YC Tech

웹 백엔드 실무 개발 프로젝트

연세대학교 미래교육원

YC (Yonsei X Codepresso) Tech Academy

3주차 recap

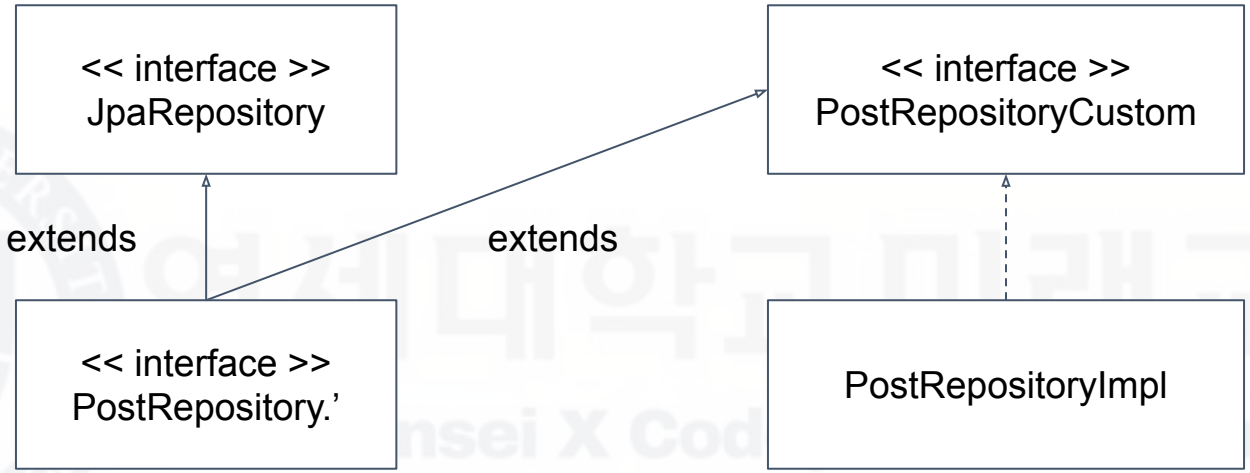
QueryDSL

복잡한 JPQL 작성 시 이슈

```
@Query("select p from Post p join fetch p.user u "
    + "where u in "
    + "(select t from Follow f inner join f.target t on f.source = :user) "
    + "or u = :user "
    + "order by p .createdAt desc" )
  List<Post> findAllAssociatedPostsByUser (@Param("user") User user, Pageable pageable);
```

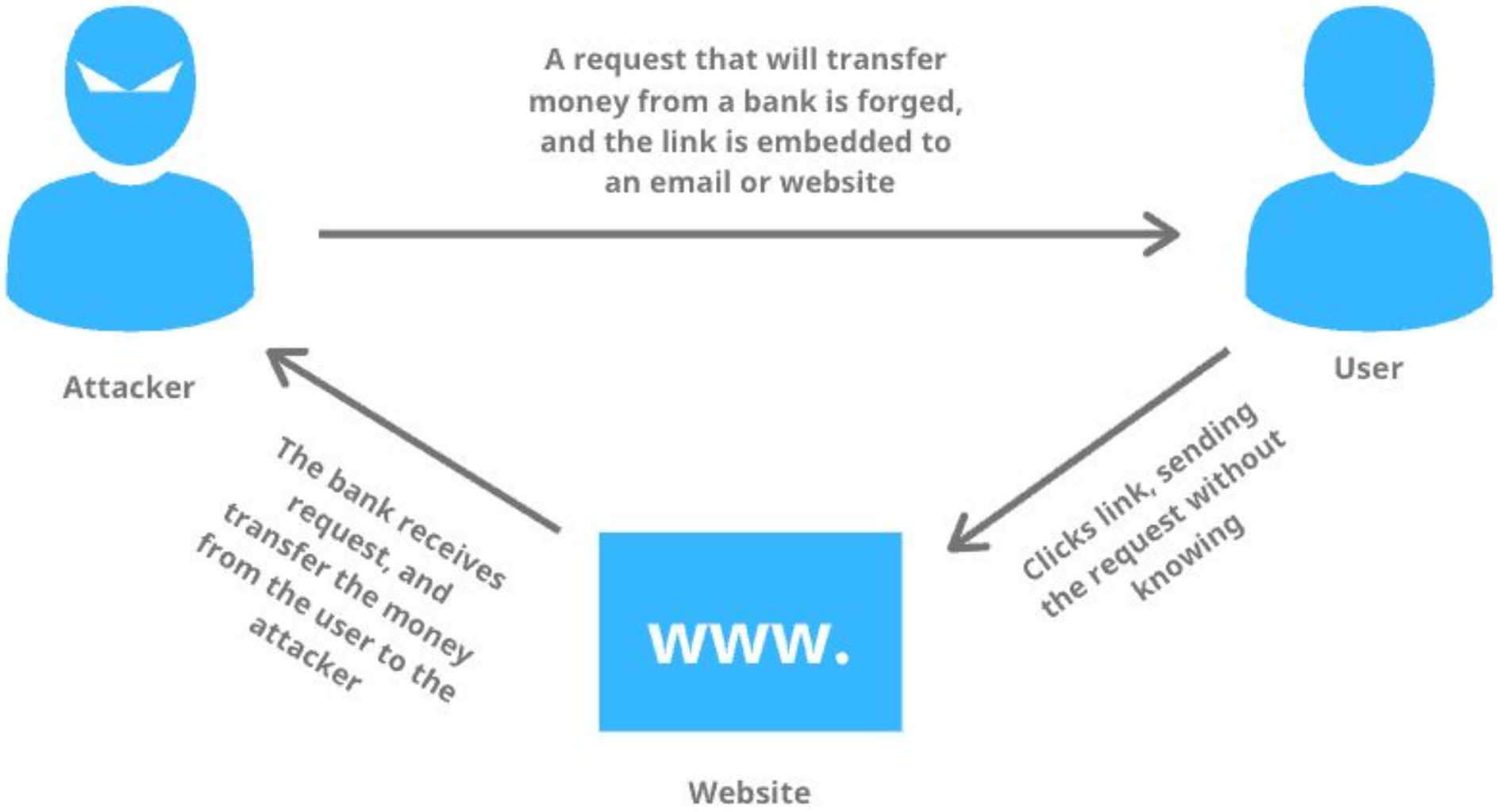
- 간단한 로직을 작성하는데 큰 문제는 없으나, 복잡한 로직의 경우 개행이 포함된 쿼리 문자열이 상당히 길어짐
- JPQL 문자열에 오타 혹은 문법적인 오류가 존재하는 경우, 정적 쿼리라면 어플리케이션 로딩 시점에 이를 발견할 수 있으나 그 외는 런타임 시점에서 에러가 발생
- 동적인 쿼리에 대한 요구사항

