

A.Spring Boot Security

Spring-Framework 6.1.6

Spring Boot 3.2.5

Spring-Security 6.2.4

지능물류빅데이터연구소 이상현

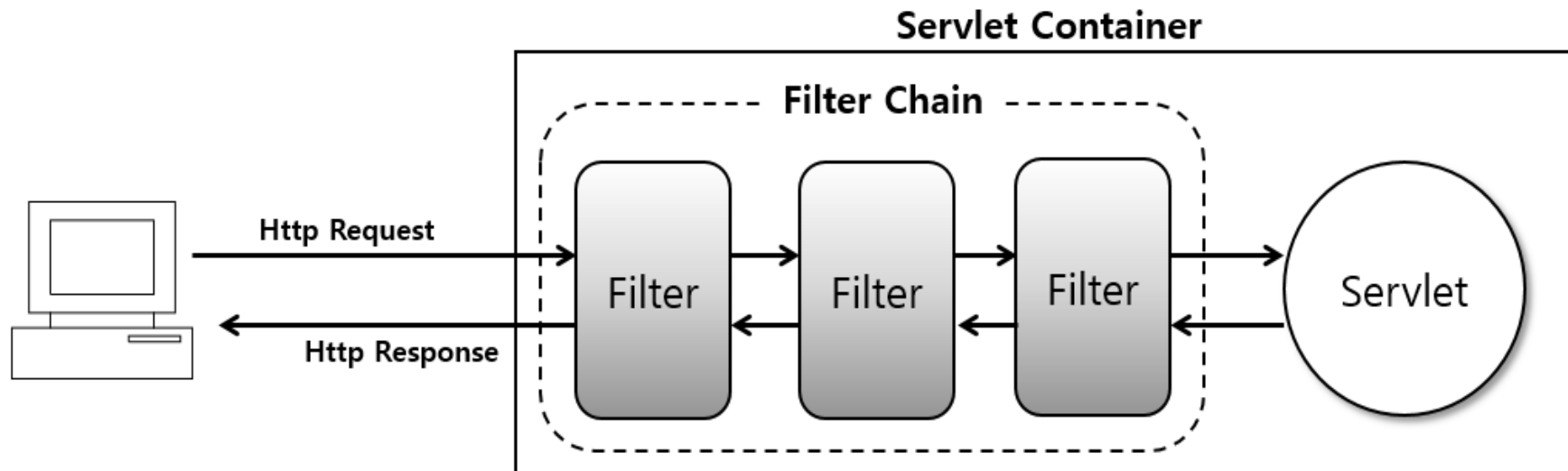
A.1. 스프링 시큐리티

A.1.1 인증(Authentication)과 인가(Authorization)

- 일반적으로 **인증(Authentication)**을 통해 사용자를 식별하고 **인가(Authorization)**를 통해 시스템 자원에 대한 접근을 통제한다.
- 예를 들어 사원증이나 **RFID** 카드를 이용해서 인증에 통과한 직원만 건물에 들어갈 수 있다.
- 직급과 직무에 따라 권한이 다르기 때문에 열람할 수 있는 문서도 제한된다. 이렇게 특정 자원에 접근할 때 권한을 체크하는 것이 인가다.

