1월 20일 회고록

생성일	@2025년 1월 20일 오후 10:36
태그	

▼ CustomOAuth2UserService.class

▼ 코드 스니펫

```
@Service
@RequiredArgsConstructor
public class CustomOAuth2UserService extends DefaultO
Auth2UserService {

    private final UserRepository userRepository;

    @Override
    public OAuth2User loadUser(OAuth2UserRequest user
Request) throws OAuth2AuthenticationException {
        OAuth2User user = super.loadUser(userReques
t);

        try {
            return this.process(userRequest, user);
        } catch(AuthenticationException ex) {
            throw ex;
        } catch (Exception ex) {
            ex.printStackTrace();
            throw new InternalAuthenticationServiceEx
ception(ex.getMessage(), ex.getCause());
        }
    }

    private OAuth2User process(OAuth2UserRequest user
Request, OAuth2User user) {
        ProviderType providerType = ProviderType.valu
```

```
eOf(userRequest.getClientRegistration().getRegistrati
onId().toUpperCase());

        OAuth2UserInfo userInfo = OAuth2UserInfoFacto
ry.getOAuth2UserInfo(providerType, user.getAttributes
());
        User savedUser = userRepository.findByUserId
(userInfo.getId());

        if (savedUser != null) {
            if(providerType != savedUser.getProviderT
ype()) {
                throw new OAuthProviderMissMatchExcep
tion(
                        "Looks like you're signed up
with " + providerType +
                        " account. Please use your "
+ savedUser.getProviderType() + " account to login."
                );
            }
            updateUser(savedUser, userInfo);
        } else {
            savedUser = createUser(userInfo, provider
Type);
        }

        return UserPrincipal.create(savedUser, user.g
etAttributes());
    }

    private User createUser(OAuth2UserInfo userInfo,
ProviderType providerType) {
        LocalDateTime now = LocalDateTime.now();
        User user = new User(
                userInfo.getId(),
                userInfo.getName(),
                userInfo.getEmail(),
                "Y",
```

```
                userInfo.getImageUrl(),
                providerType,
                RoleType.USER,
                now,
                now
        );

        return userRepository.saveAndFlush(user);
    }

    private User updateUser(User user, OAuth2UserInfo
userInfo) {
        if(userInfo.getName() != null && !user.getUse
rname().equals(userInfo.getName())) {
            user.setUsername(userInfo.getName());
        }

        if(userInfo.getImageUrl() != null && !user.ge
tProfileImageUrl().equals(userInfo.getImageUrl())) {
            user.setProfileImageUrl(userInfo.getImage
Url());
        }

        return user;
    }
}
```

1. 초기 로그인 요청
    - 클라이언트 : 사용자가 Google 로그인 버튼 클릭
    - 자체 서버 : OAuth2 클라이언트가 인증 URL을 생성하여 사용자를 Provider 서버로 리다이렉트
    - Provider 서버 : 개입 x
2. 사용자 동의
    - 클라이언트 : Provider의 동의 화면이 표시됨

- 자체 서버 : 개입 x
- Provider 서버 : 동의 화면을 보여주고 사용자의 동의를 받음

3. 인증 코드 발급
    - 클라이언트 : 동의 후 자동으로 리다이렉트 됨
    - 자체 서버 : 설정된 redirect URI로 인증 코드를 받음
    - Provider : 인증 코드를 생성하여 redirect URI로 전달
4. 액세스 토큰 교환
    - 클라이언트 : 개입 x
    - 자체 서버 : 받은 인증 코드로 Provider 서버에 액세스 토큰 요청
    - Provider 서버 : 인증 코드 검증 후 액세스 토큰 발급
5. 사용자 정보 조회 (loadUser 호출 시점)
    - 클라이언트 : 개입 x
    - 자체 서버 :
        1. 액세스 토큰으로 Provider 서버에 사용자 정보 요청
        2. loadUser() 메서드 호출됨
        3. 받은 정보로 자체 DB에 사용자 생성 또는 업데이트
        4. JWT 토큰 등 자체 인증 토큰 생성
    - Provider 서버 : 액세스 토큰을 검증하고 요청된 사용자 정보 전달
6. 로그인 완료
    - 클라이언트 : 자체 서버로부터 받은 인증 토큰으로 로그인 완료
    - 자체 서버 : 생성된 인증 토큰은 클라이언트에 전달
    - Provider : 개입 x

여기서 loadUser() 메서드는 5번 단계에서 다음과 같이 실행됩니다.