QueryDSL

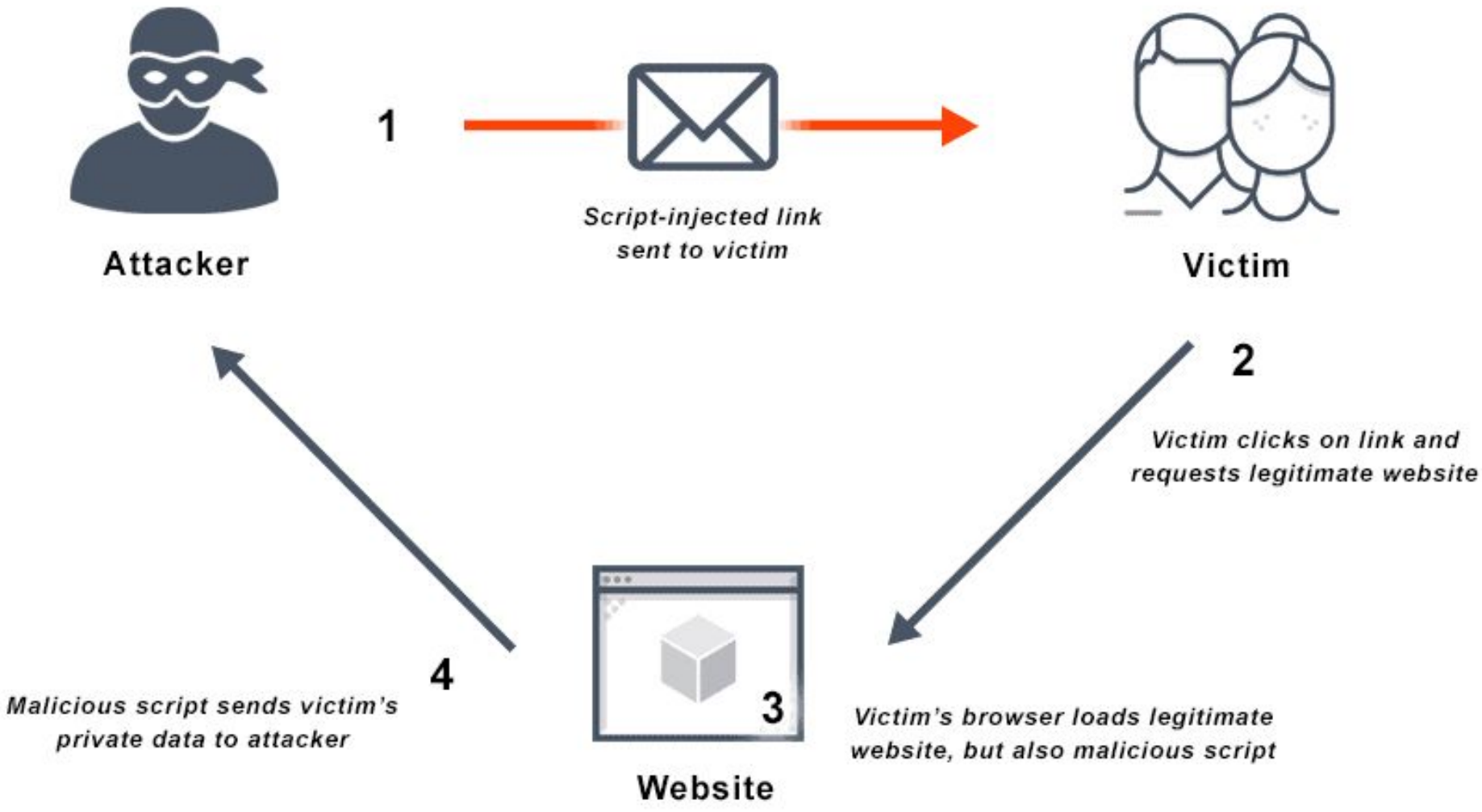


Security

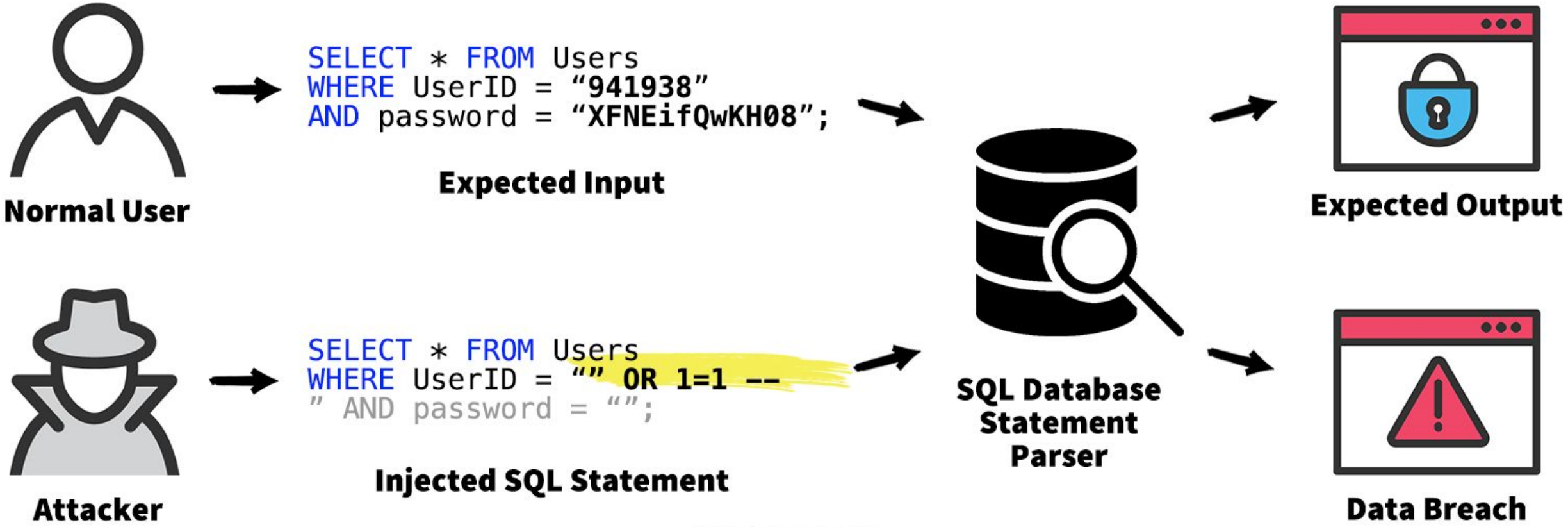
Cross-Site-Request-Forgery(CSRF)