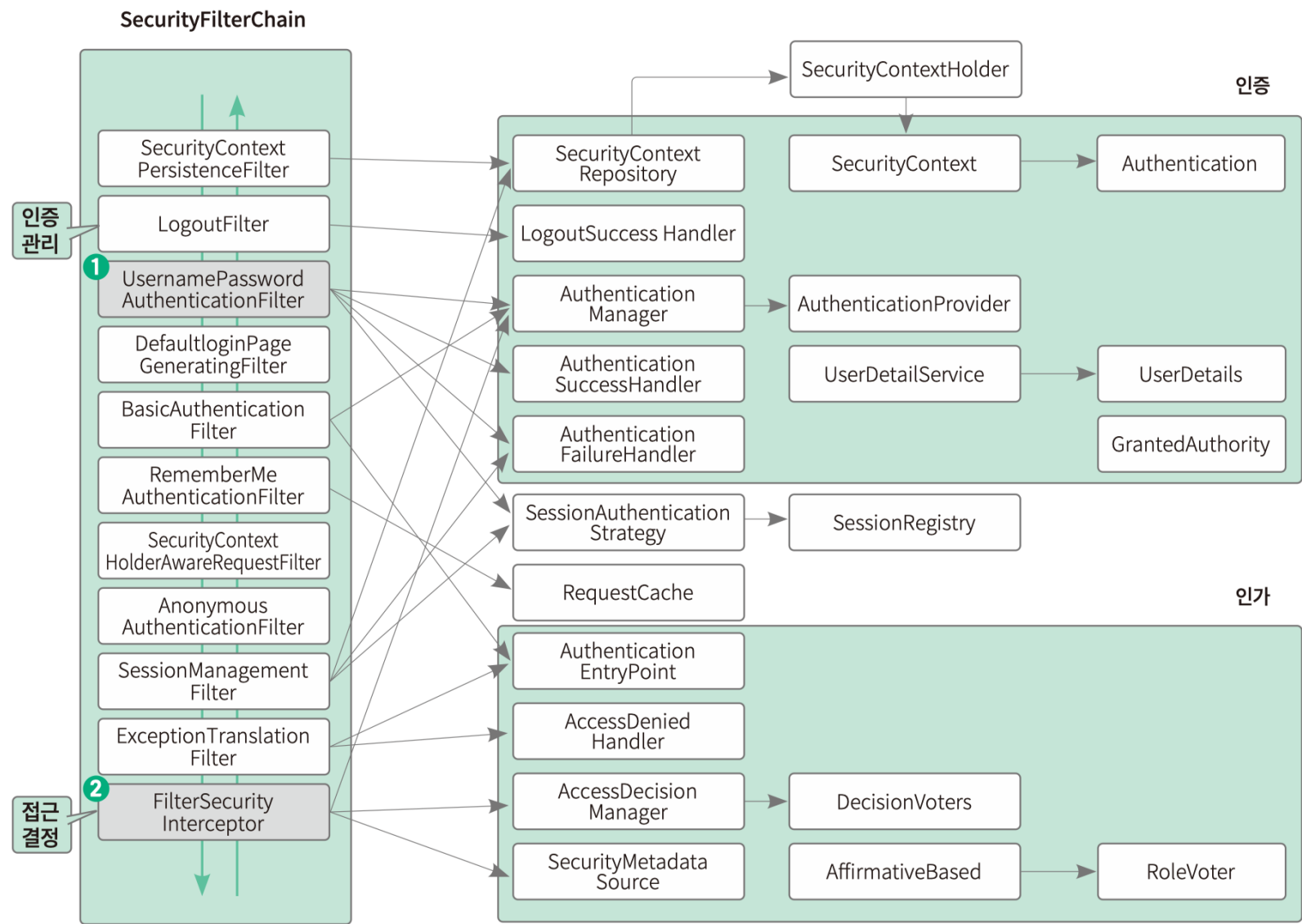
A.1.2 서블릿 필터

- 서블릿 필터는 클라이언트의 요청을 가로채서 전처리와 후처리를 수행하거나 요청을 리다이렉트하는 용도로 사용한다.
- 필터 하나가 하나의 기능을 처리하기 때문에 여러 기능이 필요한 경우에는 여러 개의 필터를 만들어 **필터 체인**을 형성하여 사용한다.



<https://docs.spring.io/spring-security/reference/servlet/architecture.html>

A.1.3 스프링 시큐리티 필터 체인



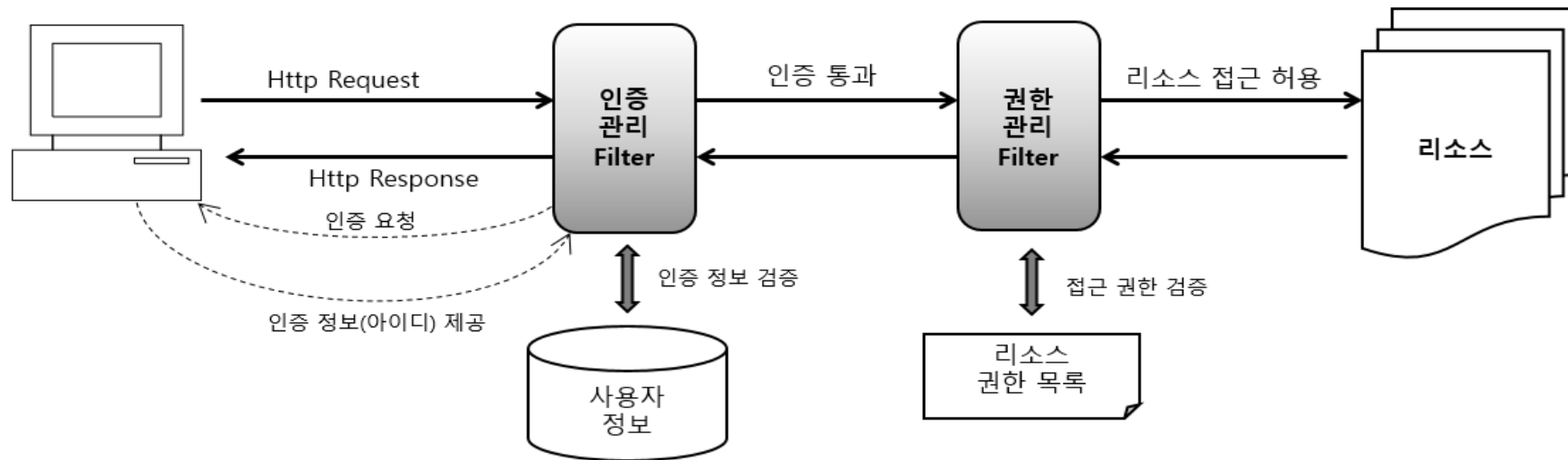
<https://docs.spring.io/spring-security/site/docs/5.4.2/reference/html5/#servlet-security-filters>

