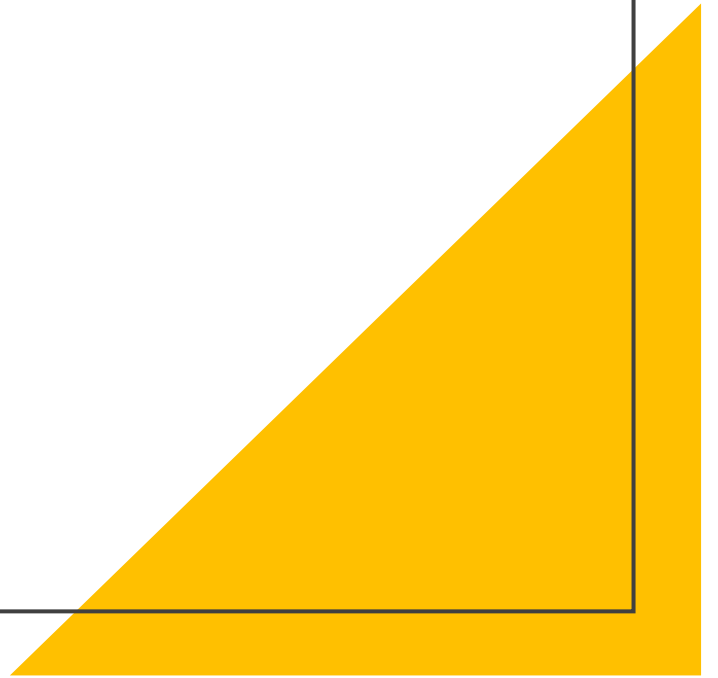


OAuth2

Spring Boot



구글 OAuth 서비스 등록

- <https://console.cloud.google.com/apis>



- 개요
- 브랜딩
- 대상
- 클라이언트**
- 데이터 액세스
- 인증 센터

← OAuth 클라이언트 ID 만들기

클라이언트 ID는 Google OAuth 서버에서 단일 앱을 식별하는 데 사용됩니다. 앱이 여러 플랫폼에서 실행되는 경우 각각 자체 클라이언트 ID가 있어야 합니다. 자세한 내용은 [OAuth 2.0 설정](#)을 참조하세요. OAuth 클라이언트 유형을 [자세히 알아보세요](#).

애플리케이션 유형 *

웹 애플리케이션

Android

Chrome 확장 프로그램

iOS

TV 및 입력 제한 기기

데스크톱 앱

Universal Windows Platform(UWP)



Google Cloud



test project

리소스, 문서, 제품 등 검색(/)



Google 인증 플랫폼 / 클라이언트 / 클라이언트 만들기



개요



브랜딩



대상



클라이언트



데이터 액세스



인증 센터



OAuth 클라이언트 ID 만들기

클라이언트 ID는 Google OAuth 서버에서 단일 앱을 식별하는 데 사용됩니다. 앱이 여러 플랫폼에서 실행되는 경우 각각 자체 클라이언트 ID가 있어야 합니다. 자세한 내용은 [OAuth 2.0 설정](#)을 참조하세요. OAuth 클라이언트 유형을 [자세히 알아보세요](#).

애플리케이션 유형 *

웹 애플리케이션



이름 *

springboot

OAuth 2.0 클라이언트의 이름입니다. 이 이름은 콘솔에서 클라이언트를 식별하는 용도로만 사용되며 최종 사용자에게 표시되지 않습니다.



아래에 추가한 URI의 도메인이 [승인된 도메인](#)으로 [OAuth 동의 화면](#)에 자동으로 추가됩니다.



개요



브랜딩



대상



클라이언트



데이터 액세스



인증 센터

프로젝트 구성

1 앱 정보

앱 이름 *

springboot

동의를 요청하는 앱의 이름

사용자 지원 이메일 *

ojiksb8022@gmail.com

사용자가 동의에 대해 문의하는 용도입니다. [자세히 알아보기](#)

다음

2 대상

3 연락처 정보

4 완료

<

만들기

취소



개요



브랜딩



대상



클라이언트



데이터 액세스



인증 센터

프로젝트 구성



내부 ?

조직 내 사용자만 사용할 수 있습니다. 인증을 위해 앱을 제출할 필요는 없습니다. [사용자 유형 자세히 알아보기](#)



외부 ?

Google 계정이 있는 모든 테스트 사용자가 사용할 수 있습니다. 앱이 테스트 모드로 시작되고 테스트 사용자 목록에 추가된 사용자에게만 제공됩니다. 앱을 프로덕션에 푸시할 준비가 되면 앱을 인증해야 할 수도 있습니다. [사용자 유형 자세히 알아보기](#)

다음



연락처 정보



완료

개요

브랜딩

대상

클라이언트

데이터 액세스

인증 센터

프로젝트 구성

✓ 앱 정보

✓ 대상

! 연락처 정보

이메일 주소 *

ojiksb8022@gmail.com ✕

purum01@naver.com ✕

이 이메일 주소는 Google에서 프로젝트 변경사항에 대해 알림을 보내기 위한 용도입니다.


다음


! 완료


만들기


취소


 개요

 브랜딩


 대상

 클라이언트


 데이터 액세스

 인증 센터


프로젝트 구성

 앱 정보

|

 대상

|

 연락처 정보

|

 완료

 [Google API 서비스: 사용자 데이터 정책](#)에 동의합니다.

계속

만들기

취소

개요

브랜딩

대상

클라이언트

데이터 액세스

인증 센터

← OAuth 클라이언트 ID 만들기

승인된 JavaScript 원본 [?]

브라우저 요청에 사용

+ URI 추가

승인된 리디렉션 URI [?]

웹 서버의 요청에 사용

URI 1 *


https://localhost:8080/login/oauth2/code/google




+ URI 추가


OAuth 클라이언트 생성됨

Google 인증 플랫폼의 클라이언트 탭에서 언제든지 클라이언트 ID에 액세스할 수 있습니다.

 OAuth 액세스는 [OAuth 동의 화면](#)에 나열된 [테스트 사용자](#)로 제한됩니다.

클라이언트 ID	347890029724-8g0a8m8ah35llea90ja7v9icle2r6vk5.ap ps.googleusercontent.com
----------	--

 2025년 6월부터는 이 대화상자를 닫으면 더 이상 클라이언트 보안 비밀번호를 보거나 다운로드할 수 없습니다. 아래 정보를 복사하거나 다운로드하여 안전하게 보관해야 합니다.

