보안 인증/인가 관련한 Security Lab (SpringBootLab03)

주요 구성 요소

단계	내용
[1 단계]	Spring Security 기본 - Spring Security 의존성 추가 및 기본 설정 - 사용자/비밀번호 인증 처리 - spring-boot-starter-security
[2 단계]	OAuth2 Client 연동 - Google OAuth2 로그인 연동 - 인증 흐름 (Authorization Code Flow) 이해 - Google API Console 에서 OAuth2 클라이언트 등록 - 사용자가 구글 로그인 승인 → 승인 코드 → Access Token 교환 - spring-boot-starter-oauth2-client
[3 단계]	OAuth2 Authorization Server - Spring Authorization Server 로 자체 인증 서버(Single Sign-On)구축 - 권한 부여 코드 플로우 및 토큰 발급 - 클라이언트 등록, 승인 코드 발급, 토큰 발급
[4 단계]	OAuth2 Resource Server - JWT 기반 보호된 API (/api/**) - Access Token 검증 및 리소스 접근 제어 -JWT Access Token 검증 → 사용자 정보 추출 - **spring-boot-starter-oauth2-resource-server**
[5 단계]	LDAP 연동 - Spring LDAP 의존성 및 설정 - LDAP 서버를 통한 사용자 인증 처리 - 보통 사내 계정/Active Directory 연동할 때 많이 씀 - spring-boot-starter-data-ldap
[6 단계]	Okta 연동 - Okta Developer 계정 연동 -Okta OAuth2 클라이언트로 로그인 처리 - 외부 IdP 기반으로 통합 로그인 구현

단계별 차이와 연결

단계	주요 개념	인증 주체
1	Spring Security 기본 로그인 처리	자체 DB(UserDetailsService)
2	외부 OAuth2 제공자 사용 (클라이언트)	Google / Facebook / Kakao
3	자체 OAuth2 인증 서버 구축	내부 사용자 및 애플리케이션
4	토큰 기반 API 보호 (자원 서버)	JWT / Access Token
5	LDAP 서버로 인증	LDAP 서버 (사내 사용자)
6	Okta (외부 IdP) 연동	Okta 인증 서버

디렉토리 구조

```
SpringBootLab03_Basic/
 ├─── src/main/java/com/
 |    ├─── test01/   → Lab01: 기본 Spring Security
 |    ├─── test02/   → Lab02: OAuth2 Client
 |    ├─── test03/   → Lab03: OAuth2 Authorization Server
 |    ├─── test04/   → Lab04: OAuth2 Resource Server
 |    ├─── test05/   → Lab05: Spring LDAP 연동
 |    └─── SpringBootLab03Application.java
 ├─── src/main/resources/
 |    ├─── templates/ (공통 템플릿)
 |    ├─── static/
 |    └─── application.yml
 └─── pom.xml
```

[실습 1] Spring Security 기본

```
# src/main/resources/application.yml
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/spring_lab?
    username: root
    password: root1234
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update  # 개발 시 update 로!
    show-sql: true
    properties:
      hibernate:
        format_sql: true
  thymeleaf:
    cache: false

server:
  port: 8080
```

기본 코드 작성단계

단계	설명	예시 코드 / 상세
1 의존성 추가	Spring Security 를 프로젝트에 포함	pom.xml 에 spring-boot-starter-security 의존성 추가
2 SecurityConfig 작성	SecurityFilterChain 으로 인증/인가 규칙 정의	HttpSecurity 로 어떤 경로를 인증 없이 허용할지, 인증 후 리디렉션 등을 지정
3.UserDetailsService	사용자 정보를 제공하는 서비스	- 테스트: InMemoryUserDetailsManager - 실제: DB 와 연결하는 사용자 조회 서비스

4. PasswordEncoder	비밀번호를 안전하게 암호화	BCryptPasswordEncoder 로 비밀번호를 암호화해 저장 & 검증
5. 컨트롤러	뷰로 이동할 수 있는 엔드포인트	/login, /register, /home 등의 URL 매핑과 폼 처리
6.뷰(HTML)	사용자에게 보여줄 페이지 (Thymeleaf 등)	index.html, login.html, home.html, register.html
7.테스트 실행	Spring Boot 로 애플리케이션 실행	mvn spring-boot:run 또는 IDE 에서 SpringBootLab03Application 실행