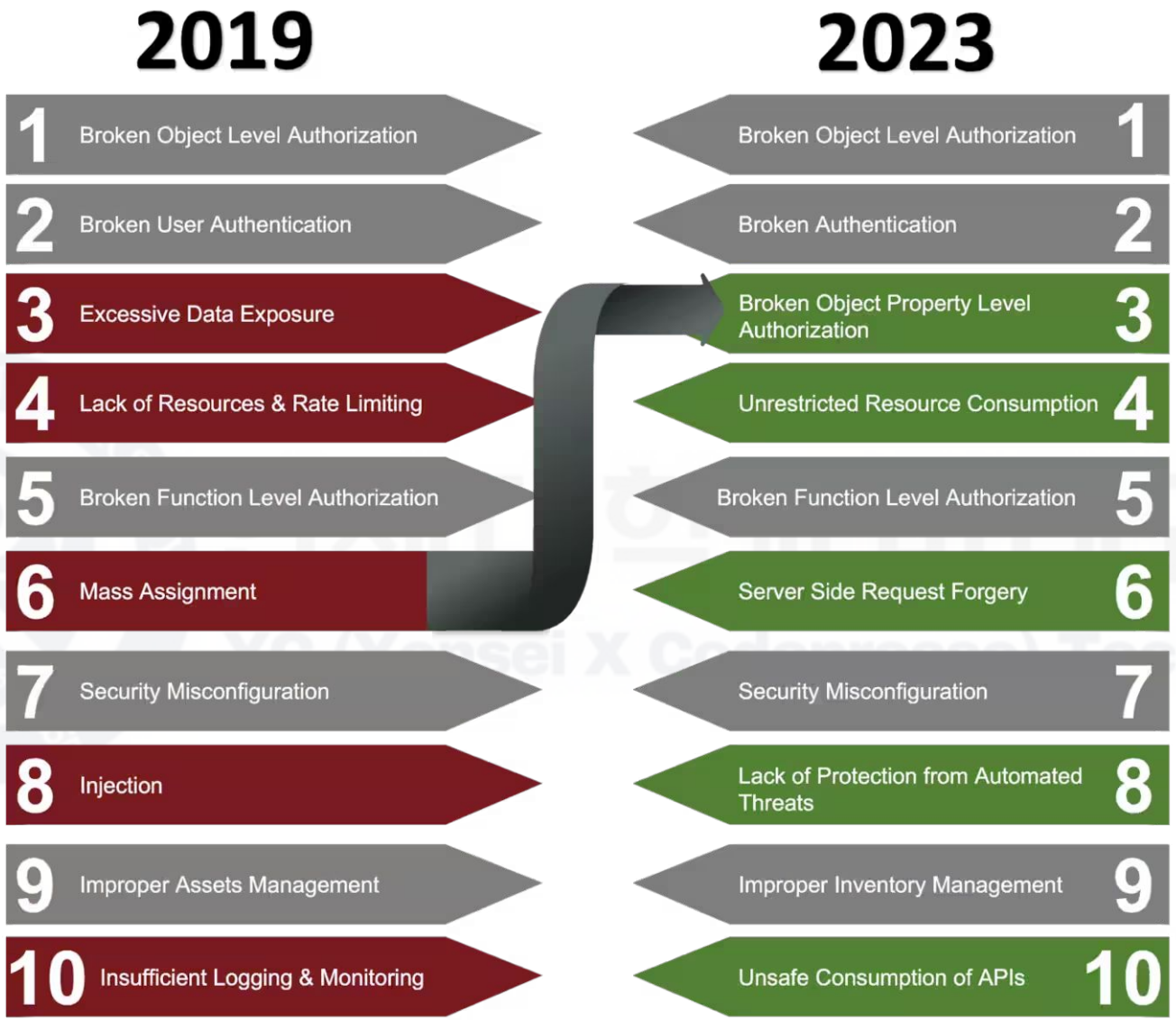
XSS



SQL injection

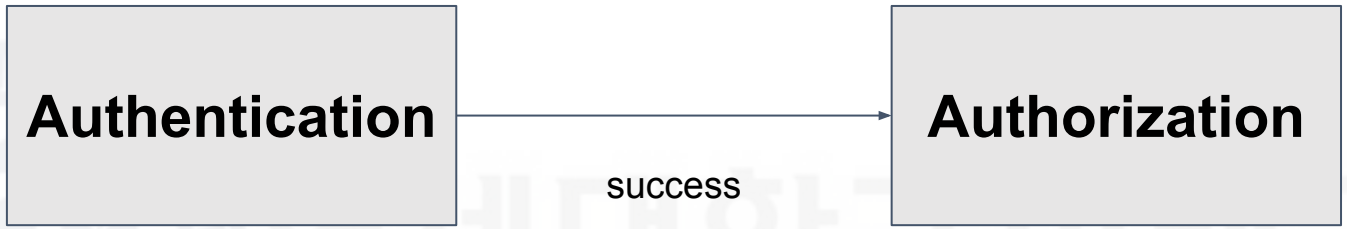


불충분한 인가



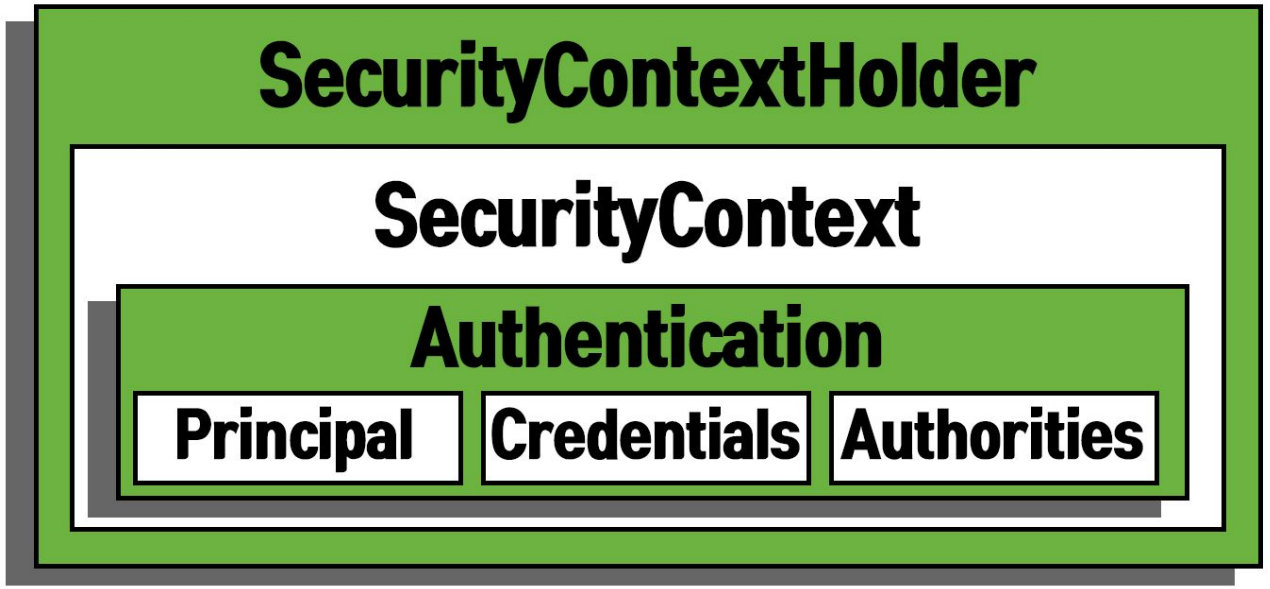
Spring security

Authorization, Authentication



- Authentication(인증) : 해당 사용자가 본인이 맞는지 확인하는 절차
- Authorization(인가) : 인증된 사용자가 요청한 자원에 접근 가능한지를 결정하는 절차

Authentication 관련 module

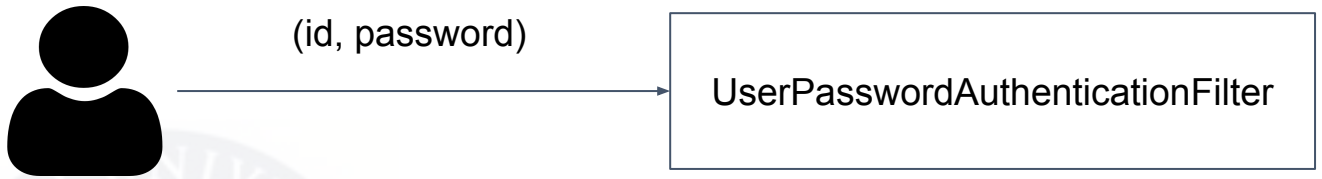


SecurityContext : Authentication 을 저장하는 저장소 역할
SecurityContextHolder : 기본적으로 thread local 로 security context 를 저장
Authentication : 접근 주체의 인증 정보와 권한을 다루는 인터페이스
- Principal : 접근 주체의 사용자 id 혹은 User 객체 저장
- Credentials : 비밀번호 정보
- Authorities : 인증된 접근 주체자의 권한 목록 저장

Authentication

```
public interface Authentication extends Principal, Serializable {
    // 현재 사용자의 권한 목록을 가져옴
    Collection<? extends GrantedAuthority> getAuthorities();

    // credentials(주로 비밀번호)을 가져옴
    Object getCredentials();

    Object getDetails();

    // Principal 객체를 가져옴.
    Object getPrincipal();

    // 인증 여부를 가져옴
    boolean isAuthenticated();

    // 인증 여부를 설정함
    void setAuthenticated(boolean isAuthenticated) throws
IllegalArgumentException;
}
```

인증 flow



1. 로그인 request

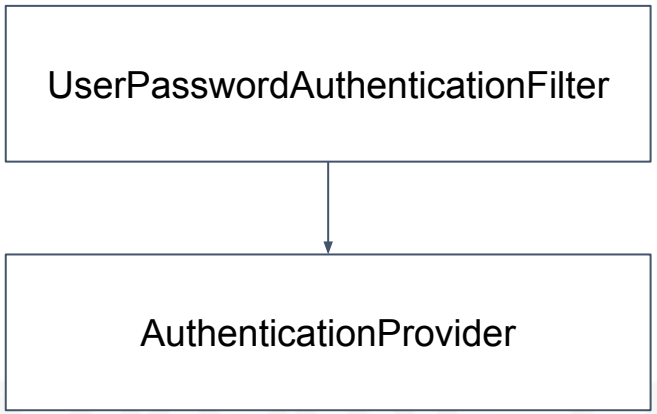
- Spring security filter chain 의 UserPasswordAuthenticationFilter 로 id 와 비밀번호 전달

인증 flow

2. Authentication 생성

- 요청받은 id 와 비밀번호를 이용하여 UsernamePasswordAuthenticationToken 생성
- UsernamePasswordAuthenticationToken
  - principal 은 Username 등의 신원을 의미
  - credentials 는 Password를 의미

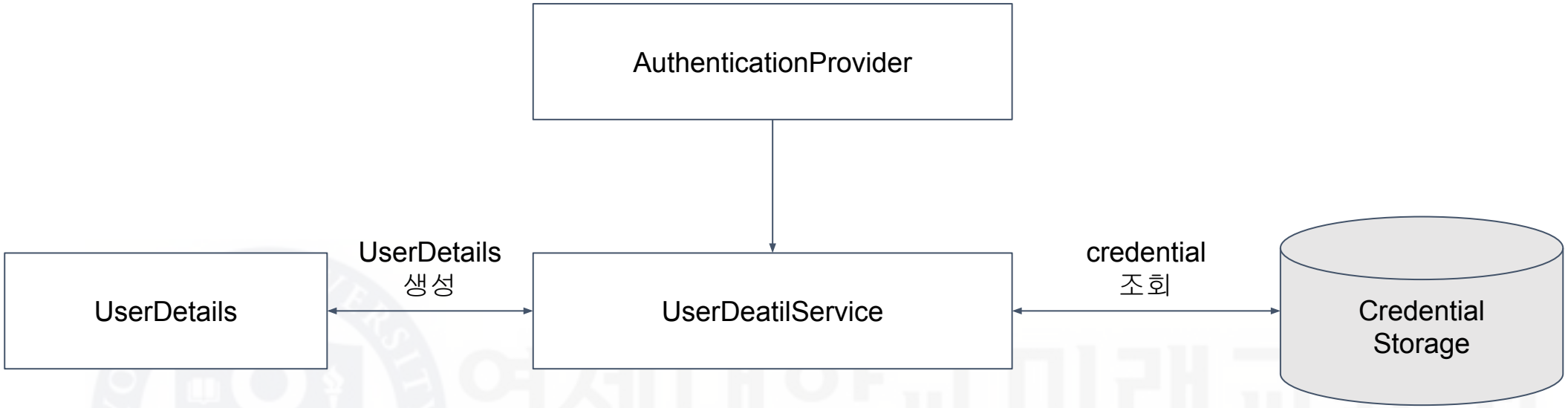
인증 flow



3. Authentication 전달

- AuthenticationManager로부터 인증 처리를 위임 받아 실질적인 인증 수행을 담당하는 컴포넌트
- Username/Password 기반의 인증 처리는 DaoAuthenticationProvider 가 담당하고 있으며, DaoAuthenticationProvider 는 UserDetailsService로부터 전달 받은 UserDetails를 이용해 인증을 처리

인증 flow



4. UserDetails 생성/조회

- 데이터베이스 등의 저장소에 저장된 사용자의 Username과 사용자의 자격을 증명해주는 크리덴셜(Credential)인 Password 그리고 사용자의 권한 정보를 포함하는 컴포넌트
- AuthenticationProvider 는 UserDetails 를 이용해 자격 증명을 수행