A.1.4 스프링 시큐리티 필터

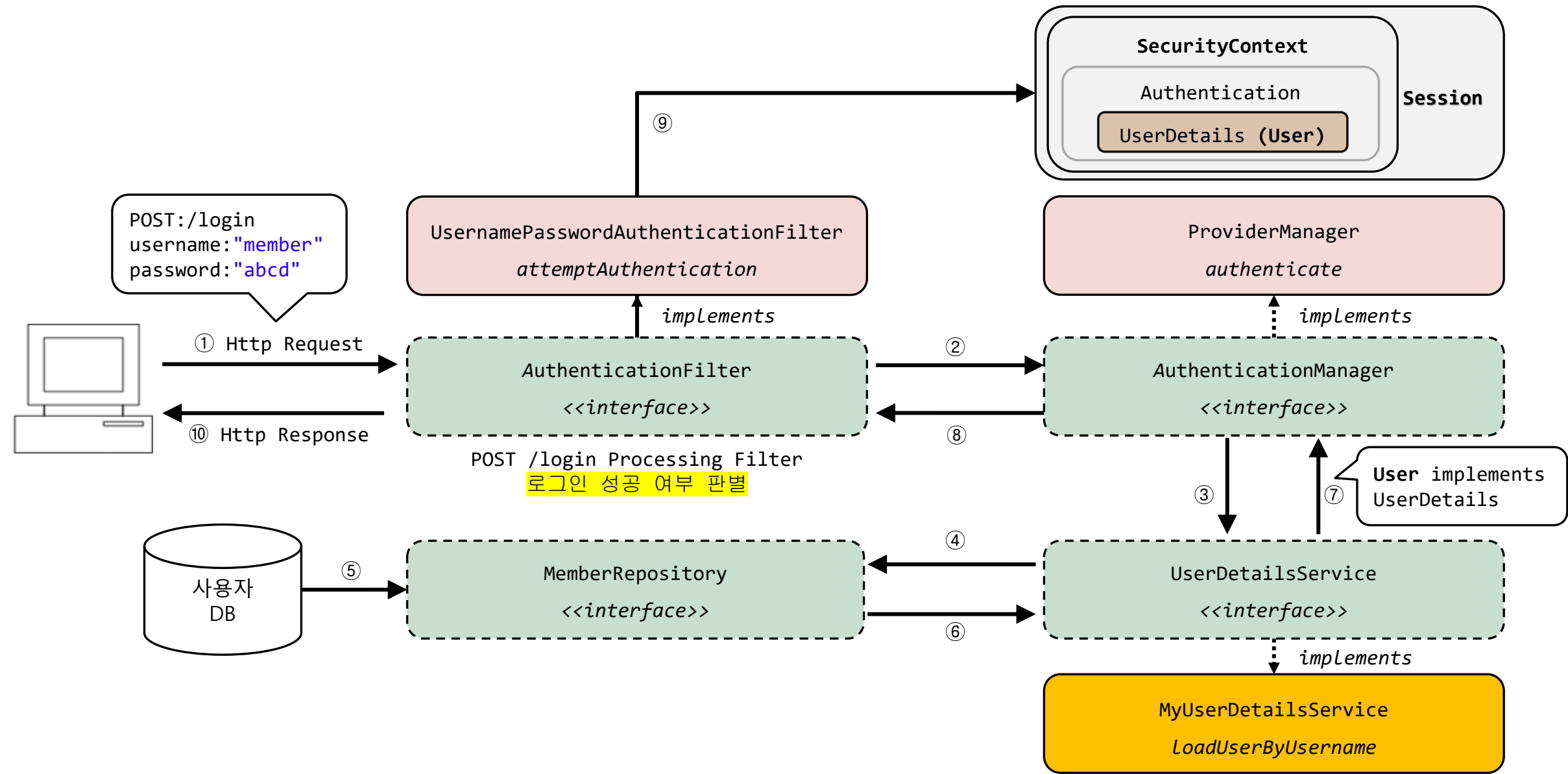
필터	기능
<code>UsernamePasswordAuthenticationFilter</code>	로그인 요청이 들어오면, 아이디/비밀번호 기반의 인증을 수행 인증에 성공하면 지정한 페이지로 이동, 실패하면 로그인 화면으로 이동
<code>OAuth2AuthorizationRequestRedirectFilter</code>	OAuth2.0 인증 요청을 처리
<code>DefaultLoginPageGeneratingFilter</code>	로그인 요청이 들어오면 기본으로 제공하는 로그인 화면 출력
<code>LoginFilter</code>	로그인 요청이 들어오면 설정한 로그인 화면 출력
<code>DefaultLogoutPageGeneratingFilter</code>	로그아웃 요청이 들어오면 기본으로 제공하는 로그아웃 화면 출력
<code>LogoutFilter</code>	지정한 경로의 요청이 들어오면 로그아웃하고 지정한 페이지로 이동
<code>BasicAuthenticationFilter</code>	Base64로 인코딩 된 토큰과 Authorization 인증 요청을 처리
<code>AuthorizationFilter</code>	URL 접근 권한을 제한 (5.5이전 버전 : FilterSecurityInterceptor)