디렉토리 구조

```
com/test01
 ├─── config/                → Spring Security 설정 파일
 |    └─── SecurityConfig.java
 ├─── controller/            → 사용자 요청 처리 컨트롤러
 |    └─── MemberController.java
 ├─── entity/
 |    └─── Member.java
 ├─── repository/
 |    └─── MemberRepository.java
 ├─── service/
 |    └─── MemberService.java
 └─── SpringBootLab03Application.java  → 메인 클래스
```

1) Entity 생성 : com.test01.entity.Member.java

```
@Entity
@Getter @Setter @ToString
@NoArgsConstructor @AllArgsConstructor
public class Member {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false, unique = true)
    private String username;
    @Column(nullable = false)
    private String password;

    @Column(nullable = false)
    private String role = "ROLE_USER";

    private LocalDateTime createdAt = LocalDateTime.now();
}
```

2) Repository 생성: com.test01.repository.MemberRepository.java

```
public interface MemberRepository extends JpaRepository<Member, Long> {
    Optional<Member> findByUsername(String username);
}
```

3) SecurityConfig 생성: com.test01.config.SecurityConfig.java

```
@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf().disable()
```

```
                .authorizeHttpRequests(auth -> auth
                    .requestMatchers("/", "/register", "/css/**", "/js/**").permitAll()
                    .anyRequest().authenticated()
                )
                .formLogin(login -> login
                    .loginPage("/login")
                    .defaultSuccessUrl("/home", true)
                    .permitAll()
                )
                .logout(logout -> logout
                    .logoutSuccessUrl("/")
                    .permitAll()
                );

            return http.build();
        }

        @Bean
        public PasswordEncoder passwordEncoder() {
            return new BCryptPasswordEncoder();
        }
    }
```

4).서비스와 회원가입/로그인

구현 :com.test01.service.CustomUserDetailsService.java

```
@Service
@RequiredArgsConstructor
public class CustomUserDetailsService implements UserDetailsService {
    private final MemberRepository memberRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        Member member = memberRepository.findByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException("사용자를 찾을 수
없습니다."));

        return User.builder()
                .username(member.getUsername())
                .password(member.getPassword())
                .roles(member.getRole().replace("ROLE_", ""))
                .build();
    }
}
```

4) 회원가입 서비스 Service: MemberService

```
    @Service
    @RequiredArgsConstructor
    public class MemberService {

        private final MemberRepository memberRepository;
        private final PasswordEncoder passwordEncoder;
```

```
    public Member register(Member member)
member.setPassword(passwordEncoder.encode(member.getPassword()));
        return memberRepository.save(member);
    }
}
```

