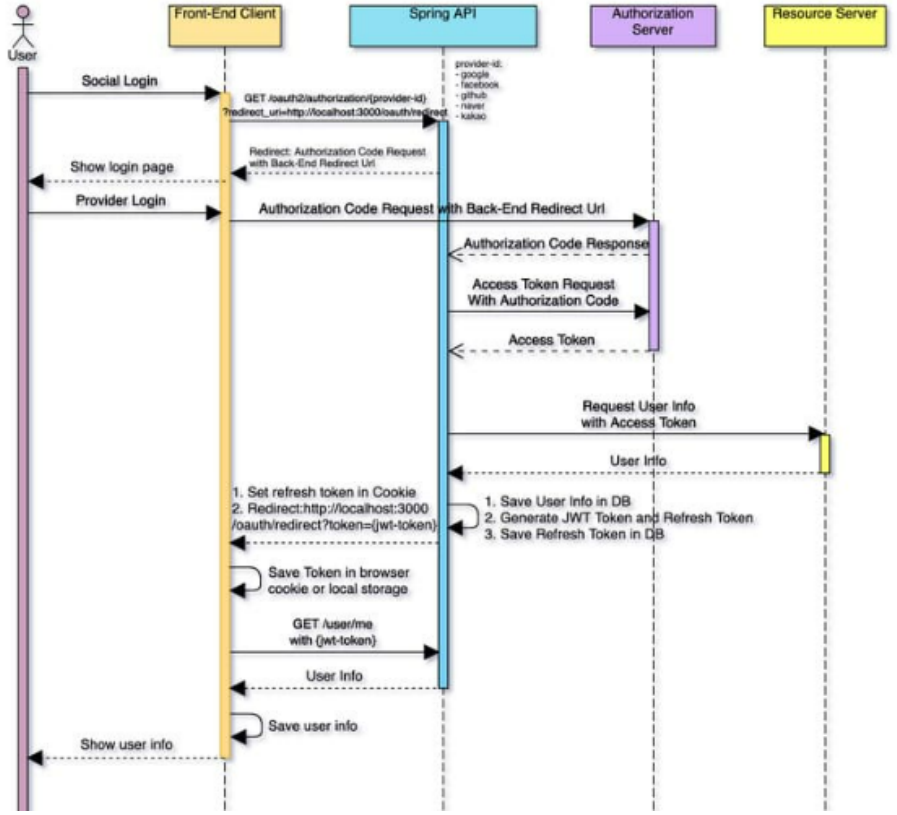
소셜 로그인

전체 시퀀스 다이어그램



시퀀스 설명

1. 소셜 로그인 요청
2. 백엔드로 GET `/oauth2/authorization/{provider-id}?redirect_uri=http://localhost:3000/oauth/redirect` 으로 OAuth 인가 요청
3. Provider 별로 Authorization Code 인증을 할 수 있도록 리다이렉트
4. 리다이렉트 화면에서 provider(구글) 서비스에 로그인(User가 Authorization Server로 로그인)
5. 로그인이 완료된 후, Authorization server로부터 백엔드로 **인가 코드** 응답받기
6. 백엔드에서 **인가 코드**를 이용하여 Authorization Server에 **액세스 토큰** 요청

7. 액세스 토큰 획득
8. 액세스 토큰을 이용하여 Resource Server에 **유저 데이터** 요청
9. 획득한 유저 데이터를 **DB에 저장** 후, **JWT 액세스 토큰과 리프레시 토큰**을 생성 후 **DB에 리프레시 토큰 저장**
10. 리프레시 토큰은 수정 불가능한 **쿠키에 저장**하고, 액세스 토큰은 프론트엔드 리다이렉트 URI에 **쿼리스트링**에 토큰을 담아 **리다이렉트**
11. 프론트엔드에서 **토큰을 저장** 후, **API 요청 시 헤더에** Authorization: Bearer {token}을 **추가**하여 요청
12. 백엔드에서는 토큰을 확인하여 **권한 확인**
13. **토큰이 만료**된 경우, **쿠키에 저장된 리프레시 토큰을 이용**하여 **액세스 토큰과 리프레시 토큰을 재발급**

UserRefreshToken

```
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "USER_REFRESH_TOKEN")
public class UserRefreshToken {
    @JsonIgnore
    @Id
    @Column(name = "REFRESH_TOKEN_SEQ")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long refreshTokenSeq;

    @Column(name = "USER_ID", length = 64, unique = true)
    @NotNull
    @Size(max = 64)
    private String userId;

    @Column(name = "REFRESH_TOKEN", length = 256)
    @NotNull
    @Size(max = 256)
```

```
    private String refreshToken;

    public UserRefreshToken(
            @NotNull @Size(max = 64) String userId,
            @NotNull @Size(max = 256) String refreshToken
    ) {
        this.userId = userId;
        this.refreshToken = refreshToken;
    }
}
```

CustomOAuth2UserService

```
@Service
@RequiredArgsConstructor
public class CustomOAuth2UserService extends DefaultOAuth2U
serService {

    private final UserRepository userRepository;

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

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

    private OAuth2User process(OAuth2UserRequest userReques
```

```
t, OAuth2User user) {
        ProviderType providerType = ProviderType.valueOf(us
erRequest.getClientRegistration().getRegistrationId().toUpp
erCase());

        OAuth2UserInfo userInfo = OAuth2UserInfoFactory.get
OAuth2UserInfo(providerType, user.getAttributes());
        User savedUser = userRepository.findByUserId(userIn
fo.getId());

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

        return UserPrincipal.create(savedUser, user.getAttr
ibutes());
    }

    private User createUser(OAuth2UserInfo userInfo, Provid
erType 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 userI
nfo) {
        if (userInfo.getName() != null && !user.getUsername
().equals(userInfo.getName())) {
            user.setUsername(userInfo.getName());
        }

        if (userInfo.getImageUrl() != null && !user.getProf
ileImageUrl().equals(userInfo.getImageUrl())) {
            user.setProfileImageUrl(userInfo.getImageUrl
());
        }

        return user;
    }
}
```

GoogleOAuth2UserInfo

```
public class GoogleOAuth2UserInfo extends OAuth2UserInfo {

    public GoogleOAuth2UserInfo(Map<String, Object> attribu
tes) {
        super(attributes);
    }

    @Override
    public String getId() {
```

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

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

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

    @Override
    public String getImageUrl() {
        return (String) attributes.get("picture");
    }
}
```