클라이언트 보안 비밀번호	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
생성일	2025년 9월 6일 PM 3시 50분 29초 GMT+9
상태	 사용 설정됨

[확인](#)

Google 인증 플랫폼 / 클라이언트

개요

브랜딩

대상

클라이언트

데이터 액세스

인증 센터

클라이언트

+ 클라이언트 만들기

삭제

삭제된 OAuth 클라이언트 복원

OAuth 2.0 클라이언트 ID

<input type="checkbox"/>	이름	생성일 ↓	유형	클라이언트 ID	작업
<input type="checkbox"/>	springboot	2025. 9. 6.	웹 애플리케이션	347890029724-8g0a...	

Google 인증 플랫폼 / 데이터 액세스

개요

브랜딩

대상

클라이언트

데이터 액세스

인증 센터

데이터 액세스

범위는 사용자에게 앱 승인을 요청하는 권한을 나타내며 프로젝트에서 사용자의 Google 계정에 있는 특정 유형의 비공개 사용자 데이터에 액세스하도록 허용합니다. [자세히 알아보기](#)

범위 추가 또는 삭제

× 선택한 범위 업데이트

i 아래에는 사용 설정된 API의 범위만 나와 있습니다. 이 화면에 누락된 범위를 추가하려면 [Google API 라이브러리](#)에서 API를 찾아 사용 설정하거나 아래의 '붙여넣은 범위' 텍스트 상자를 사용하세요. 라이브러리에서 사용 설정한 새 API를 확인하려면 페이지를 새로고침하세요.

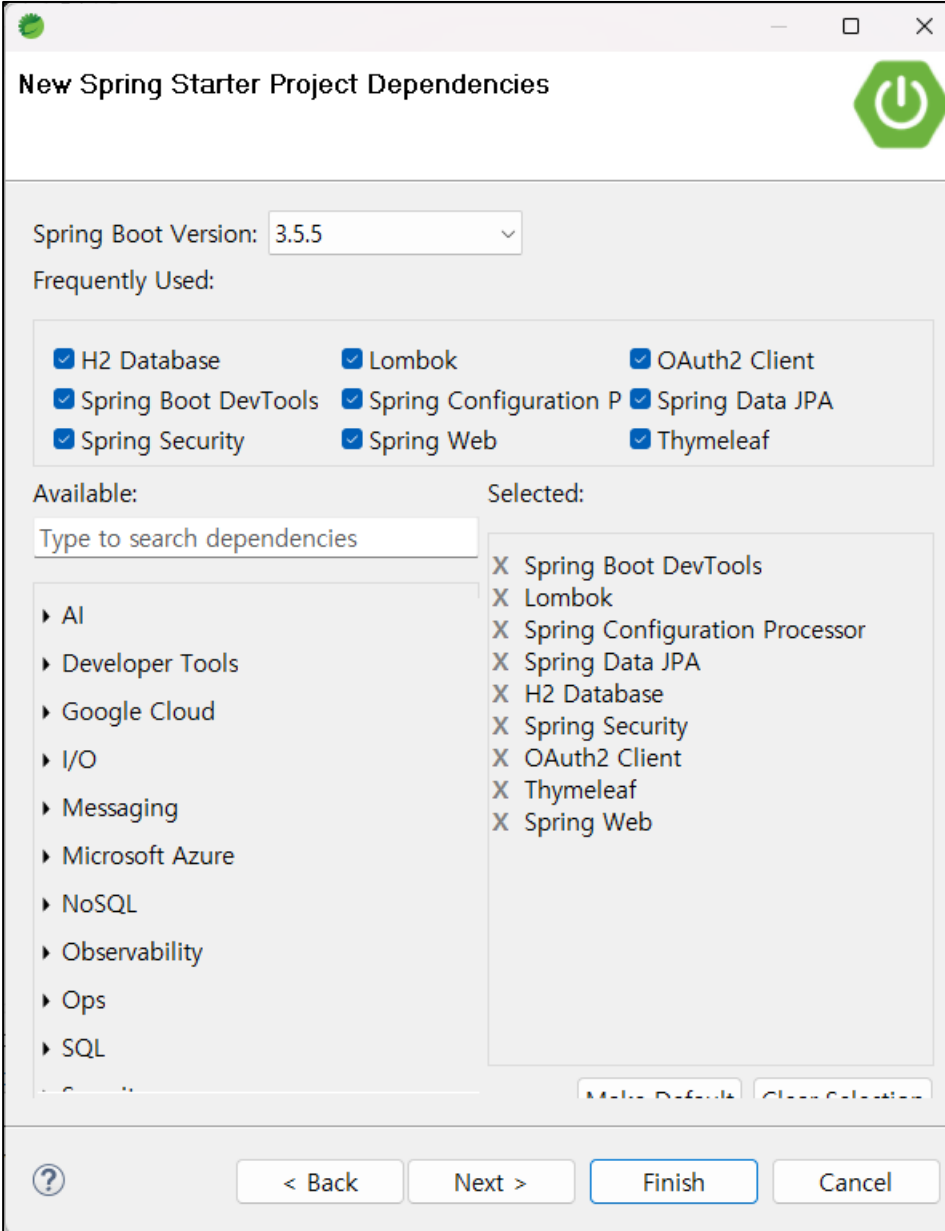
≡ 필터 속성 이름 또는 값 입력



<input type="checkbox"/>	API ↑	범위	사용자에게 표시되는 설명
<input checked="" type="checkbox"/>		.../auth/userinfo.email	기본 Google 계정의 이메일 주소 확인
<input checked="" type="checkbox"/>		.../auth/userinfo.profile	개인정보(공개로 설정한 개인정보 포함) 보기
<input checked="" type="checkbox"/>		openid	Google에서 내 개인 정보를 나와 연결
<input type="checkbox"/>	Analytics Hub API	.../auth/bigquery	View and manage your data in Google BigQuery and see the email address for your Google Account
<input type="checkbox"/>	Analytics Hub API	.../auth/cloud-platform	Google Cloud 데이터 확인, 수정, 구성, 삭제 및 Google 계정의 이메일 주소 확인
<input type="checkbox"/>	BigQuery API	.../auth/bigquery.readonly	Google BigQuery에서 데이터를 봅니다.
<input type="checkbox"/>	BigQuery API	.../auth/cloud-platform.read-only	Google Cloud 서비스 전체의 데이터 조회 및 Google 계정의 이메일 주소 확인
<input type="checkbox"/>	BigQuery API	.../auth/devstorage.full_control	Manage your data and permissions in Cloud Storage and see the email address for your Google Account