5) Controller & Thymeleaf 뷰 페이지

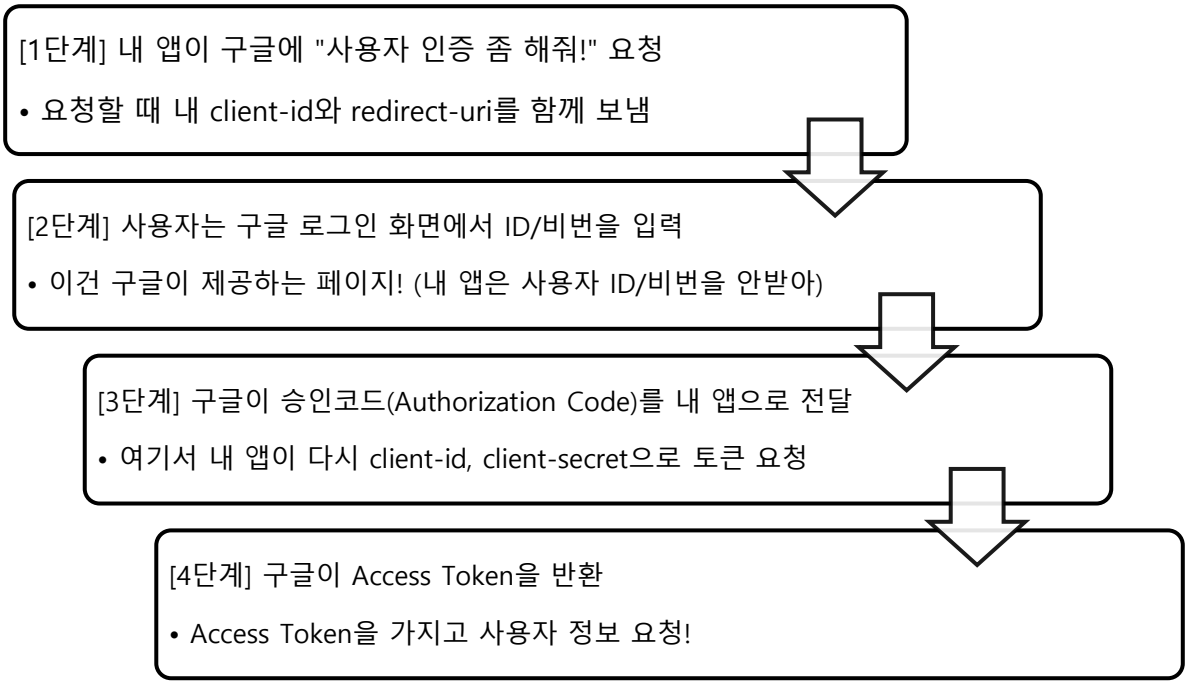
기능	HTTP 메소드	메소드명: 리턴타입	뷰 페이지	설명
메인 페이지	GET	index(): String	index.html	메인 페이지를 반환한다.
회원가입 폼	GET	registerForm(): String	register.html	회원가입 폼을 보여준다.
회원가입 처리	POST	register(Member member): String	(리다이렉트)	회원가입 처리 후 /login 으로 리다이렉트한다.
로그인 폼	GET	loginForm(): String	login.html	로그인 폼을 보여준다.
홈 페이지	GET	home(): String	home.html	인증된 사용자만 접근 가능한 홈 페이지를 보여준다. (home.html)
전체 회원 조회	GET	getAllMembers(): List<Member>	(JSON or 별도 뷰)	전체 회원 목록을 JSON 등으로 반환한다.

6) 테스트하기

1. application.yml 에 MySQL 정보 확인
2. DB 와 연결되는지 확인 (콘솔에서 Hibernate DDL 로그 확인!)
3. localhost:8080 접속 → 회원가입 → 로그인 → /home 확인

7) Thymeleaf 뷰 페이지

[실습 2] OAuth2 Client 연동 _구글 로그인 과정을 확인해 보자.

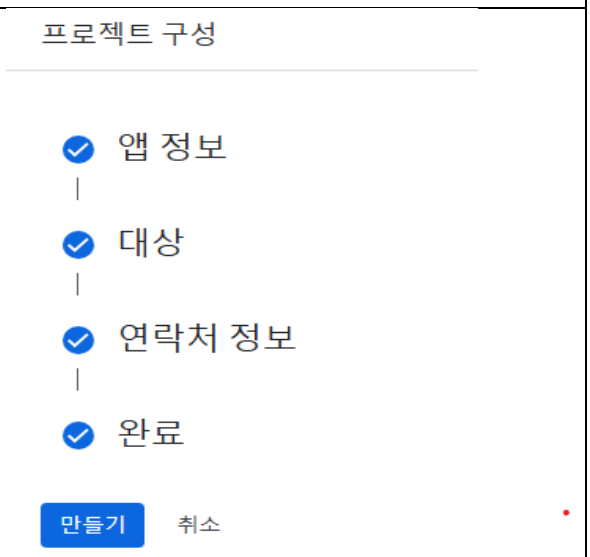


사용자 ID/비번은 **구글 로그인 페이지**에서만 입력되고,
내 앱에서는 **클라이언트 정보**(client-id, secret, redirect-uri)를 미리 설정

- client-id: 내 앱의 식별자 (구글 API Console 에서 발급)
- client-secret: 내 앱의 비밀 키

Application.yml

```yaml
spring:
  security:
    oauth2:
      client:
        registration:
          google:
            client-id: {발급받은-구글-클라이언트-ID}
            client-secret: {발급받은-구글-클라이언트-시크릿}
            scope:
              - email
              - profile
```

Google Cloud Console 접속	구글 계정 로그인
새 프로젝트 생성 또는 기존 프로젝트 선택	새로 만들거나 기존 프로젝트 선택
OAuth 동의 화면 구성 1 좌측 메뉴 → **API 및 서비스** → **OAuth 동의 화면** 2. 사용자 유형: 보통 **외부** 선택 3. 이름, 이메일 등 기본 정보 입력 4. OAuth 승인 범위(기본 email, profile 범위면 충분)	
OAuth2 클라이언트 ID 발급 1 좌측 메뉴 → **사용자 인증 정보** **2+ 사용자 인증 정보 만들기** 클릭 → **OAuth 클라이언트 ID** 선택 3. 애플리케이션 유형: 보통 **웹 애플리케이션** 선택	✅ **Client ID** ✅ **Client Secret** 이렇게 두 값이 자동으로 생성되면 application.xml 에 복붙!!