A.1.5 스프링 시큐리티 동작 원리

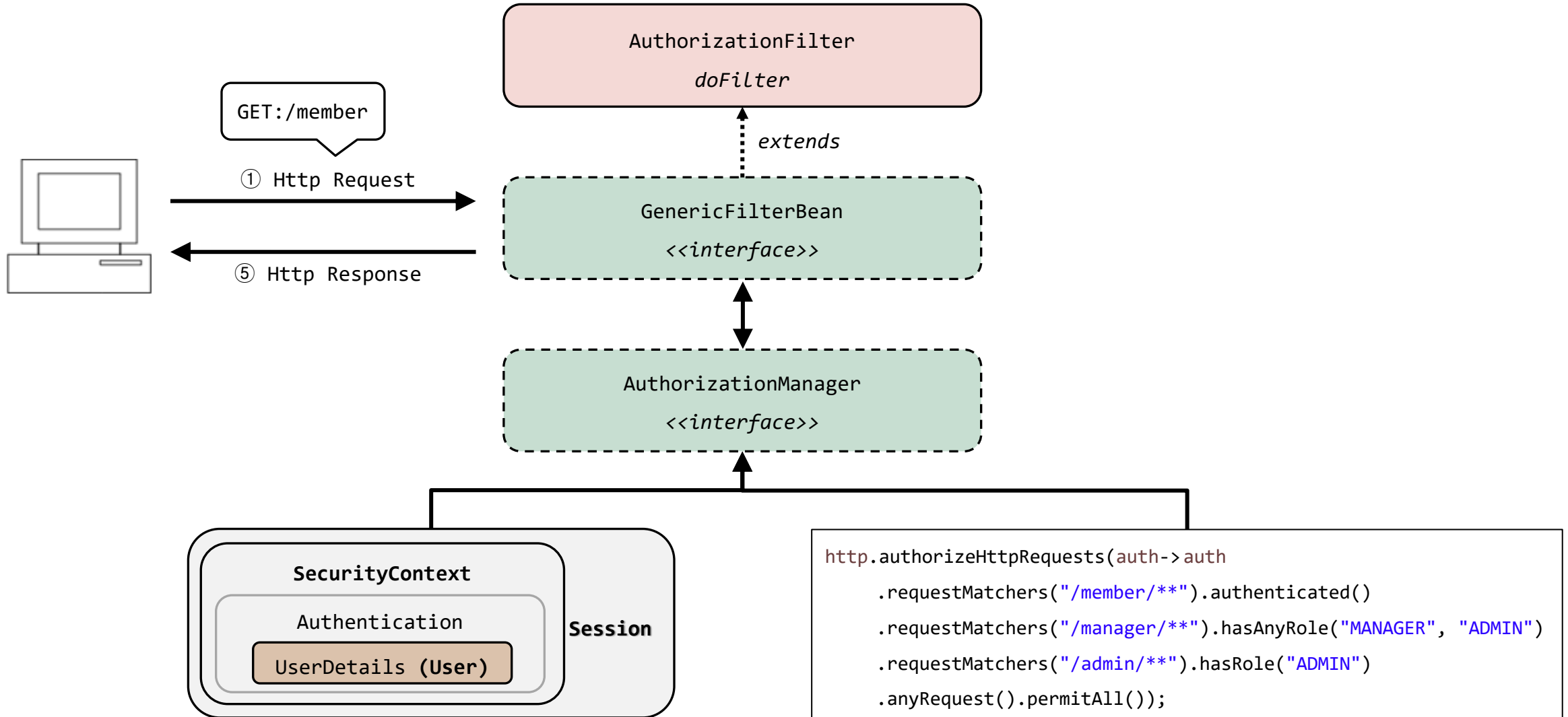
- 시큐리티 필터 중에서 **UsernamePasswordAuthenticationFilter**가 실제로 사용자가 입력한 인증 정보를 이용해서 인증을 처리해 준다.
- 그리고 **AuthorizationFilter**는 인증에 성공한 사용자가 해당 리소스에 접근할 **권한**이 있는지 검증한다.