업데이트

New → Spring Starter Project



The image shows a 'New Spring Starter Project Dependencies' dialog box. At the top, it has a title bar with a green Spring logo, a minimize button, a maximize button, and a close button. The title 'New Spring Starter Project Dependencies' is on the left, and a green power button icon is on the right. Below the title bar, there is a 'Spring Boot Version:' label followed by a dropdown menu showing '3.5.5'. Underneath is the 'Frequently Used:' section, which contains a grid of nine checkboxes, all of which are checked: 'H2 Database', 'Spring Boot DevTools', 'Spring Security', 'Lombok', 'Spring Configuration Processor', 'Spring Web', 'OAuth2 Client', 'Spring Data JPA', and 'Thymeleaf'. Below this is the 'Available:' section, which has a search bar labeled 'Type to search dependencies' and a list of categories with expandable arrows: 'AI', 'Developer Tools', 'Google Cloud', 'I/O', 'Messaging', 'Microsoft Azure', 'NoSQL', 'Observability', 'Ops', and 'SQL'. To the right of the 'Available:' section is the 'Selected:' section, which contains a list of the same dependencies as the 'Frequently Used' section, each preceded by an 'X' mark. At the bottom of the 'Available:' and 'Selected:' sections are two buttons: 'Make Default' and 'Clear Selection'. At the very bottom of the dialog box are four buttons: a help button with a question mark icon, a '< Back' button, a 'Next >' button, and a 'Finish' button (which is highlighted with a blue border). To the right of the 'Finish' button is a 'Cancel' button.

New Spring Starter Project Dependencies

Spring Boot Version: 3.5.5

Frequently Used:

- ☒ H2 Database
- ☒ Lombok
- ☒ OAuth2 Client
- ☒ Spring Boot DevTools
- ☒ Spring Configuration Processor
- ☒ Spring Data JPA
- ☒ Spring Security
- ☒ Spring Web
- ☒ Thymeleaf

Available:

Type to search dependencies

- ▶ AI
- ▶ Developer Tools
- ▶ Google Cloud
- ▶ I/O
- ▶ Messaging
- ▶ Microsoft Azure
- ▶ NoSQL
- ▶ Observability
- ▶ Ops
- ▶ SQL

Selected:

- X Spring Boot DevTools
- X Lombok
- X Spring Configuration Processor
- X Spring Data JPA
- X H2 Database
- X Spring Security
- X OAuth2 Client
- X Thymeleaf
- X Spring Web

Make Default Clear Selection

? < Back Next > Finish Cancel

OAuth 라이브러리 추가

```
<!-- OAuth -->
```

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
```

```
</dependency>
```

application.properties

```
spring.application.name=edu_0Auth
```

```
#H2
```

```
spring.datasource.url=jdbc:h2:tcp://localhost/~ /test
```

```
spring.datasource.driver-class-name=org.h2.Driver
```

```
spring.datasource.username=sa
```

```
spring.datasource.password=
```

```
spring.jpa.hibernate.ddl-auto=update
```

```
spring.jpa.show-sql=true
```

```
spring.jpa.properties.hibernate.format_sql=true
```

```
# Google OAuth 설정
```

```
spring.security.oauth2.client.registration.google.client-id= 구글 클라이언트 ID
```

```
spring.security.oauth2.client.registration.google.client-secret= 구글 클라이언트 Secret
```

```
spring.security.oauth2.client.registration.google.scope=profile,email
```

```
package com.edu.oauth.config;
```

```
@Configuration
```

```
@EnableWebSecurity
```

```
public class SecurityConfig {
```

```
    @Bean
```

```
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
        http
```

```
            .authorizeHttpRequests(authorizeRequests ->  
                                    authorizeRequests.requestMatchers("/").permitAll()  
                                    .anyRequest().authenticated() )
```

```
            .oauth2Login(oauth2Login -> oauth2Login.loginPage("/login") );
```

```
        return http.build();
```

```
    }
```

```
    @Bean
```

```
    public PasswordEncoder passwordEncoder() {
```

```
        return new BCryptPasswordEncoder();
```

```
    }
```

```
}
```



```
package com.edu.controller;
```

```
import org.springframework.security.core.annotation.AuthenticationPrincipal;  
import org.springframework.security.oauth2.core.user.OAuth2User;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;
```

```
@Controller
```

```
public class HomeController {
```

```
    @GetMapping("/")
```

```
    public String home(Model model, @AuthenticationPrincipal OAuth2User oauth2User) {
```

```
        if (oauth2User != null) {
```

```
            model.addAttribute("name", oauth2User.getAttribute("name"));
```

```
        }
```

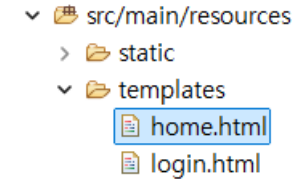
```
        return "home";
```