1) Entity 생성 : com.test01.entity.Member.java

```
@Entity
@Getter @Setter @ToString
@NoArgsConstructor @AllArgsConstructor
public class Member {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false, unique = true)
    private String username;
    @Column(nullable = false)
    private String password;

    @Column(nullable = false)
    private String role = "ROLE_USER";

    private LocalDateTime createdAt = LocalDateTime.now();
}
```

2). MemberRepository

```
public interface MemberRepository extends JpaRepository<Member,
Long> {
    Optional<Member> findByUsername(String username);
}
```

3). MemberService

```
@Service
public class MemberService {

     @Autowired
    private  MemberRepository memberRepository;


    public void saveOrUpdateMember(OAuth2User oauth2User) {
        String email = oauth2User.getAttribute("email");
        String name = oauth2User.getAttribute("name");
```

```
        Member member = memberRepository.findByUsername(email)
                .orElse(new Member());
        member.setUsername(email);
        member.setUsername(name);
        if (member.getPassword() == null) {
            member.setPassword("oauth2user");
        }

        memberRepository.save(member);
        System.out.println("저장된 사용자: " +
member.getUsername());
    }
}
```

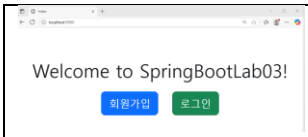
4). OAuth2LoginController

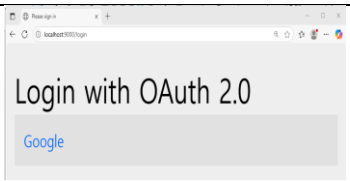
```
@Controller
public class OAuth2LoginController {

     @Autowired
    private  MemberService memberService;

    public OAuth2LoginController(MemberService memberService) {
        this.memberService = memberService;
    }

    @GetMapping("/home")
    public String home(@AuthenticationPrincipal OAuth2User
oauth2User) {
        memberService.saveOrUpdateMember(oauth2User);
        return "home";
    }
}
```

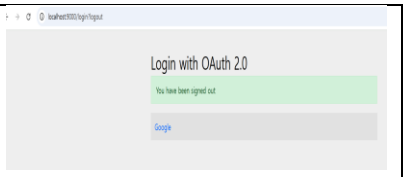
5). SecurityConfig

```
@Configuration
public class SecurityConfig {

        @Bean
         public SecurityFilterChain filterChain(HttpSecurity http) throws Exception
{
            http
                .authorizeHttpRequests(auth -> auth
                    .requestMatchers("/", "/index", "/login", "/register").permitAll()
                    .anyRequest().authenticated()
                )
                .oauth2Login(oauth2 -> oauth2
                    .defaultSuccessUrl("/home", true)
                );

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

실행결과







[실습 3] OAuth2 Authorization Server 인증

항목	com.test02	com.test03
주제/역할	OAuth2 Client (구글 로그인)	폼 기반 인증 (내부 사용자 DB)
SecurityConfig	.oauth2Login() 사용 (OAuth2 로그인 활성화)	.formLogin()만 사용 (내부 로그인)
컨트롤러	Google OAuth2 로그인 성공 후 사용자 처리	사용자 등록, 로그인, 홈 직접 처리
OAuth2 Client 설정	spring.security.oauth2.client.registration 설정 포함	없음
로그인 과정	Google 로 로그인 → 사용자 정보 DB 에 저장	사용자명/비밀번호를 자체 DB 로 직접 로그인

***보안 흐름(SecurityConfig)과 로그인 처리 컨트롤러만 다름**

디렉토리 구조