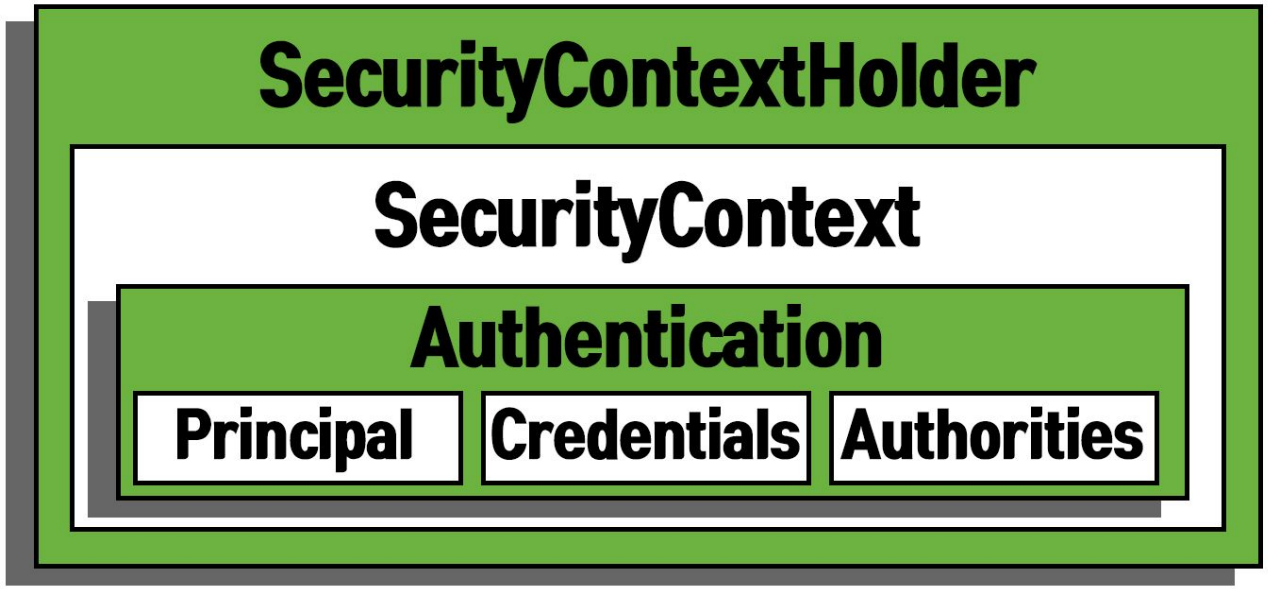
인증 flow

```
public interface UserDetails extends Serializable {

    Collection<? extends GrantedAuthority> getAuthorities(); // (1) 권한 정보
    String getPassword(); // (2) 패스워드
    String getUsername(); // (3) Username

    boolean isAccountNonExpired();  // (4) 사용자 계정의 만료 여부
    boolean isAccountNonLocked();   // (5) 사용자 계정의 Lock 여부
    boolean isCredentialsNonExpired(); // (6) Credentials(Password)의 만료 여부
    boolean isEnabled();            // (7) 사용자의 활성화 여부
}

public interface UserDetailsService {
    UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;
}
```

4. UserDetails 생성/조회

- 데이터베이스 등의 저장소에 저장된 사용자의 Username과 사용자의 자격을 증명해주는 크리덴셜(Credential)인 Password 그리고 사용자의 권한 정보를 포함하는 컴포넌트
- AuthenticationProvider 는 UserDetails 를 이용해 자격 증명을 수행
- UserDeatilsService를 통해 UserDetails 로드

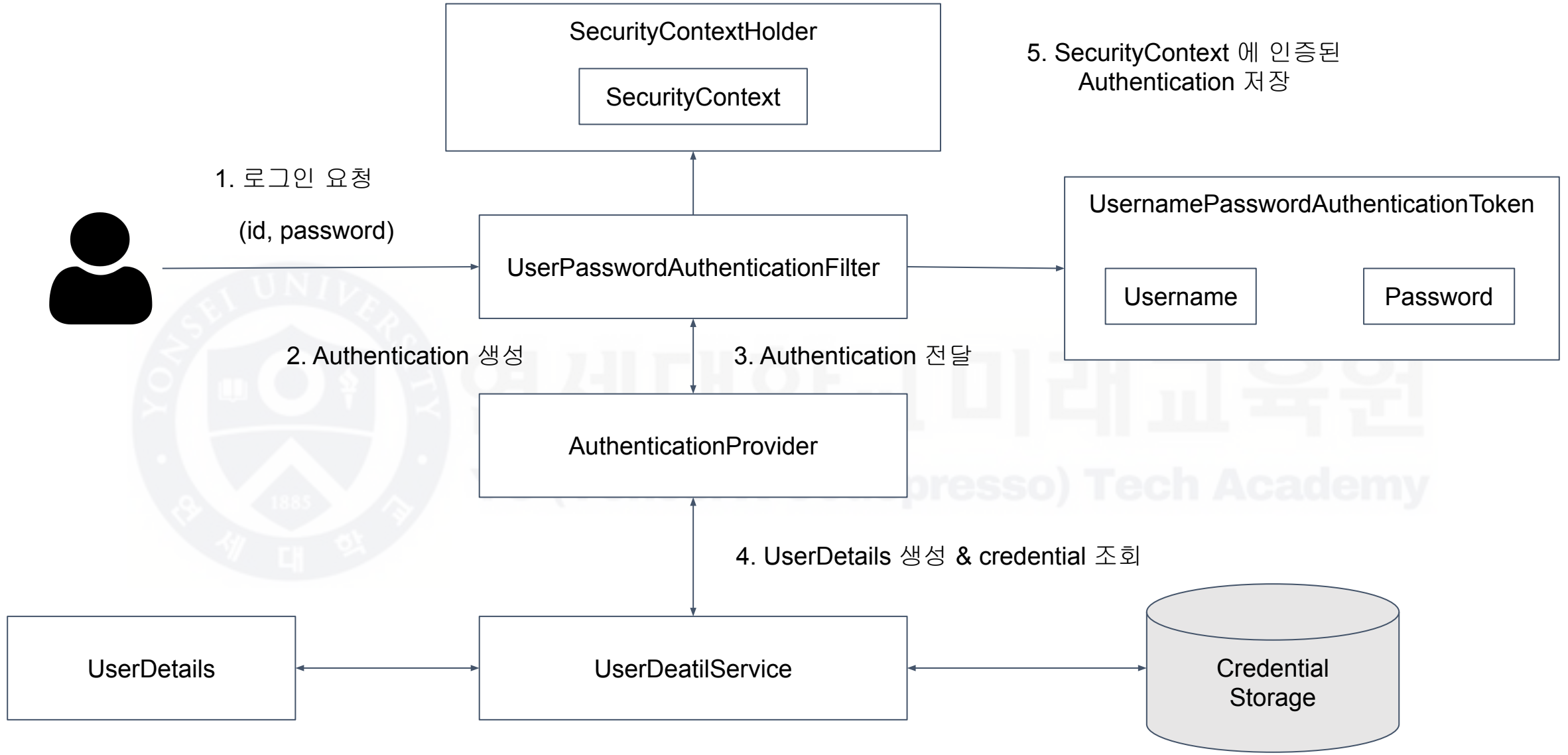
인증 flow



5. 인증된 Authentication 을 SecurityContext 에 저장

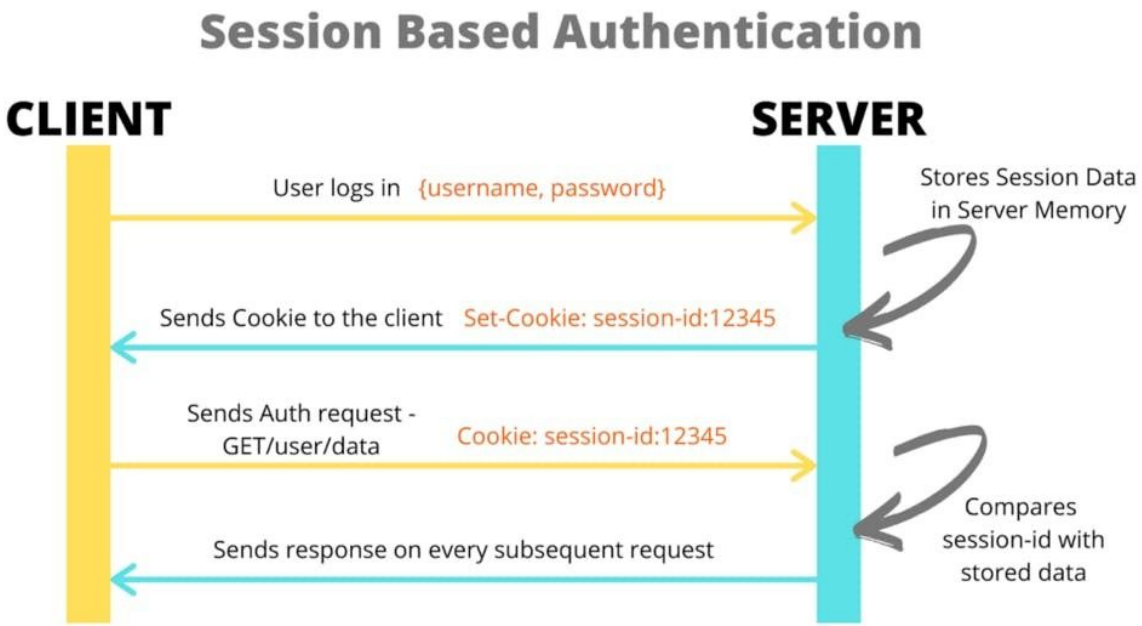
- SecurityContext 는 인증된 Authentication 객체를 저장하는 컴포넌트
- SecurityContextHolder 는 SecurityContext를 관리하는 역할