```
    }
```

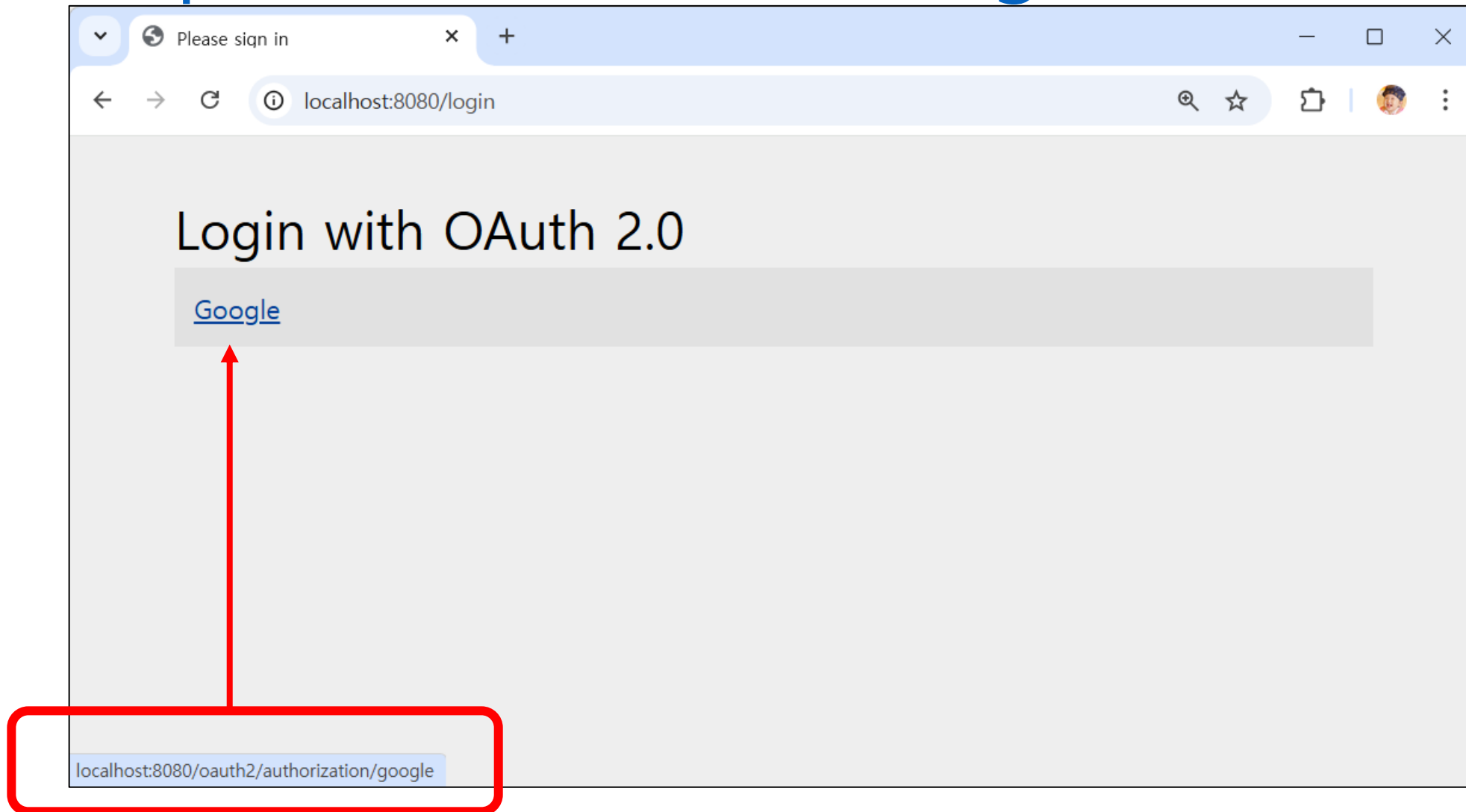
```
}
```

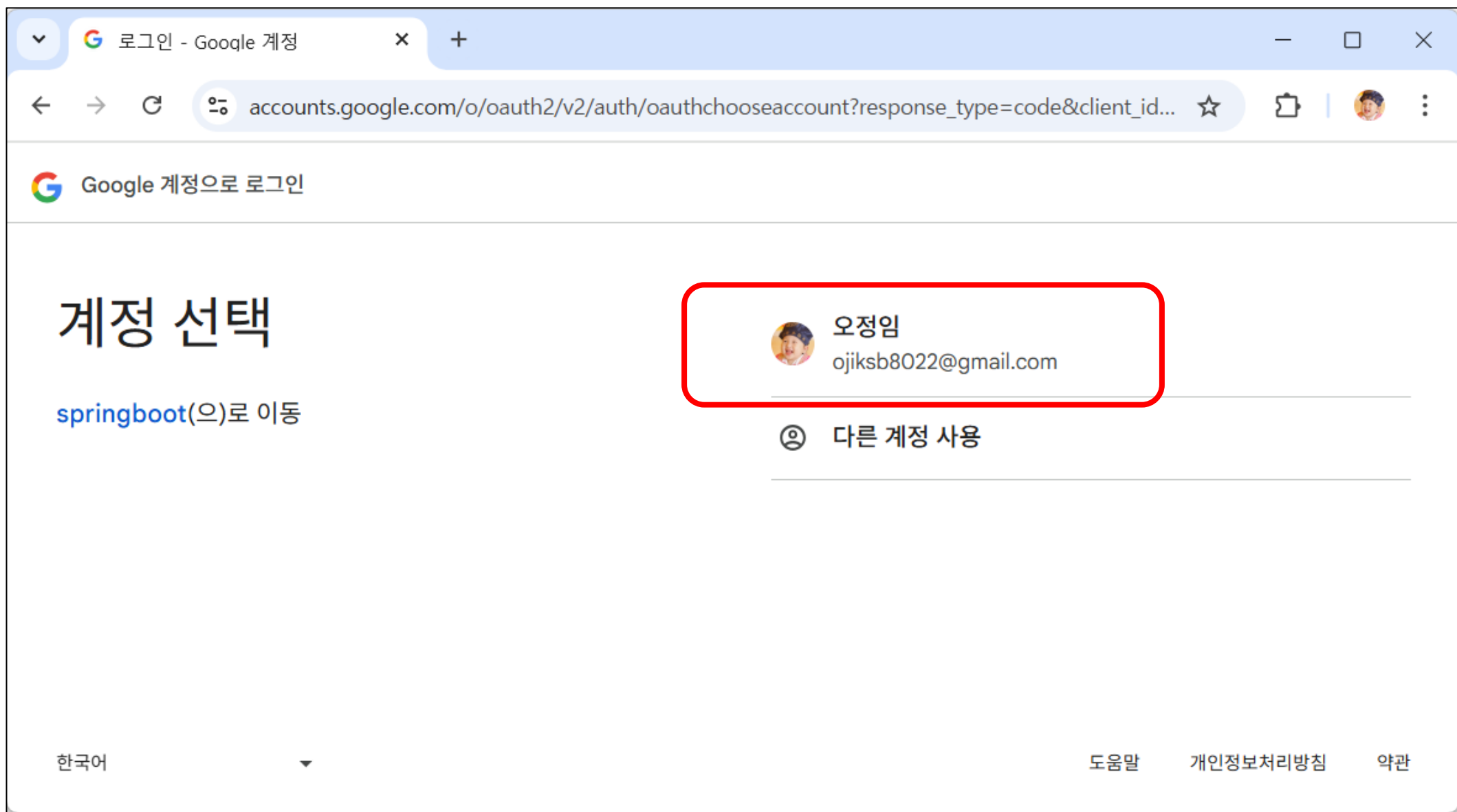
home.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>OAuth2 Login Example</title>
</head>
<body>
  <div>
    <h1>Welcome</h1>
    <div th:if="${name}">
      <p>Hello, <span th:text="${name}">User</span>!</p>
      <a th:href="@{/logout}">Logout</a>
    </div>
    <div th:if="${name == null}">
      <a th:href="@{/oauth2/authorization/google}">Login with Google</a>
    </div>
  </div>
</body>
</html>
```



<http://localhost:8080/login> 접속






로그인 - Google 계정

accounts.google.com/signin/oauth/id?authuser=0&part=Aji8hAMiwwgq5fRTCtnuDsjxcuUoUIT...