A.1.6 스프링 시큐리티 인증 절차 개략도

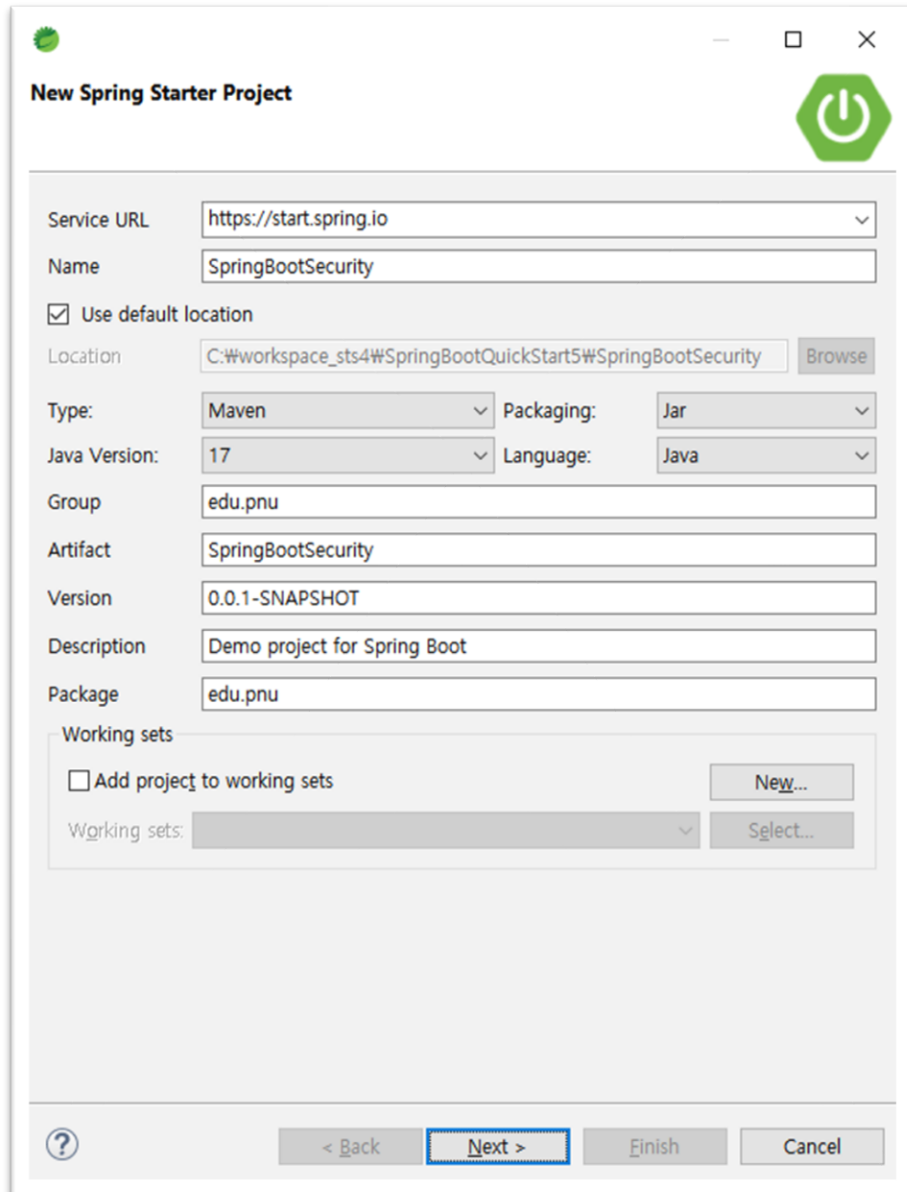


A.1.7 스프링 시큐리티 인가 절차 개략도



A.2. 시큐리티 프로젝트 시작

A.2.1 프로젝트 생성



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

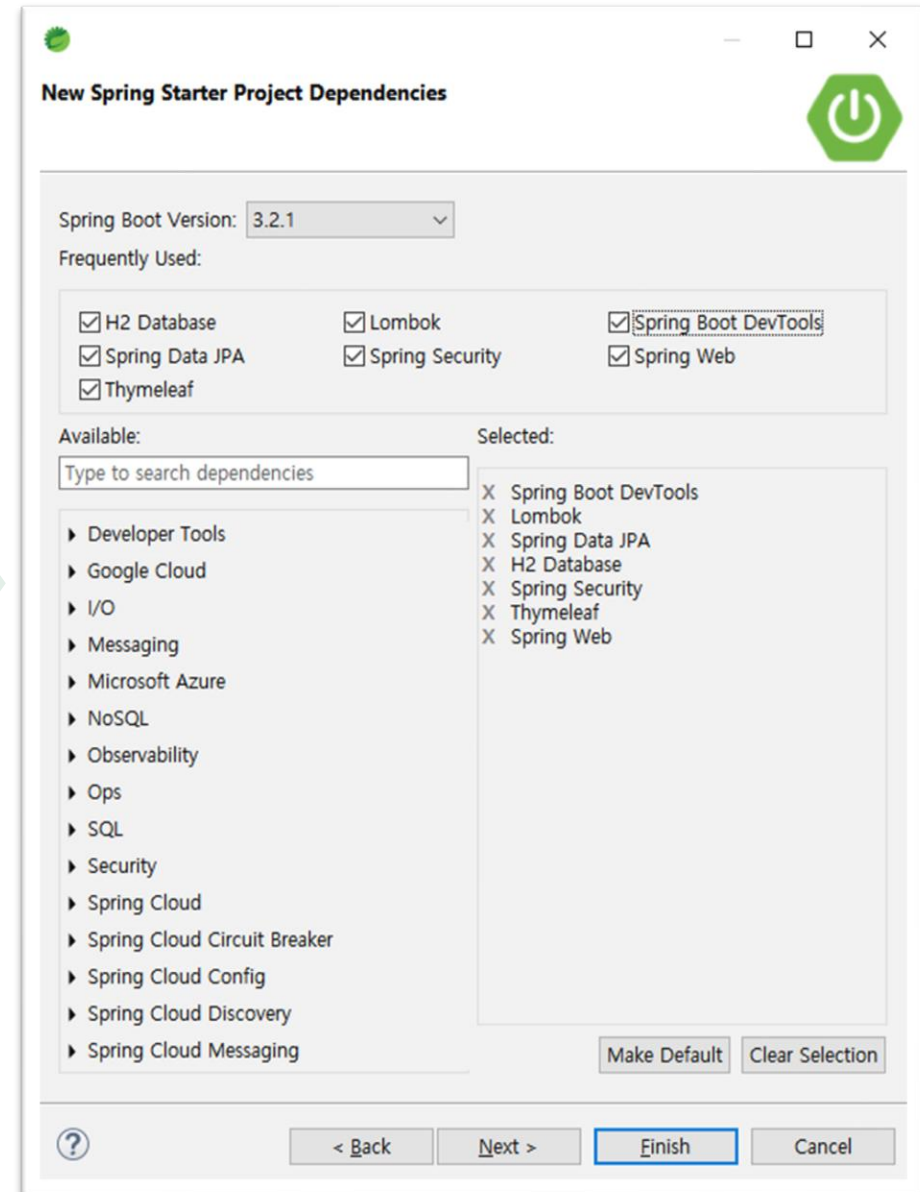
Description:

Package:

Working sets

☐ Add project to working sets

Working sets:



New Spring Starter Project Dependencies

Spring Boot Version:

Frequently Used:

☒ H2 Database ☒ Lombok ☒ Spring Boot DevTools

☒ Spring Data JPA ☒ Spring Security ☒ Spring Web

☒ Thymeleaf

Available:

Type to search dependencies

- Developer Tools
- Google Cloud
- I/O
- Messaging
- Microsoft Azure
- NoSQL
- Observability
- Ops
- SQL
- Security
- Spring Cloud
- Spring Cloud Circuit Breaker
- Spring Cloud Config
- Spring Cloud Discovery
- Spring Cloud Messaging

Selected:

- X Spring Boot DevTools
- X Lombok
- X Spring Data JPA
- X H2 Database
- X Spring Security
- X Thymeleaf
- X Spring Web

A.2.2 시큐리티 사용자

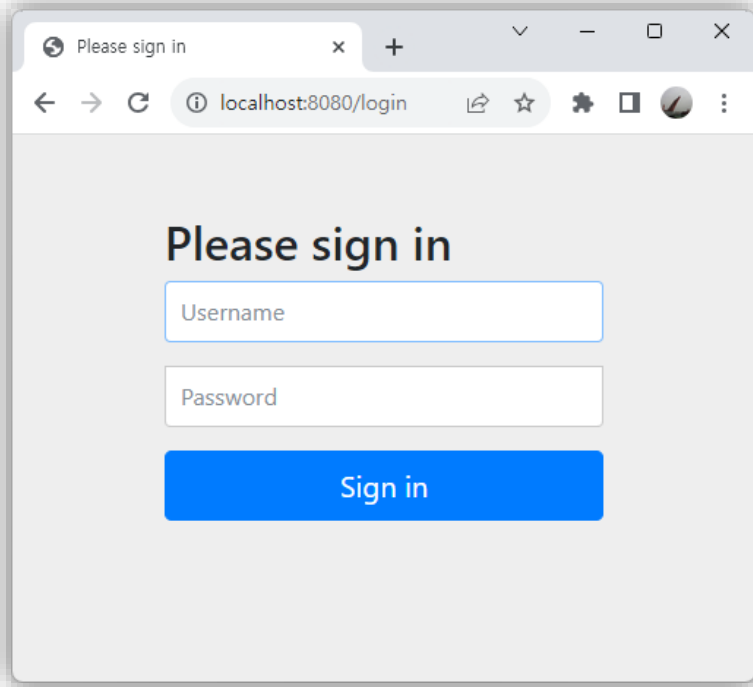
- 시큐리티 스타터만 추가해도 시큐리티 관련 기본 설정들이 자동으로 처리된다.
- **프로젝트를 생성하고 바로 실행**하면, 아이디가 'user'인 사용자가 자동으로 생성된다.
- 사용자 'user'의 비밀번호는 콘솔에 'Using generated security password : '로 출력되는 문자열이다.

```
Using generated security password: 3dba1986-48d9-4a75-8464-4942d44e6d6c
2019-03-27 14:08:34.663 INFO 19788 --- [ restartedMain] o.s.s.web.DefaultSecuri
2019-03-27 14:08:34.763 INFO 19788 --- [ restartedMain] o.s.b.w.embedded.tomcat
2019-03-27 14:08:34.766 INFO 19788 --- [ restartedMain] com.rubypaper.Chapter06.
```

- 브라우저에서 <http://localhost:8080>을 입력하면 로그인 화면이 자동으로 나타난다.
 - username : user
 - password : 자동 생성 암호
- ➔ 현재는 로그인을 해도 url에 매핑되어 있는 컨트롤러가 없기 때문에 404(Not Found)에러가 발생한다.