인증 flow



Session, Token

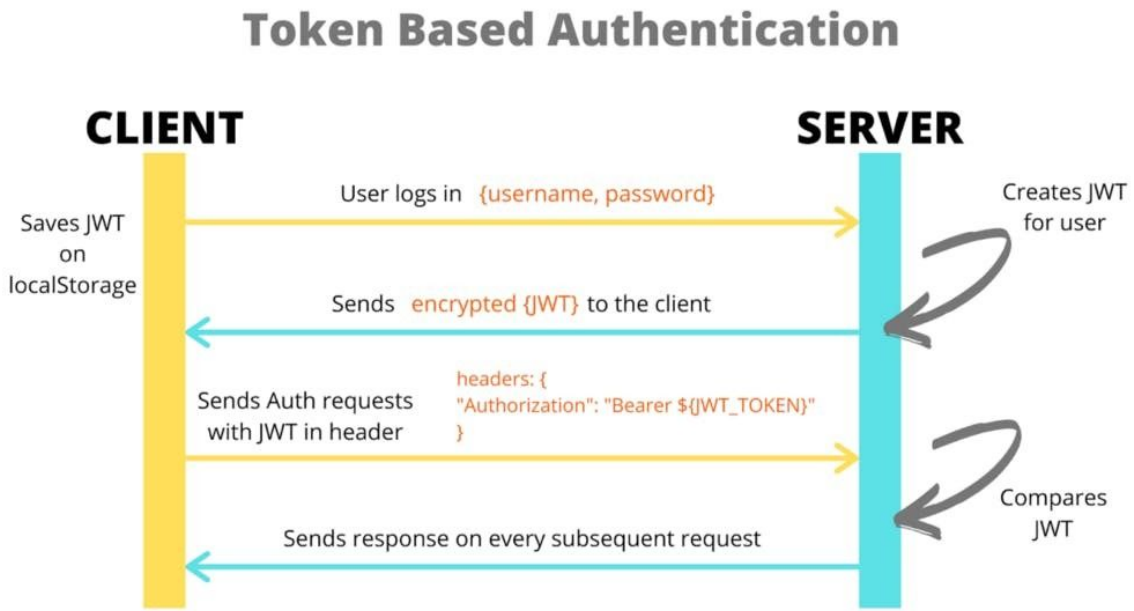
Session



Session 기반 인증 방법

- HTTP 프로토콜을 사용하는 인터넷 사용자가 어떤 웹사이트를 방문할 경우, 웹 사이트의 여러 페이지에 걸쳐 사용되는 사용자 정보를 저장하는 방법
- HTTP 프로토콜은 기본적으로 connectionless, stateless 하므로 클라이언트에서 로그인 상태를 유지할 방법이 필요함. 이러한 stateful 방법이 제공되지 않으면 로그인 정보를 유지할 수 없음
- session 정보를 메모리나 디스크, DB 등에 저장

Token



Token 기반 인증 방법

- Token 기반 인증은 인증 완료 후 클라이언트 측에 저장
- 생성된 토큰을 추적하지 않고 서버가 아는 것은 토큰의 유효성 여부만 관리

Session vs Token based Authentication

	Session	Token
장점	+ 서버에서 로그인 상태 확인이 간편 + 상대적으로 안전하다. 서버측에서 관리하기 때문에 클라이언트 변조에 영향받거나 데이터의 Stale (손상) 우려가 없다.	+ 클라이언트에 저장되기 때문에 서버의 메모리에 부담이 되지않으며 Scale 에 있어 대비책을 고려할 필요가 없다 + 멀티 디바이스 환경에 대한 부담이 없다.
단점	- 유저들의 세션에 대한 정보를 서버 메모리에 들고 있게 된다는 부담이 있다. - 서버 메모리에 세션 정보가 저장되기 때문에 Scale Out / Scale In 이 부담이 된다 - 멀티 디바이스 환경에서 로그인 시 구현이 복잡해진다.	- 상대적으로 Stale (손상) 의 위험이 크다. - 결국 구현을 하다보면 서버측에 token blacklist 를 관리하게 될 가능성이 있고 그렇게 되면 서버측 메모리의 소모가 발생하게 된다 - XSS 공격에 취약할 수 있어 가능한 민감한 정보는 포함시키지 않을 필요가 있다.

JWT

Json Web Token

Header.Payload.Signature

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 .eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ
.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```


- JSON 형식의 토큰에 대한 표준 규격인데요. 주로 사용자의 인증(authentication) 또는 인가(authorization) 정보를 서버와 클라이언트 간에 안전하게 주고 받기 위해서 사용
- https://jwt.io/ 에서 값 확인
- 서버에서는 토큰에 포함되어 있는 서명(signature) 정보를 통해서 위변조 여부를 빠르게 검증

Json Web Token

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

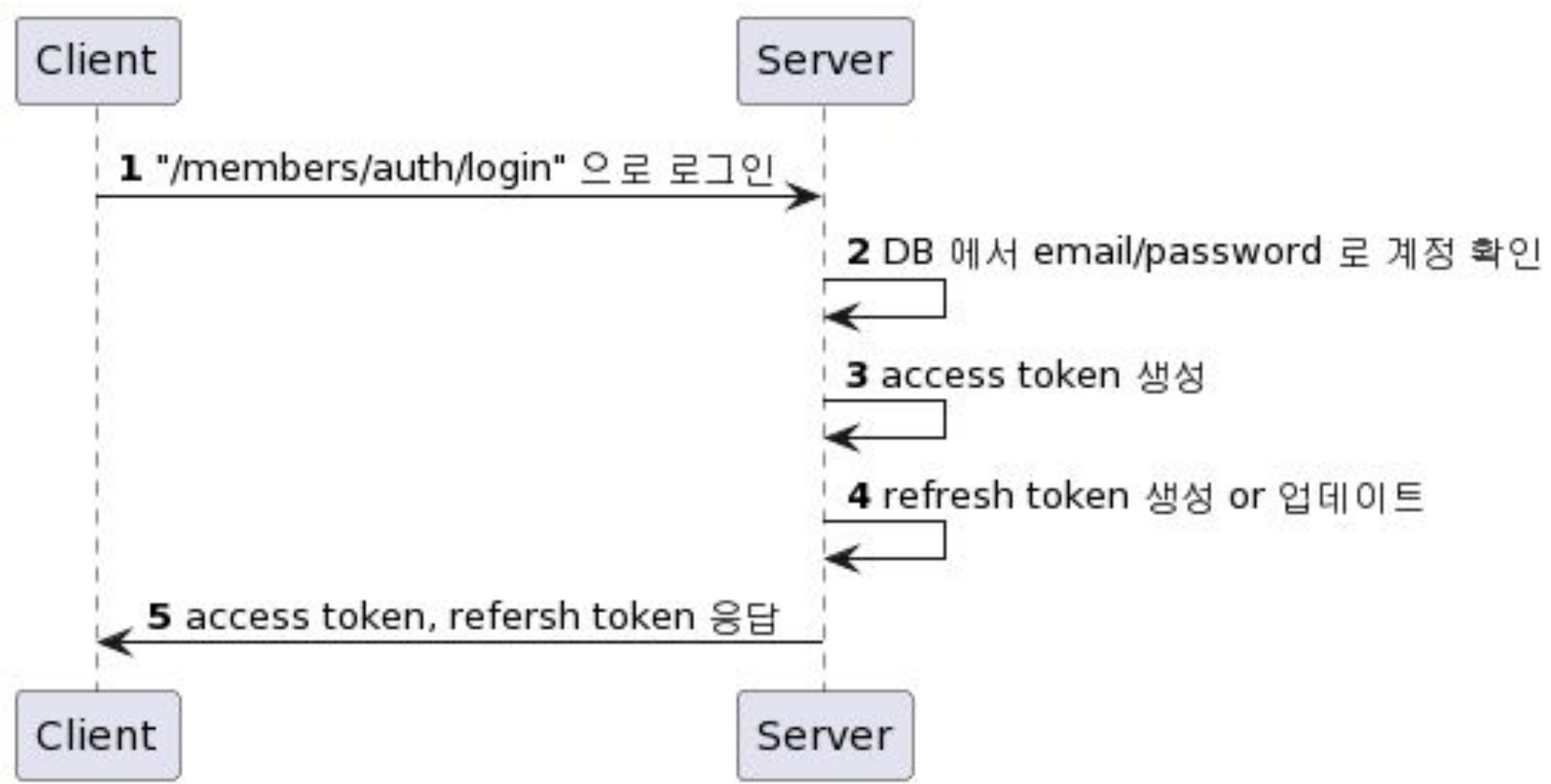
VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