Google 계정으로 로그인

springboot에 다시 로그인
하는 중입니다

 ojiksb8022@gmail.com ▼

springboot의 개인정보처리방침 및 서비스 약관을 검토하여
springboot에서 내 데이터를 처리하고 보호하는 방법을 알아보세요.

[Google 계정](#)에서 언제든지 변경할 수 있습니다.

[Google 계정으로 로그인](#)에 대해 자세히 알아보기

취소

계속

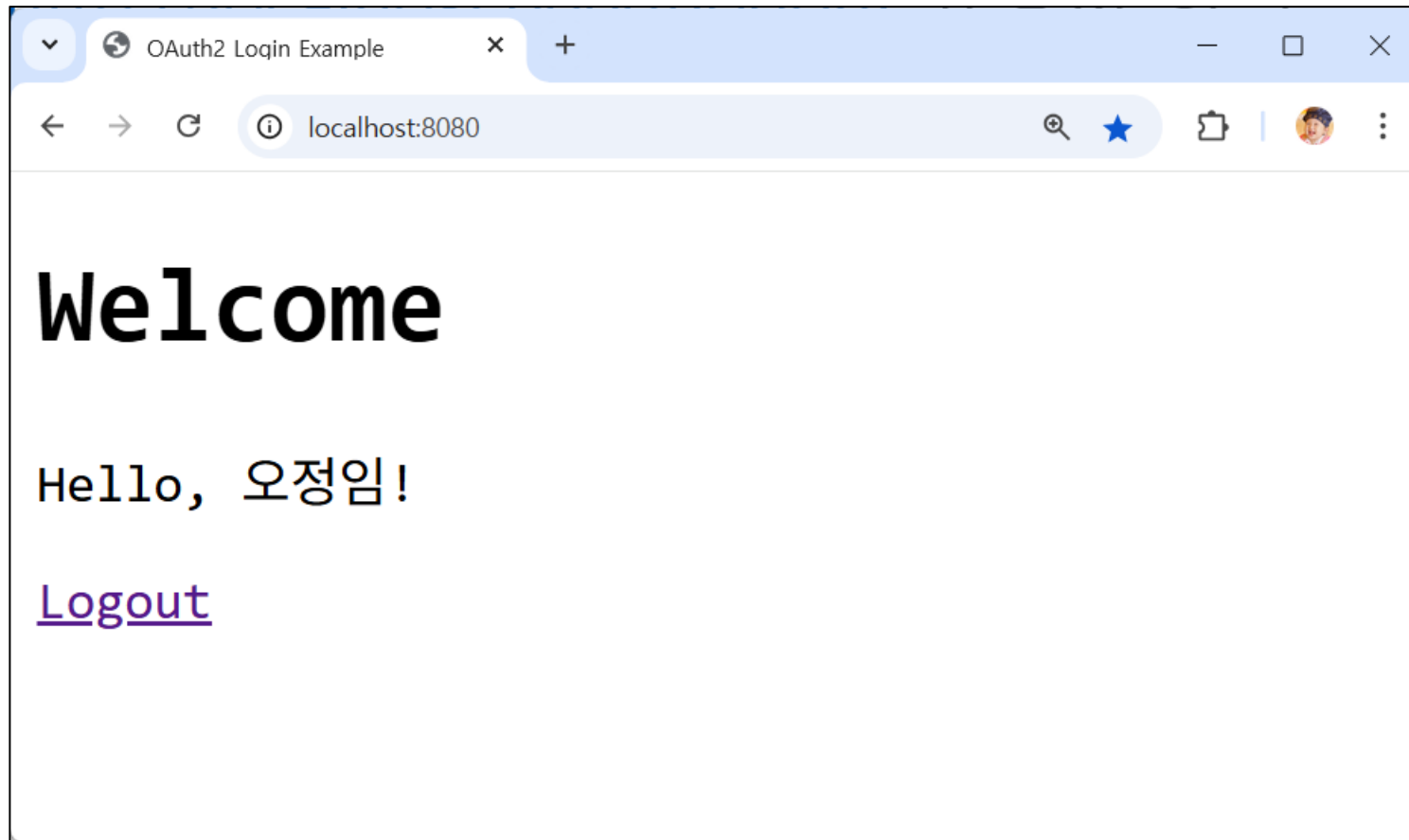
한국어 ▼

도움말

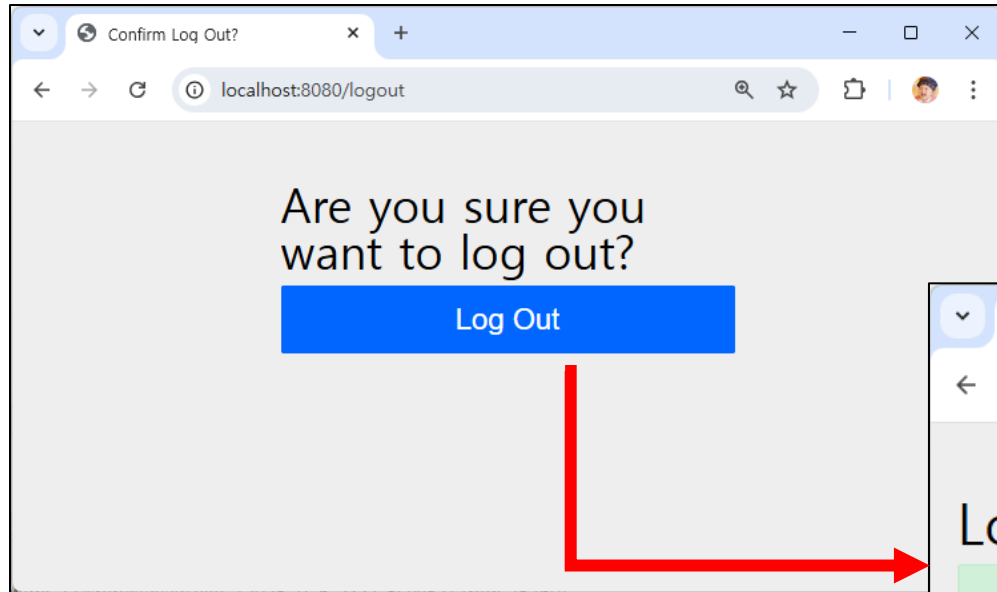
개인정보처리방침

약관

http://localhost:8080/으로 REDIRECT



http://localhost:8080/logout 요청 화면



OAuth2User

- Spring Security에서 OAuth 2.0 인증 후 사용자 정보를 다루기 위해 사용되는 인터페이스이다.
- 주요 기능
 - OAuth2User 인터페이스는 인증된 사용자의 정보를 제공한다.이를 통해 사용자의 속성, 권한, 고유 식별자 등을 얻을 수 있다.
- 사용예
 - OAuth2 인증 후 사용자 정보를 기반으로 추가적인 로직을 처리하는 데 사용된다. 예를 들어, 소셜 로그인을 통해 가져온 사용자 정보를 데이터베이스에 등록하거나, 해당 사용자의 권한을 설정하는 데 사용된다.