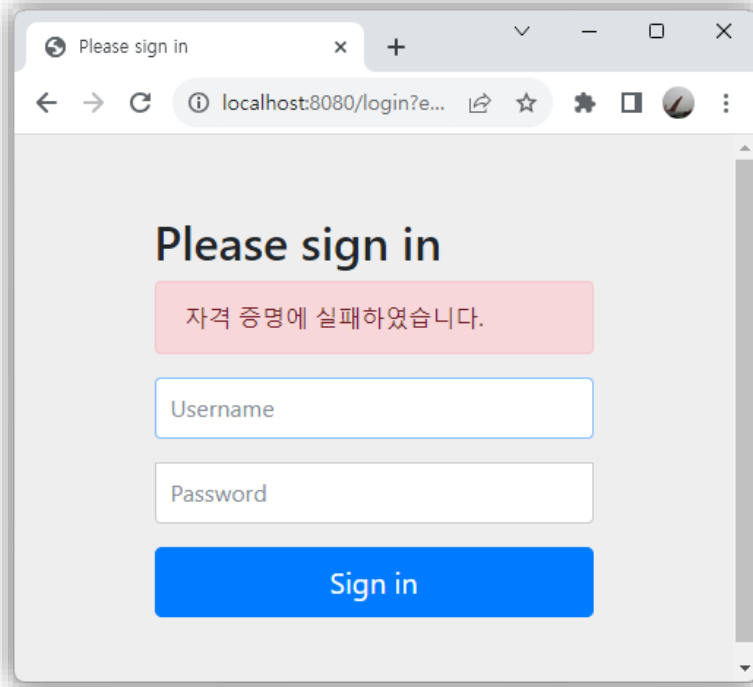
A.2.3 스프링 시큐리티 로그인 화면

로그인



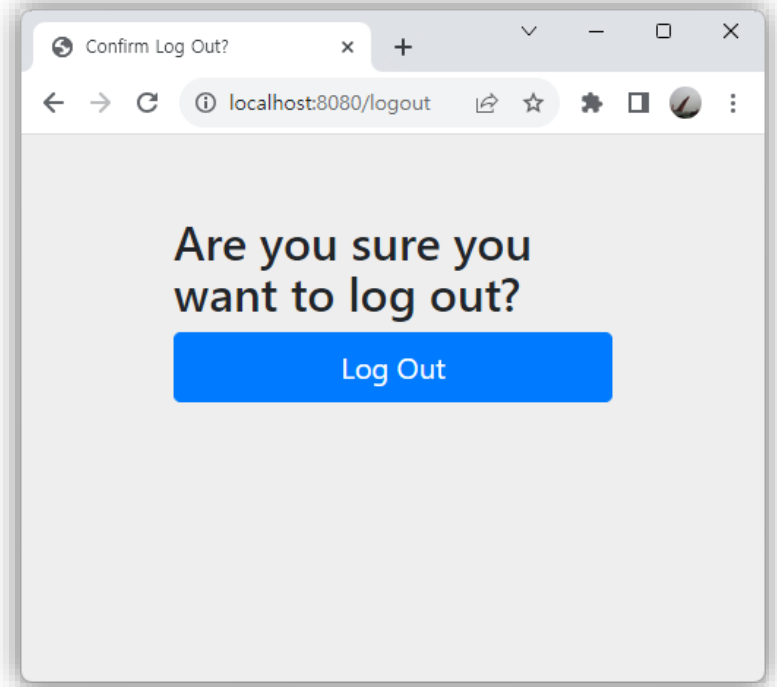
A browser window titled "Please sign in" with the address bar showing "localhost:8080/login". The page has a light gray background. It features the text "Please sign in" in bold. Below it are two input fields: "Username" and "Password". At the bottom is a blue button labeled "Sign in".

로그인 실패



A browser window titled "Please sign in" with the address bar showing "localhost:8080/login?e...". The page has a light gray background. It features the text "Please sign in" in bold. Above the input fields is a red error message box containing the text "자격 증명에 실패하였습니다." (Authentication failed). Below the error message are the "Username" and "Password" input fields. At the bottom is a blue button labeled "Sign in".

로그아웃



A browser window titled "Confirm Log Out?" with the address bar showing "localhost:8080/logout". The page has a light gray background. It features the text "Are you sure you want to log out?" in bold. Below the text is a blue button labeled "Log Out".

A.2.4 시큐리티 화면 구성

- 시큐리티 적용 시나리오

요청 URL	의미
/	인증을 하지 않은 모든 사용자가 접근할 수 있다.
/member	인증을 통과한 사용자만 접근할 수 있다.
/manager	인증을 통과했고, MANAGER 권한과 ADMIN 권한을 가진 사용자만 접근할 수 있다.
/admin	인증을 통과했고, ADMIN 권한을 가진 사용자만 접근할 수 있다.

A.2.5 시큐리티 컨트롤러 & View - 테스트를 위한 컨트롤러 작성 (p415)

Controller : *edu.pnu.controller.SecurityController.java*

```
@Controller
public class SecurityController {

    @GetMapping("/{", "/index"})
    public String index() {
        System.out.println("index 요청");
        return "index";
    }

    @GetMapping("/member")
    public void member() {
        System.out.println("Member 요청");
    }

    @GetMapping("/manager")
    public void manager() {
        System.out.println("Manager 요청");
    }
}
```

```
@GetMapping("/admin")
public void admin() {
    System.out.println("Admin 요청");
}

@GetMapping("/loginSuccess")
public void loginSuccess() {
    System.out.println("loginSuccess 요청");
}
}
```

- 컨트롤러 메소드 리턴에 따른 View 호출
 1. String이면 [return명.html]을 호출하라는 의미
 - return "index"
 - index.html
 2. void면 [url명.html]을 호출하라는 의미.
 - /member
 - member.html

A.2.6 시큐리티 컨트롤러 & View - 테스트를 위한 View 작성 (p416 ~ 417)

View

src/main/resources/templates/index.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"> >
<head>
    <meta charset="UTF-8">
    <title>시큐리티 테스트</title>
</head>
<body>
    <h1>Index 화면입니다.</h1>
    <a th:href="@{/loginSuccess}">loginSuccess로 가기</a>
</body>
</html>
```

src/main/resources/templates/member.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>시큐리티 테스트</title>
</head>
<body>
    <h1>로그인 성공한 사용자만 접근할 수 있는 화면입니다.</h1>
    <a th:href="@{/loginSuccess}">뒤로 가기</a>
</body>
</html>
```

src/main/resources/templates/manager.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>시큐리티 테스트</title>
</head>
<body>
    <h1>Manager 권한을 가진 사용자를 위한 화면입니다.</h1>
    <a th:href="@{/loginSuccess}">뒤로 가기</a>
</body>
</html>
```

src/main/resources/templates/admin.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>시큐리티 테스트</title>
</head>
<body>
    <h1>Admin 권한을 가진 사용자를 위한 화면입니다.</h1>
    <a th:href="@{/loginSuccess}">뒤로 가기</a>
</body>
</html>
```