1. 액세스 토큰으로 Provider 서버에서 사용자 정보를 조회할 때
2. 조회된 정보를 바탕으로 자체 서비스의 사용자 정보를 처리할 때 이 두 작업을 수행합니다.

이전 단계들(1~4)은 Spring Security OAuth2 Client가 자동으로 처리하므로 우리가 작성한 CustomOAuth2UserService는 오직 사용자 정보 처리 단계에서만 실행됩니다.

▼ application.yml 설정

▼ 코드 스니펫

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/login?useSSL=fal
se&serverTimezone=UTC&useLegacyDatetimeCode=false&all
owPublicKeyRetrieval=true
    username: root
    password: dudgks56
    hikari:
      pool-name: jpa-hikari-pool
      maximum-pool-size: 5
      jdbc-url: ${spring.datasource.url}
      username: ${spring.datasource.username}
      password: ${spring.datasource.password}
      driver-class-name: ${spring.datasource.driver-c
lass-name}
      data-source-properties:
        rewriteBatchedStatements: true
  # JPA 설정
  jpa:
    database-platform: org.hibernate.dialect.MySQLDia
lect
    generate-ddl: true
    hibernate:
      ddl-auto: create
    show-sql: true
    properties:
      hibernate:
        format_sql: true
        default_batch_fetch_size: 100
        jdbc.batch_size: 20
```

```
        order_inserts: true
        order_updates: true
  # Security OAuth
  security:
    oauth2.client:
      registration:
        google:
          clientId: '630701881868-ldmfn4d0br9k9lo1c37
qjpk3r0p05fjc.apps.googleusercontent.com'
          clientSecret: 'GOCSPX-fVcOQK1k9xlMYnFB4YD5E
cUbjX8z'
          scope:
            - email
            - profile
#        facebook:
#          clientId: '{페이스북 client-id}'
#          clientSecret: '{페이스북 client-secret}'
#          scope:
#            - email
#            - public_profile
        naver:
          clientId: 'pKfJJr7XidGCz1FGMWKe'
          clientSecret: 'mqi6MQE1Ou'
          clientAuthenticationMethod: post
          authorizationGrantType: authorization_code
          redirectUri: "http://localhost:8080/login/o
auth2/code/naver"
          scope:
            - nickname
            - email
            - profile_image
          clientName: Naver
#        kakao:
#          clientId: '{카카오 client-id}'
#          clientSecret: '{카카오 client-secret}'
#          clientAuthenticationMethod: post
#          authorizationGrantType: authorization_code
#          redirectUri: "{baseUrl}/{action}/oauth2/co
```

```
de/{registrationId}"
#          scope:
#            - profile_nickname
#            - profile_image
#            - account_email
#          clientName: Kakao
      # Provider 설정
      provider:
        naver:
          authorizationUri: https://nid.naver.com/oau
th2.0/authorize
          tokenUri: https://nid.naver.com/oauth2.0/to
ken
          userInfoUri: https://openapi.naver.com/v1/n
id/me
          userNameAttribute: response
#        kakao:
#          authorizationUri: https://kauth.kakao.com/
oauth/authorize
#          tokenUri: https://kauth.kakao.com/oauth/to
ken
#          userInfoUri: https://kapi.kakao.com/v2/use
r/me
#          userNameAttribute: id

# cors 설정
cors:
  allowed-origins: 'http://localhost:3000'
  allowed-methods: GET,POST,PUT,DELETE,OPTIONS
  allowed-headers: '*'
  max-age: 3600