OAuth2User 메소드

- `getAttributes()`
 - 사용자의 속성 정보를 `Map<String, Object>` 형태로 반환하는 메소드이다. 예를 들어, 사용자의 이름, 이메일, 프로필 사진 URL 등의 정보가 포함될 수 있다.
- `getAuthorities()`
 - 사용자가 가지고 있는 권한을 반환하는 메소드이다. 이 권한 정보는 `GrantedAuthority`의 리스트 형태로 제공된다.
- `getName()`
 - 사용자의 고유 식별자를 반환하는 메소드이다. 이 값은 사용자 식별에 사용될 수 있으며, OAuth2 공급자에 따라 다르게 설정될 수 있다.

OAuth2 로그인시 사용자 정보 처리

- DefaultOAuth2UserService 상속
- public OAuth2User loadUser(OAuth2UserRequest)
OAuth2 인증 과정에서 제공자로부터 사용자 정보를 가져오기 위한 메소드
- OAuth2UserRequest
OAuth 2.0 인증 흐름에서 사용자의 액세스 토큰과 클라이언트 등록 정보를 캡슐화하여, OAuth 2.0 제공자와의 통신을 가능하게 하는 객체이다. 이를 통해 Spring Security는 사용자 정보를 요청하고 인증된 사용자 객체를 생성할 수 있다.

```

package com.edu.security;

import org.springframework.security.oauth2.client.userinfo.DefaultOAuth2UserService;
import org.springframework.security.oauth2.client.userinfo.OAuth2UserRequest;
import org.springframework.security.oauth2.core.OAuth2AuthenticationException;
import org.springframework.security.oauth2.core.user.OAuth2User;
import org.springframework.stereotype.Service;

@Service
public class ClubOAuthUserDetailsService extends DefaultOAuth2UserService {

    @Override
    public OAuth2User loadUser(OAuth2UserRequest userRequest) throws OAuth2AuthenticationException {
        // 디버깅 용 로그
        System.out.println("-----");
        System.out.println("userRequest : " + userRequest);

        // clientName 가져오기
        String clientName= userRequest.getClientRegistration().getClientName();
        System.out.println("ClientName : " + clientName);
        System.out.println(userRequest.getAdditionalParameters());

        // 부모 클래스의 loadUser 메소드 호출
        OAuth2User oAuth2User=super.loadUser(userRequest);

        // 디버깅 용 로그
        System.out.println("=====");
        oAuth2User.getAttributes().forEach((k, v) -> {
            System.out.println(k + ": " + v);
        });

        // 필수적인 사용자 정보 추가 로직이나 커스터마이징을 여기에 추가할 수 있음
        return oAuth2User;
    }
}

```

Google 로그인 후 Consol

```
-----  
userRequest : org.springframework.security.oauth2.client.userinfo.OAuth2UserRequest@1  
ClientName : Google  
{id_token=eyJhbGciOiJSUzI1NiIsImtpZCI6IjI1MTU4Nzk1MDg0NGE2NTZiZTM1NjNkOGM1YmQ2Zjg
```

```
=====
```

```
sub: 102571910302247849738
```

```
name: 오정임
```

```
given_name: 정임
```

```
family_name: 오
```

```
picture: https://lh3.googleusercontent.com/a/ACg8ocJ1KyiU9kfKiSX4VGglnp1yo0LqzHQtoQsR
```

```
email: ojiksb8022@gmail.com
```

```
email_verified: true
```

OAuth 회원 가입 처리

```
package com.edu.entity;

@MappedSuperclass
@EntityListeners(AuditingEntityListener.class)
@Getter
public abstract class BaseEntity {

    @CreatedDate
    @Column(name = "regdate", updatable = false)
    private LocalDateTime regDate;

    @LastModifiedDate
    @Column(name = "moddate")
    private LocalDateTime modDate;
}
```

```
package com.edu.entity;

public enum ClubMemberRole {
    USER, MANAGER, ADMIN
}
```

```
package com.edu.entity;

@Entity
@Getter
@ToString
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class ClubMember extends BaseEntity {

    @Id
    private String email;
    private String password;
    private String name;
    private boolean fromSocial;

    @ElementCollection(fetch = FetchType.LAZY)
    @Builder.Default
    private Set<ClubMemberRole> roleSet = new HashSet<>( );

    public void addMemberRole(ClubMemberRole clubMemberRole) {
        roleSet.add(clubMemberRole);
    }
}
```

```
package com.edu.repository;
```

```
public interface ClubMemberRepository extends JpaRepository<ClubMember, String> {
```

```
    @EntityGraph(attributePaths = {"roleSet"}, type = EntityGraph.EntityGraphType.LOAD)
```

```
    @Query("select m from ClubMember m where m.fromSocial = :social and m.email = :email")
```

```
    Optional<ClubMember> findByEmail(@Param("email") String email, @Param("social") boolean social);
```

```
}
```

```

@Service
public class Club0AuthUserDetailsService extends Default0Auth2UserService {

    @Autowired
    private ClubMemberRepository repository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Override
    public 0Auth2User loadUser(0Auth2UserRequest userRequest) throws 0Auth2AuthenticationException {
        ~ 생략 ~
        String email= oAuth2User.getAttribute("email");
        if (email != null) {
            saveSocialMember(email);
        }

        return oAuth2User;
    }

    private ClubMember saveSocialMember(String email) {
        Optional<ClubMember> result = repository.findByEmail(email, true);
        if (result.isPresent()) {
            return result.get();
        }

        ClubMember clubMember= ClubMember.builder()
            .email(email)
            .password(passwordEncoder.encode("1111"))
            .fromSocial(true)
            .build();

        clubMember.addMemberRole(ClubMemberRole.USER);
        repository.save(clubMember);
        return clubMember;
    }
}