```
├─── src/main/java/com/
|    ├─── test03/   → Lab03: OAuth2 Authorization Server
|    |    ├─── config/AuthorizationServerConfig  인증서버
|    |    ├─── config/SecurityConfig    일반 리소스 보안
|    |    ├─── controller/UserController
|    |    ├─── SpringBootLab03Application.java
```

파일확인

폴더/파일	역할
config/AuthorizationServerConfig	OAuth2 서버, 클라이언트 등록, 토큰 엔드포인트 설정
config/SecurityConfig	Resource Server 보안 규칙, formLogin, UserDetailsService 설정
controller/UserController	Resource API: Bearer 토큰 필요
application.yml	포트 및 기본 Spring Security 사용자

1) 의존성 추가

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-oauth2-authorization-server</artifactId>
</dependency>
```

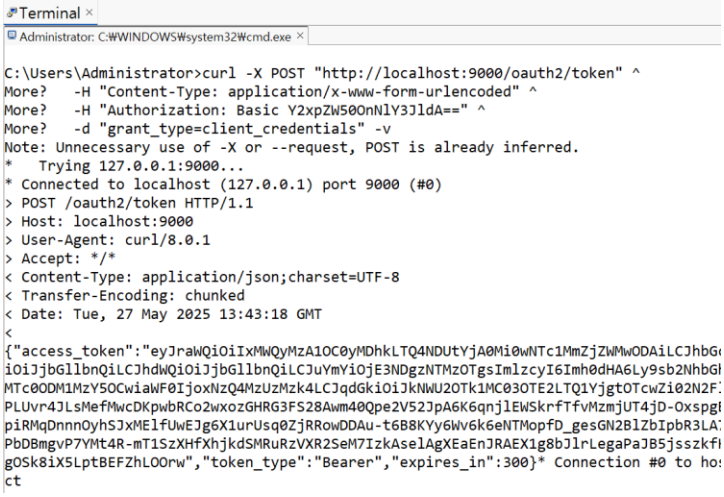
2). application.yml AuthorizationServerSettings 추가

```
server:
  port: 9000

spring:
  security:
    user:
      name: user
            password: password
```

3). AuthorizationServerConfig 구현 /4). SecurityConfig.java

5) OAuth2 승인화면(consent) Spring Security 가 자체 제공

```
curl -X POST "http://localhost:9000/oauth2/token" ^
  -H "Content-Type: application/x-www-form-urlencoded" ^
  -H "Authorization: Basic Y2xpZW50OnNlY3JldA==" ^
  -d "grant_type=client_credentials" -v
```


Terminal
Administrator: C:₩WINDOWS₩system32₩cmd.exe
C:\Users\Administrator>curl -X POST "http://localhost:9000/oauth2/token" ^
More?   -H "Content-Type: application/x-www-form-urlencoded" ^
More?   -H "Authorization: Basic Y2xpZW50OnNlY3JldA==" ^
More?   -d "grant_type=client_credentials" -v
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 127.0.0.1:9000...
* Connected to localhost (127.0.0.1) port 9000 (#0)
> POST /oauth2/token HTTP/1.1
> Host: localhost:9000
> User-Agent: curl/8.0.1
> Accept: */*
< Content-Type: application/json;charset=UTF-8
< Transfer-Encoding: chunked
< Date: Tue, 27 May 2025 13:43:18 GMT
<
{"access_token":"eyJraWQiOiIxMWQyMzA1OC0yMDhkLTQ4NDUtYjA0Mi0wNTc1MmZjZWMwODAiLCJhbG
iOiJjbGllbnQiLCJhdWQiOiJjbGllbnQiLCJuYmYiOjE3NDgzNTMzOTgsImlzcyI6Imh0dHA6Ly9sb2NhbG
MTc0ODM1MzY5OCwiaWF0IjoxNzQ4MzUzMzk4LCJqdGkiOiJkNWU2OTk1MC03OTE2LTQ1YjgtOTcwZi02N2F
PLUvr4JLsMefMwcDKpwbRCo2wxozGHRG3FS28Awm40Qpe2V52JpA6K6qnjlEWSkrfTfvMzmjUT4jD-Oxspg
piRMqDnnnOyhSJxMElfUwEJg6X1urUsq0ZjRRowDDAu-t6B8KYy6Wv6k6eNTMopfD_gesGN2BlZbIpbR3LA
PbDBmgvP7YMt4R-mT1SzXHfXhjkdSMRuRzVXR2SeM7IzkAselAgXEaEnJRAEX1g8bJlrLegaPaJB5jsszkf
gOSk8iX5LptBEFZhLOOrw","token_type":"Bearer","expires_in":300}* Connection #0 to hos
ct