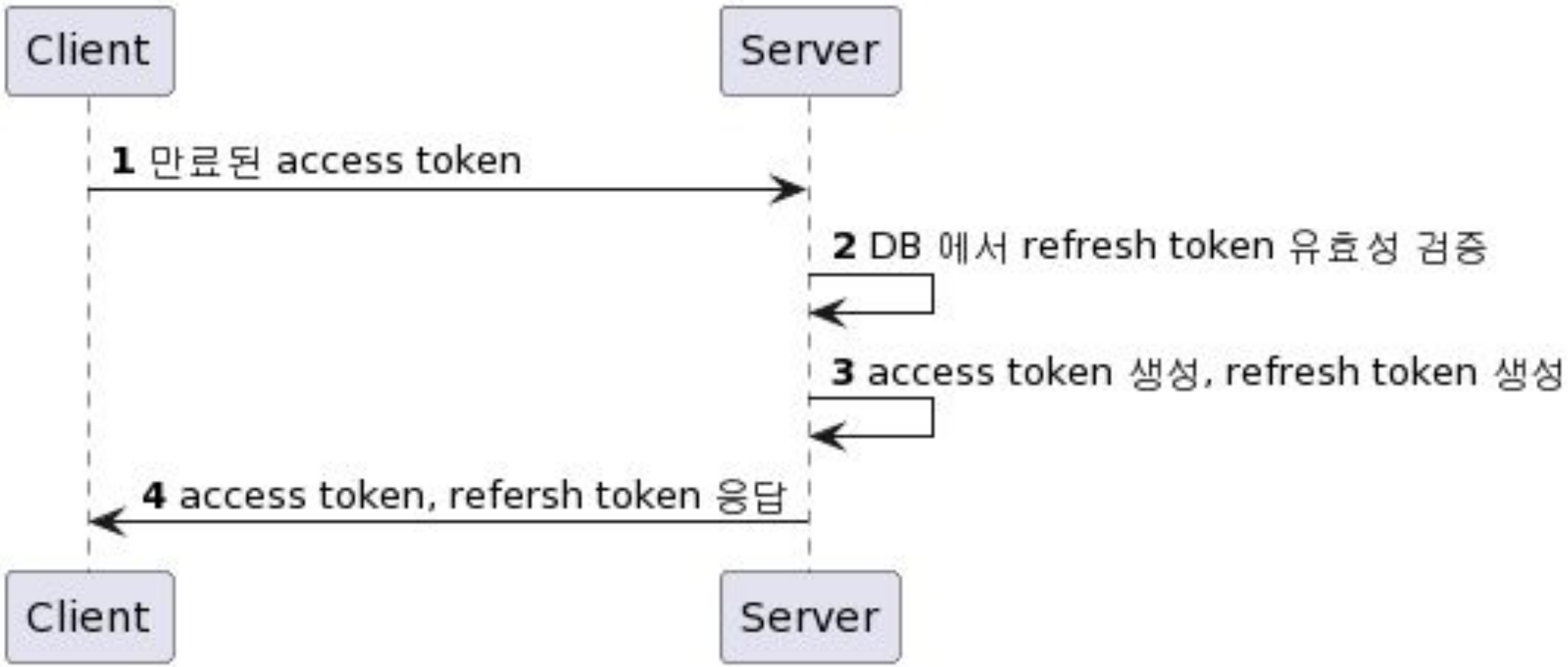
JWT 키 값

- sub 키: 인증 주체(subject)
- iss 키: 토큰 발급처
- typ 키: 토큰의 유형(type)
- alg 키: 서명 알고리즘(algorithm)
- iat 키: 발급 시각(issued at)
- exp 키: 말료 시작(expiration time)
- aud 키: 클라이언트(audience)

정상 로그인 Flow

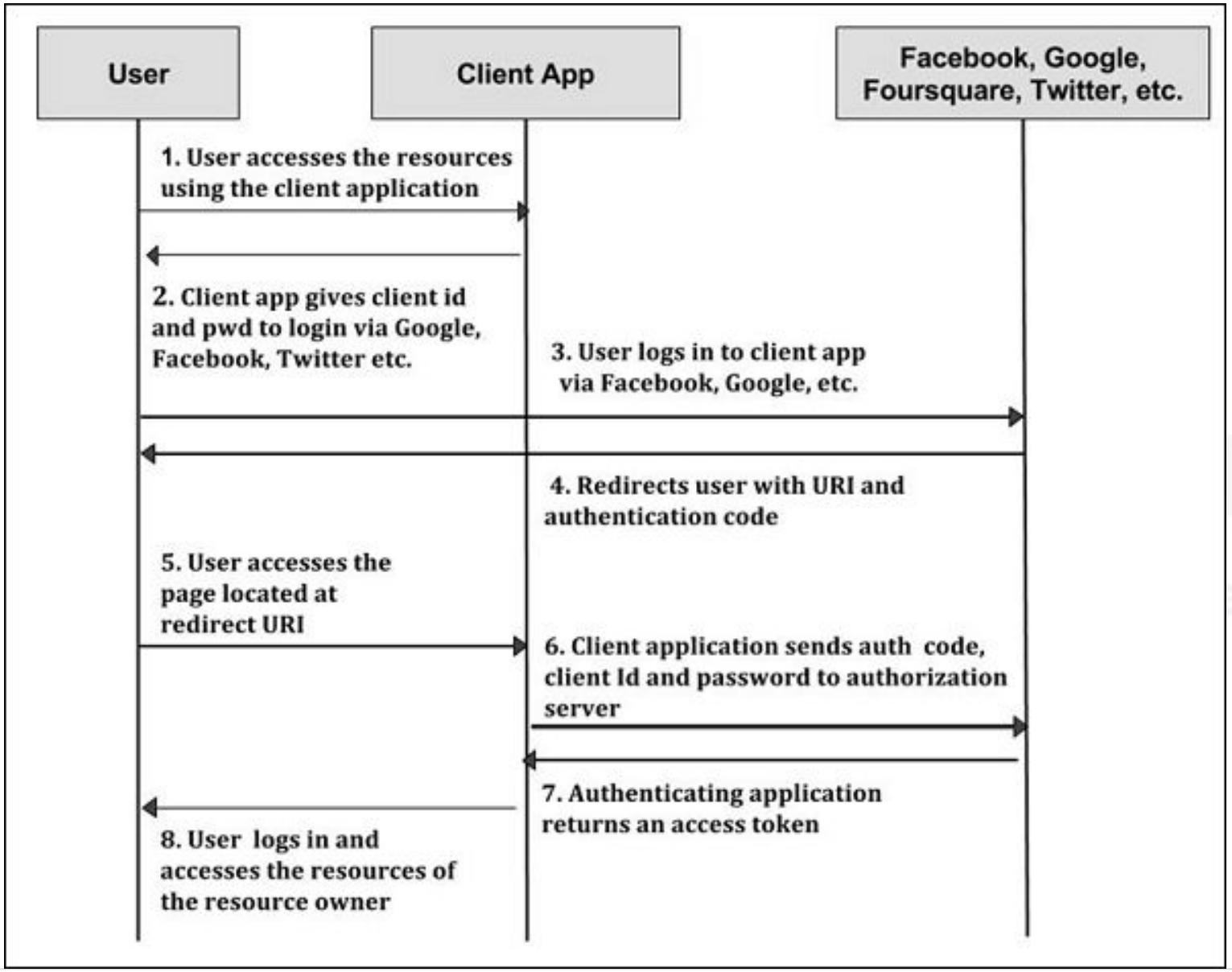


만료된 token Flow



Oauth 2.0

OAuth2.0 인증 flow

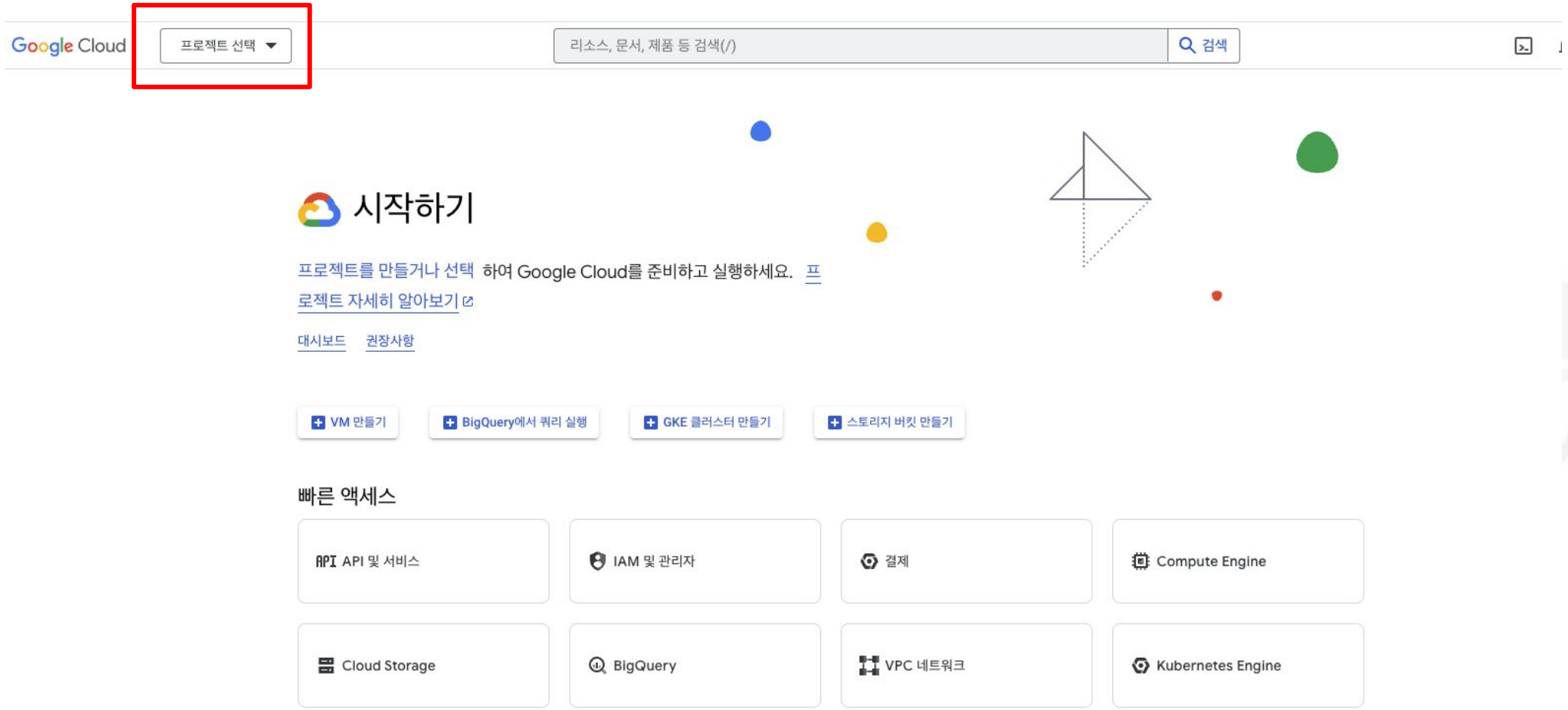


OAuth2.0

- Third-Party 프로그램에게 리소스 소유자를 대신하여 리소스 서버에서 제공하는 자원에 대한 접근 권한을 위임하는 방식을 제공
- 구글, 페이스북, 카카오, 네이버 등에서 제공하는 간편 로그인 기능도 OAuth2 프로토콜 기반의 사용자 인증 기능을 제공