```


edu_OAuth2

Spring Security 6: 폼 로그인 + Google OAuth2 통합 예제

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
  </dependency>
</dependencies>
```

application.properties

```
spring.application.name=edu_0Auth2
```

```
#H2
```

```
spring.datasource.url=jdbc:h2:tcp://localhost/~ /test
```

```
spring.datasource.driver-class-name=org.h2.Driver
```

```
spring.datasource.username=sa
```

```
spring.datasource.password=
```

```
spring.jpa.hibernate.ddl-auto=update
```

```
spring.jpa.show-sql=true
```

```
spring.jpa.properties.hibernate.format_sql=true
```

```
# Google
```

```
spring.security.oauth2.client.registration.google.client-id= ID YOUR_CLIENT_ID
```

```
spring.security.oauth2.client.registration.google.client-secret= YOUR_CLIENT_SECRET
```

```
spring.security.oauth2.client.registration.google.scope=profile,email
```

```
spring.security.oauth2.client.provider.google.user-name-attribute=name
```

```
# Google Redirection URI:
```

```
# http://localhost:8080/login/oauth2/code/google
```

```
package com.example.demo.config;
```

```
@Configuration
```

```
public class SecurityConfig {
```

```
    /** 데모용 인메모리 유저(실무에선 제거하고 DB UserDetailsService 사용) */
```

```
    @Bean
```

```
    public UserDetailsService users(PasswordEncoder pe) {
```

```
        return new InMemoryUserDetailsManager(  
            User.withUsername( "user" )  
                .password( pe.encode( "password" ) )  
                .roles( "USER" )  
                .build( )
```

```
        );
```

```
    }
```

```
    @Bean
```

```
    public PasswordEncoder passwordEncoder( ) {
```

```
        return new BCryptPasswordEncoder( );
```

```
    }
```

@Bean

```
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
    http
```

```
        .authorizeHttpRequests(auth -> auth  
            .requestMatchers("/", "/login",  
                "/oauth2/authorization/**", // OAuth2 시작  
                "/login/oauth2/code/**", // OAuth2 콜백  
                "/css/**", "/js/**", "/images/**", "/error").permitAll()  
            .anyRequest().authenticated()  
        )
```

```
// 폼 로그인과 OAuth2 로그인을 같은 /login 페이지에서 시작
```

```
        .formLogin(form -> form  
            .loginPage("/login") // GET /login → 내가 만든 login.html  
            .loginProcessingUrl("/login") // POST /login 처리  
            .defaultSuccessUrl("/", true).permitAll()  
        )
```

```
        .oauth2Login(oauth -> oauth  
            .loginPage("/login") // 같은 페이지에서 OAuth 버튼 노출  
            .defaultSuccessUrl("/", true)  
        )
```

```
        .logout(logout -> logout  
            .logoutUrl("/logout").logoutSuccessUrl("/")  
            .invalidateHttpSession(true).deleteCookies("JSESSIONID").permitAll()  
        )
```

```
        .csrf(Customizer.withDefaults());
```

```
    return http.build();
```

```
}
```

```
}
```

```
package com.example.demo.web;
```

```
@Controller
```

```
public class LoginController {
```

```
    @GetMapping("/login")
```

```
    public String login() {
```

```
        return "login";
```

```
        // templates/login.html
```

```
    }
```

```
    @GetMapping("/")
```

```
    public String home(Principal principal, Model model) {
```

```
        if (principal != null) {
```

```
            model.addAttribute("name", principal.getName());
```

```
        }
```

```
        return "home";
```

```
        // templates/home.html
```

```
    }
```

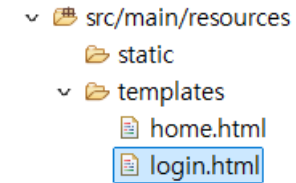
```
}
```

login.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Please sign in</title>
  <meta charset="UTF-8"/>
</head>
<body>
  <h1>Please sign in</h1>

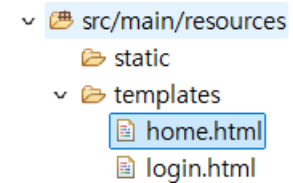
  <form th:action="@{/login}" method="post">
    <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}"/>
    <div>
      <label for="username">Username</label>
      <input id="username" name="username" type="text" autocomplete="username" />
    </div>
    <div>
      <label for="password">Password</label>
      <input id="password" name="password" type="password" autocomplete="current-password" />
    </div>
    <button type="submit">Sign in</button>
  </form>

  <h2>Login with OAuth 2.0</h2>
  <p><a href="/oauth2/authorization/google">Google</a></p>
</body>
</html>
```