# jwt secret key 설정
jwt.secret: '8sknjlO3NPTBqo319DHLNqsQAfRJEdKsETOds'

# 토큰 관련 secret Key 및 RedirectUri 설정
app:
  auth:
```

```
    tokenSecret: 926D96C90030DD58429D2751AC1BDBBC
    tokenExpiry: 1800000
    refreshTokenExpiry: 604800000
  oauth2:
    authorizedRedirectUris:
      - http://localhost:3000/oauth/redirect
```

Provider 설정은 각 OAuth2 제공자(Naver, Kakao 등)의 고유한 엔드포인트 정보를 Spring Security에 알려주기 위해 필요합니다.

Google, Facebook과 같은 일반적인 제공자들은 Spring Security OAuth2에 기본으로 등록되어 있어서 별도 설정이 필요 없습니다. 하지만 **Naver, Kakao** 같은 **국내 서비스는 기본으로 등록되어 있지 않아 수동으로 설정**해야 합니다.

Provider 설정에서 정의하는 주요 URL들:

```
provider:
  naver:
    authorizationUri: <https://nid.naver.com/oauth2.0/au
thorize>  # 사용자 동의 화면 URL
    tokenUri: <https://nid.naver.com/oauth2.0/token>
# 액세스 토큰 발급 URL
    userInfoUri: <https://openapi.naver.com/v1/nid/me>
# 사용자 정보 조회 URL
    userNameAttribute: response
# 사용자 정보의 식별자 필드
```

각 URL이 사용되는 시점:

1. `authorizationUri`: 사용자가 소셜 로그인 버튼을 클릭했을 때 리다이렉트되는 네이버 동의 화면 주소
2. `tokenUri`: 인증 코드를 액세스 토큰으로 교환할 때 사용되는 주소
3. `userInfoUri`: 발급받은 액세스 토큰으로 사용자 정보를 조회할 때 사용되는 주소
4. `userNameAttribute`: 제공자마다 사용자 정보를 담는 JSON 구조가 다른데, 네이버의 경우 실제 사용자 정보가 response 필드 안에 있어서 이를 명시

이렇게 Provider 설정을 해주면 Spring Security OAuth2 Client가 이 정보들을 바탕으로 OAuth2 인증 흐름을 자동으로 처리할 수 있게 됩니다.

▼ Callback URL

Callback URL(또는 Redirect URI)은 OAuth2 인증 과정에서 Provider(네이버, 구글 등)가 인증 완료 후 인증 코드를 전달할 목적지 URL입니다.

실제 동작 과정을 보면:

1. 사용자가 "네이버로 로그인" 버튼 클릭
2. 네이버 로그인 화면으로 이동하여 로그인
3. 네이버가 **우리 서버의 Callback URL로** 인증 코드를 포함하여 리다이렉트

```
<http://localhost:8080/login/oauth2/code/naver?code=4IJ
F...&state=Rd2Gk>...
```

설정 방법:

1. 애플리케이션 설정

```
naver:
  redirectUri: "<http://localhost:8080/login/oauth2/cod
e/naver>"
```

1. 네이버 개발자 센터 설정

- 등록해야 할 Callback URL: `http://localhost:8080/login/oauth2/code/naver`

주의사항:

- **개발자 센터에 등록한 URL과 애플리케이션의 redirectUri가 정확히 일치해야 함**
- localhost 개발 환경과 실제 배포 환경의 URL이 다르면 둘 다 등록해야 함
- Spring Security의 기본 Callback URL 형식:
  - `http://[도메인]/login/oauth2/code/[provider-id]`
  - provider-id는 설정의 registration 아래 키 값(google, naver 등)

따라서 404 에러가 발생했다면:

1. 네이버 개발자 센터에서 Callback URL이 제대로 등록되어 있는지
2. application.yml의 redirectUri와 일치하는지
   확인이 필요합니다.