user의 패스워드가 null이기 때문에 OAuth 유저는 일반적인
    // 로그인할 수 없음.
    UserAccount user = new UserAccount();
    user.setUserId(username);
    user.setEmail(email);
    user.setNickname(nickname);
    user.setUserRole(UserRole.USER);
    user.setUserPassword(password);
    userRepository.save(user);
    return new PrincipalDetails(user, oAuth2User.getAttributes());
} else {
    UserAccount user = _user.get();
    return new PrincipalDetails(user, oAuth2User.getAttributes());
}
}
}

```

▶ PrincipalDetails

```

public class PrincipalDetails implements UserDetails, OAuth2User {
    private UserAccount user;
}

```

```
private Map<String, Object> attributes;

public PrincipalDetails(UserAccount user, Map<String, Object> attributes) {
    this.user = user;
    this.attributes = attributes;
}

public PrincipalDetails(UserAccount user) {
    this.user = user;
}

public UserAccount getUser(){
    return user;
}

@Override
public Map<String, Object> getAttributes() {
    return attributes;
}

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    Collection<GrantedAuthority> collect = new ArrayList<>();
    collect.add(()-> {return user.getUserRole().getValue();});
    return collect;
}

@Override
public String getPassword() {
    return user.getUserPassword();
}

@Override
public String getUsername() {
```

```
        return user.getUserId();
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    @Override
    public String getName() {
        String sub = attributes.get("id").toString();
        return sub;
    }
}
```