home.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head><meta charset="UTF-8"/><title>Home</title></head>
<body>
  <h1>Home</h1>
  <div th:if="${name}">
    <p>Hello, <span th:text="${name}">User</span>!</p>
    <form action="/logout" method="post">
      <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}"/>
      <button type="submit">Logout</button>
    </form>
  </div>
  <div th:if="${name == null}">
    <a href="/login">Login</a>
  </div>
</body>
</html>
```



http://localhost:8080/login 접속



Please sign in

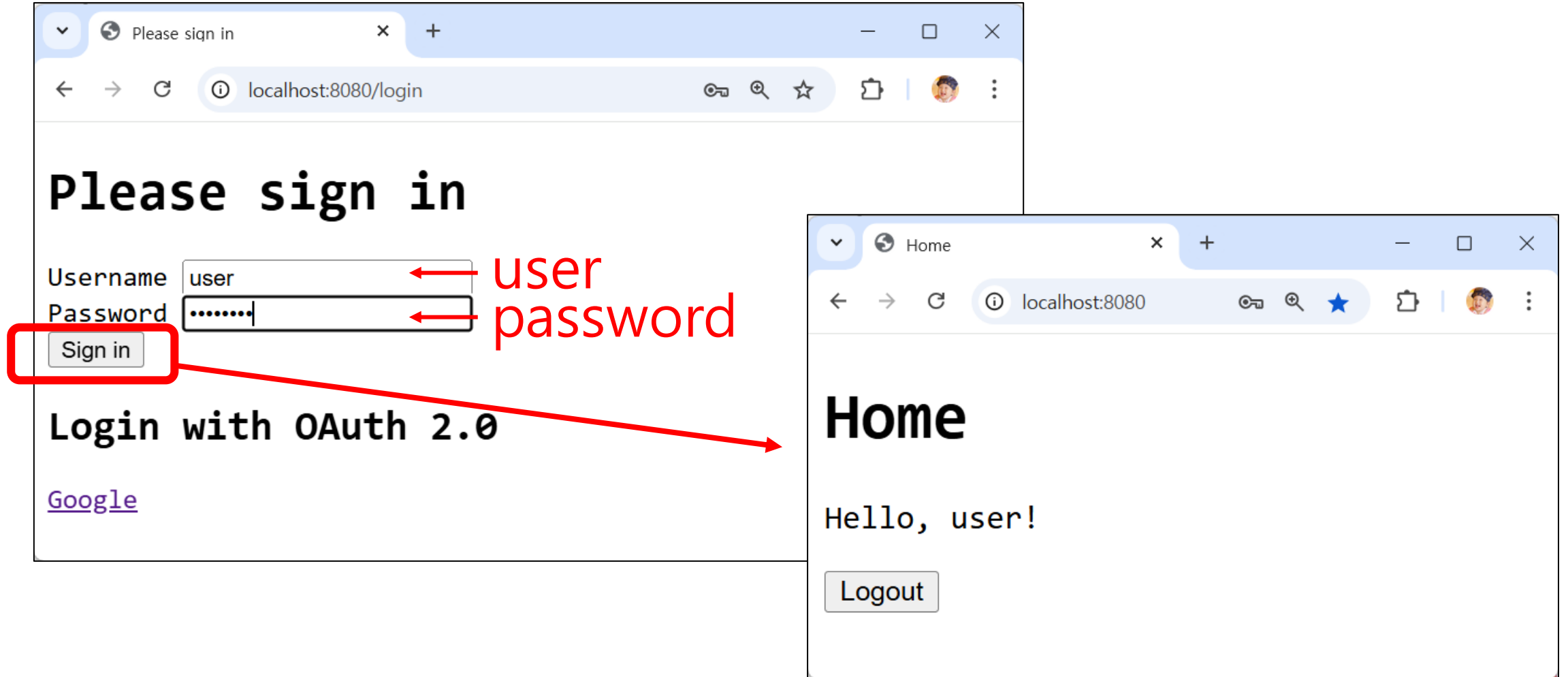
Username

Password

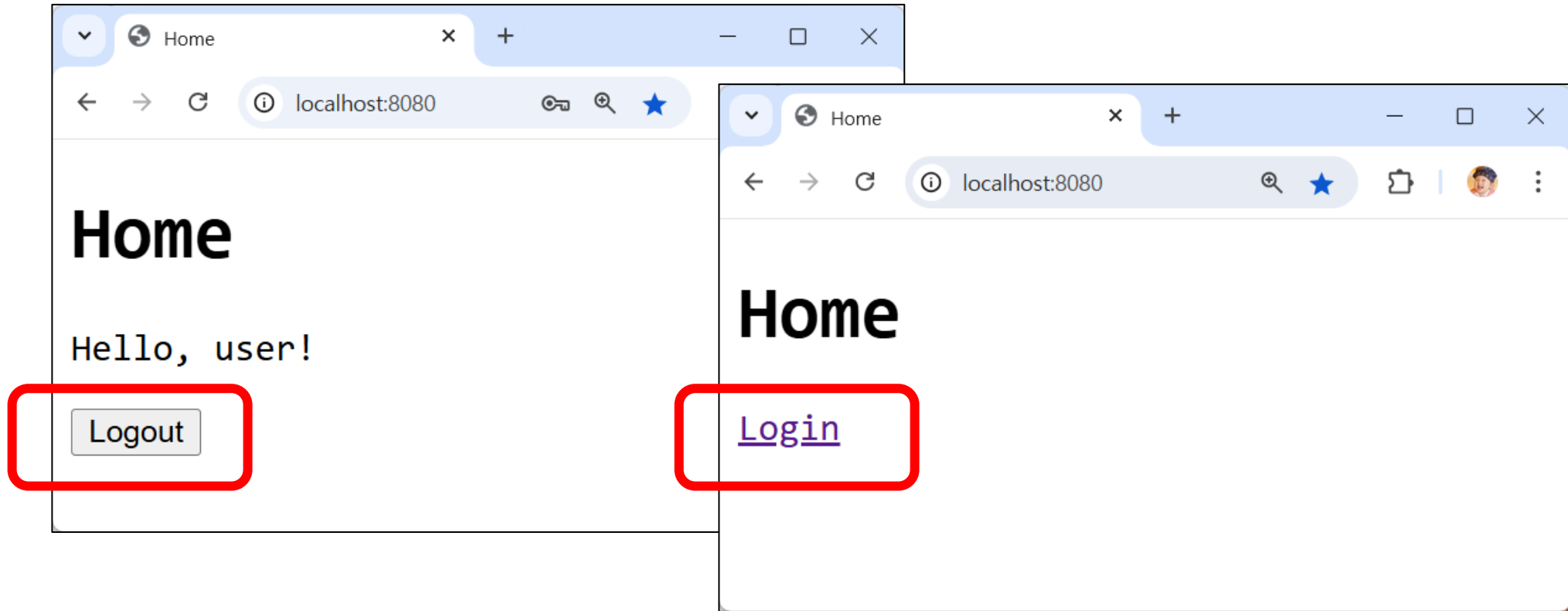
Login with OAuth 2.0

[Google](#)

인메모리 데모 유저 계정으로 테스트



로그아웃 후 다시 로그인



Google 로그인

The diagram illustrates the Google OAuth 2.0 login process through three sequential browser window states:

- Initial Login Page:** The browser window shows the URL `localhost:8080/login`. The page content includes the heading "Please sign in", input fields for "Username" and "Password", a "Sign in" button, and the text "Login with OAuth 2.0". A red box highlights the "Google" link, with a red arrow pointing to the next window.
- Google Account Selection Page:** The browser window shows the URL `accounts.google.com/o/oauth2/v2/auth/o...`. The page content includes the heading "계정 선택" (Account Selection), a user profile for "오정임" (Oh Jeong-im) with email "ojksb8022@gmail.com", and the text "springboot(으)로 이동" (Move to springboot). There is also a link for "다른 계정 사용" (Use another account).
- Home Page:** The browser window shows the URL `localhost:8080`. The page content includes the heading "Home", a greeting "Hello, 102571910302247849738!", and a "Logout" button.