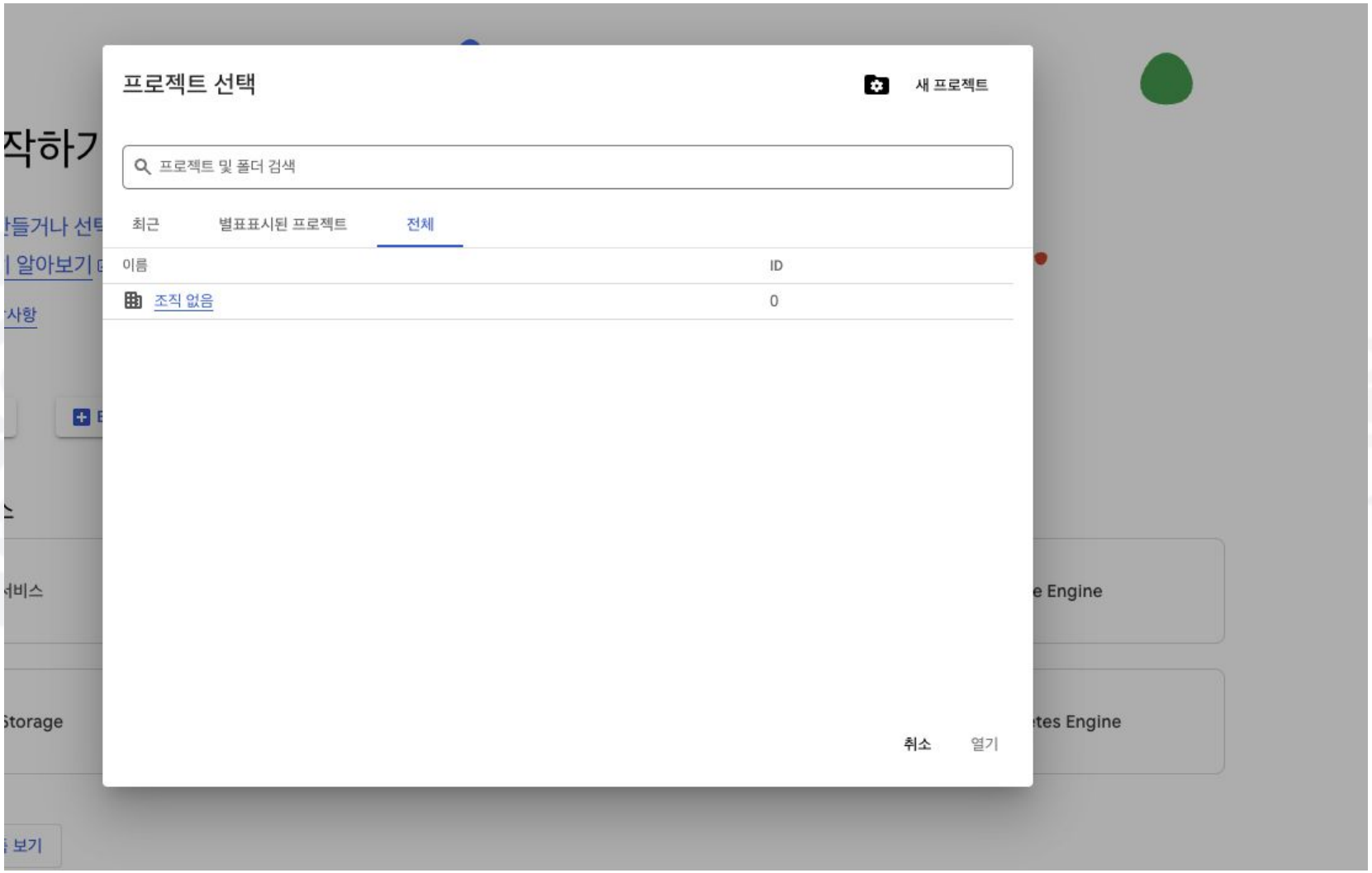
Google app

https://console.cloud.google.com/

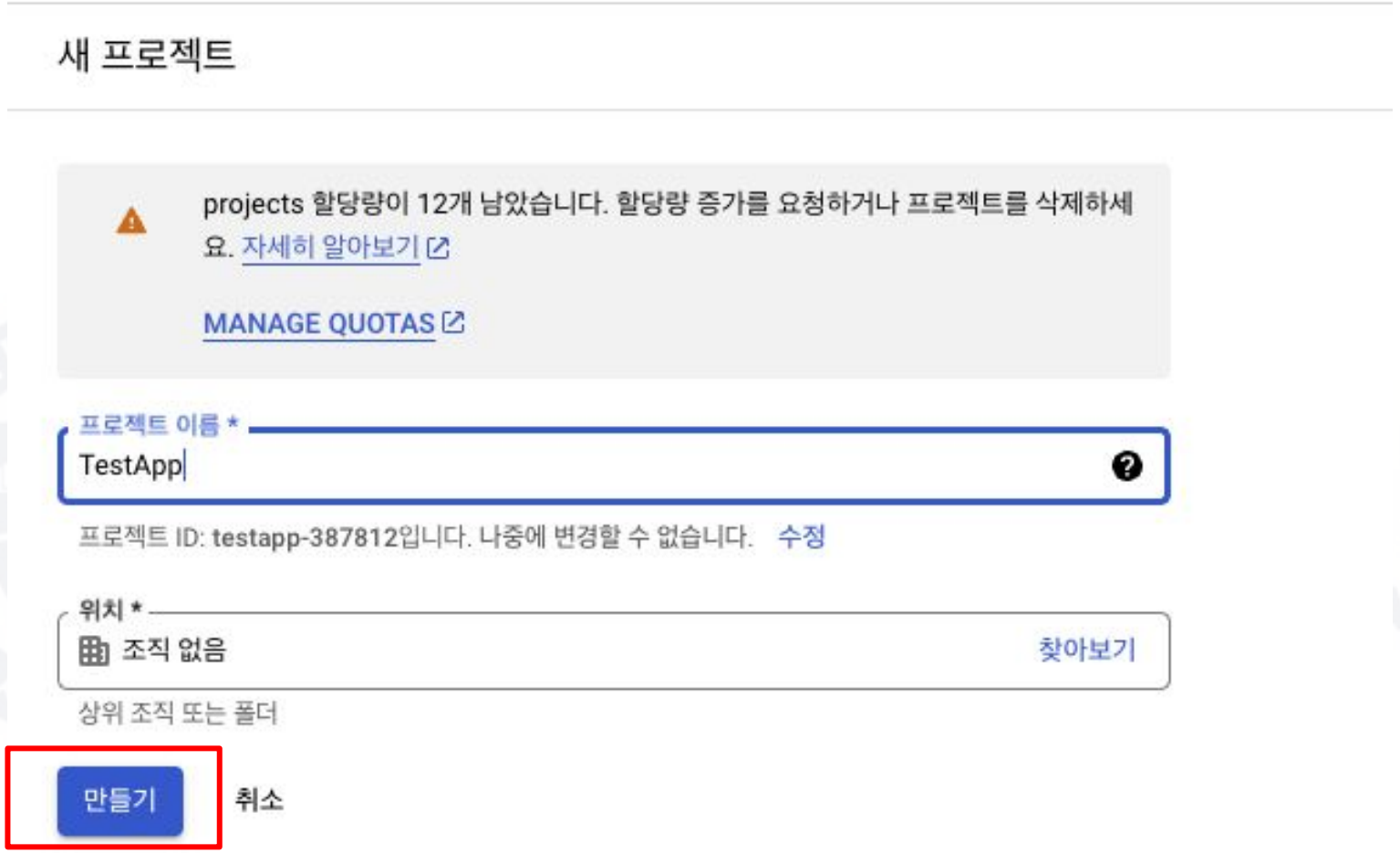


Google app

`새 프로젝트`

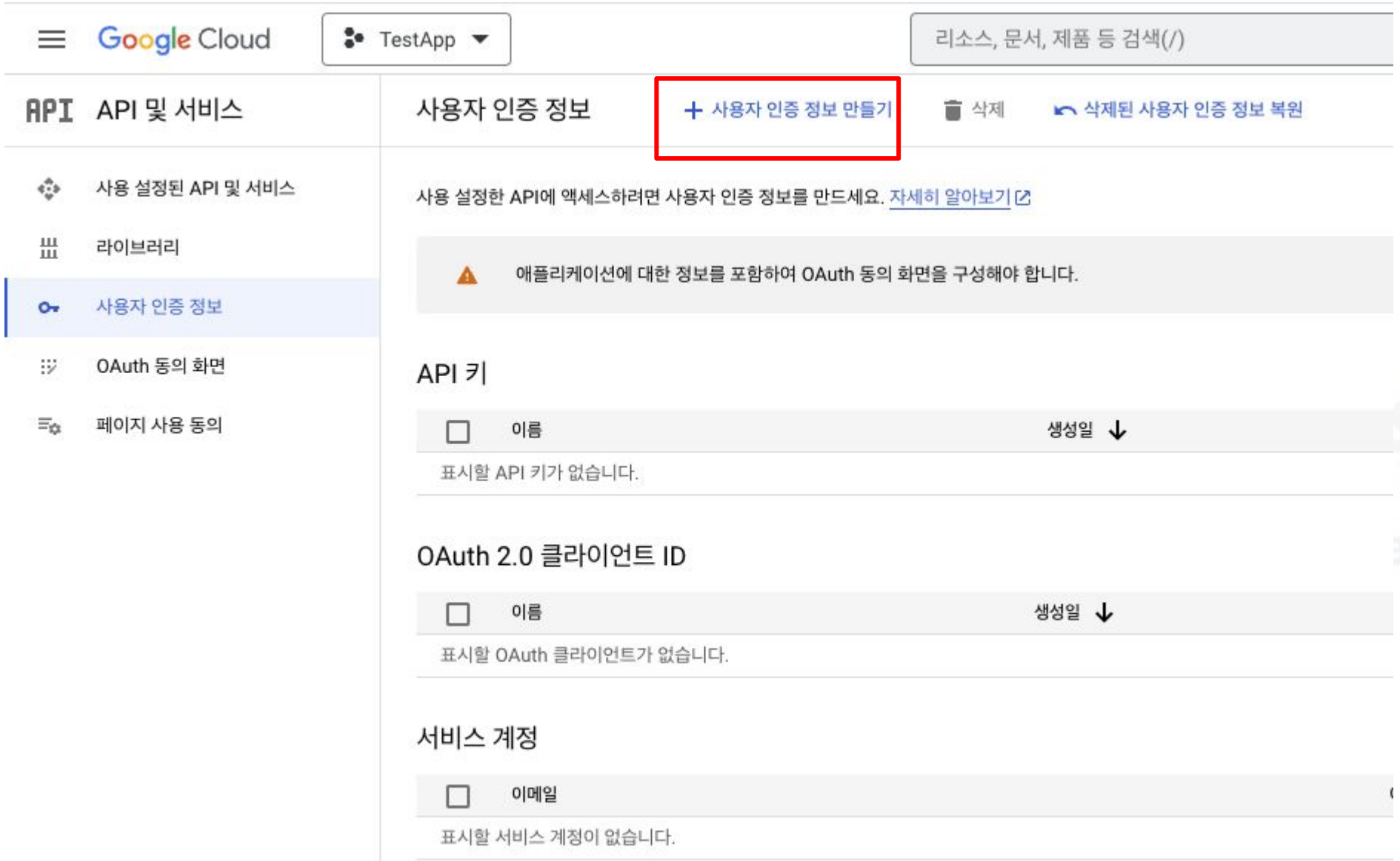


Google app

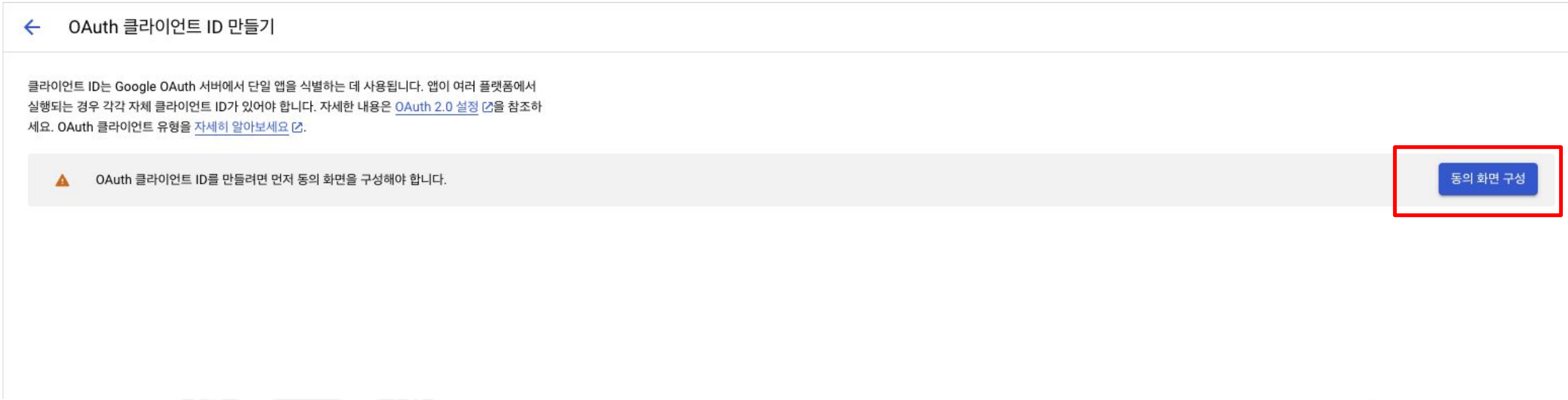


Google app

Google app

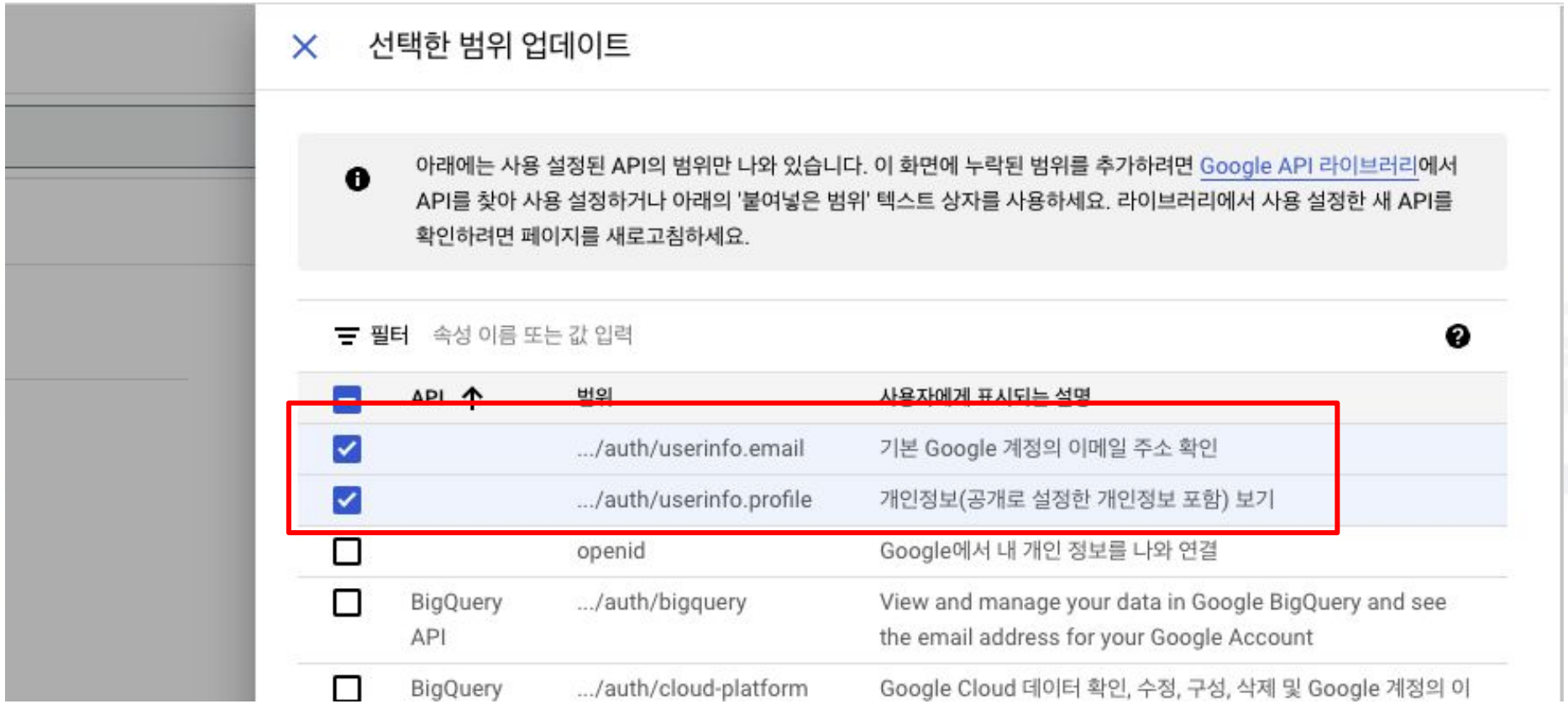


Google app



Google app

`범위 추가 삭제`



Google app

호스트/login/oauth2/code/google 로 설정

클라이언트 ID, 클라이언트 보안 비밀번호 복사

Notes

- Spring security 를 사용하지 않고도 구현 가능
- 요구사항에 맞춰 security 설정
- 서비스가 요구하는 보안 수준과 사용자 편의에 맞게 구현


“Done is better
than perfect.”
- Mark Zuckerberg

To DO

- ❑ 사용자 Entity & Repository 구현
- ❑ API 보안 구현
- ❑ 팀 프로젝트 인증/회원 기능 구현

EOD