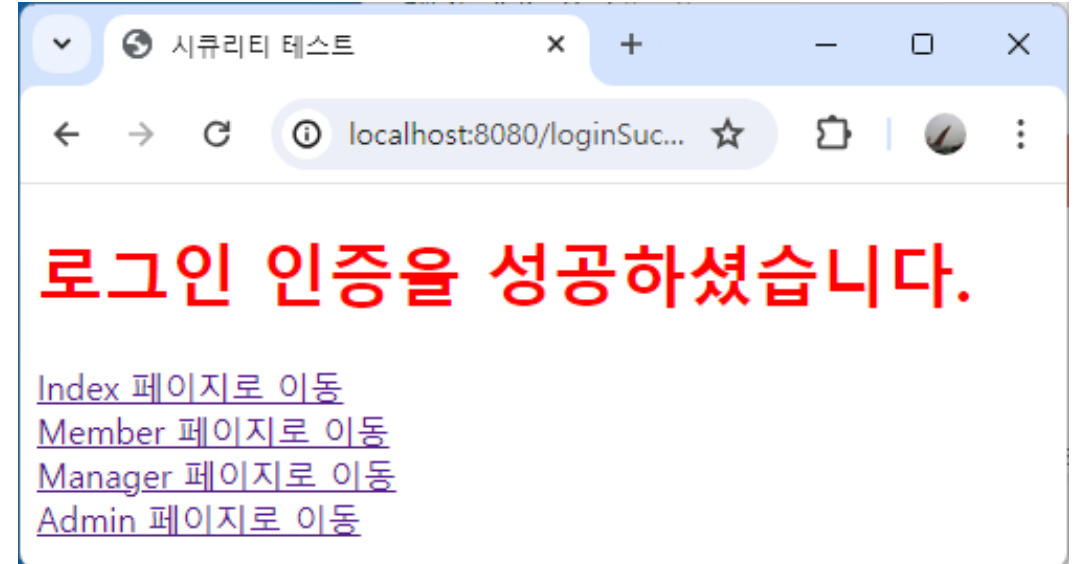

A.2.7 로그인 성공 화면 적용하기 (p424)

서버 실행 후 테스트

View

src/main/resources/templates/loginSuccess.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>시큐리티 테스트</title>
</head>
<body>
    <h1><font color="red">로그인 인증을 성공하셨습니다.</font></h1>
    <a th:href="@{/}">Index 페이지로 이동</a><br/>
    <a th:href="@{/member}">Member 페이지로 이동</a><br/>
    <a th:href="@{/manager}">Manager 페이지로 이동</a><br/>
    <a th:href="@{/admin}">Admin 페이지로 이동</a>
</body>
</html>
```



- 여기까지 완성되면 서버를 실행해서 로그인 후 모든 url에 대해 정상 작동하는지 확인한다.
- url에 대한 권한 설정을 아직 하지 않았기 때문에 모든 url이 호출된다.

A.3. 시큐리티 커스터마이징하기

A.3.1 시큐리티 설정 파일 작성

서버 실행 후 테스트

edu.pnu.config.SecurityConfig.java

```
@Configuration .....> 이 클래스가 설정 클래스라고 정의 (IoC 컨테이너에 로드)
@EnableWebSecurity .....> 스프링 시큐리티 적용에 필요한 필터 객체들 자동 생성

public class SecurityConfig {
    @Bean .....> 이 메서드가 리턴 하는 객체를 IoC 컨테이너에 등록하라는 지시
    SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

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

- SecurityFilterChain 객체를 생성해서 Bean으로 등록하면 기본 로그인 화면이 나타나지 않는다. 기본 로그인 화면을 사용하거나 사용자가 작성한 로그인 화면을 사용하겠다는 설정을 해야 한다.
- 여기까지 완성되면 서버를 실행해서 정상적으로 실행되는 지 확인 (로그인 없이 모든 URL에 접근 가능하면 정상)

A.3.2 시큐리티 설정 파일 수정

서버 실행 후 테스트

edu.pnu.config.SecurityConfig.java

@Bean

```
SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
```

```
    http.authorizeHttpRequests(security->security
```

```
        .requestMatchers("/member/**").authenticated()
```

```
        .requestMatchers("/manager/**").hasAnyRole("MANAGER", "ADMIN")
```

```
        .requestMatchers("/admin/**").hasRole("ADMIN")
```

```
        .anyRequest().permitAll());
```

접근 권한 설정

```
    http.csrf(cf->cf.disable());
```



CSRF 보호 비활성화 (사이트간 요청 위조)

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

```
}
```

- 여기까지 완성되면 서버를 실행해서 **/index**, **/loginSuccess**는 정상적으로 호출이 되지만, 다른 url들은 **403 (Forbidden – Access Denied)**에러가 나타나는지 확인한다.
- 로그인 화면을 어떤 것을 사용할지 아직 설정하지 않아서 **/login**을 호출해도 에러가 발생한다.

A.3.3 사용자 인증하기

서버 실행 후 테스트

```
@Bean
SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    ~ 생략 ~

    http.formLogin(form->{}); .....> SpringBoot가 제공하는 로그인 사용하겠다는 설정

    return http.build();
}
```

- 완성되면 서버를 실행해서

- `/index`가 정상적으로 호출되는지 확인
- `/member`, `/manager`, `/admin`을 호출하면 로그인 화면이 나타나는지 확인
- `/member`를 호출했을 때 로그인 화면에서 **user/기본 암호**를 입력해서 정상 작동하는지 확인한다.
- 로그인 이후에 `/manager`와 `/admin`은 **403 (Forbidden – Access Denied)**에러가 나타나는지 확인한다.

A.3.4 사용자 정의 로그인 화면 적용 및 컨트롤러 작성

SecurityConfig.java

```
SecurityFilterChain securityFilterChain(HttpSecurity http)
    throws Exception {
    :
    http.formLogin(form->
        form.loginPage("/login")
            .defaultSuccessUrl("/loginSuccess", true)
    );
    return http.build();
}
```

LoginController.java

```
@Controller
public class LoginController {
    @GetMapping("/login")
    public void login() {
        System.out.println("login 요청");
    }

    @GetMapping("/loginSuccess")
    public void loginSuccess() {
        System.out.println(
            "loginSuccess 요청");
    }
}

// SecurityController에 loginSuccess 삭제
```

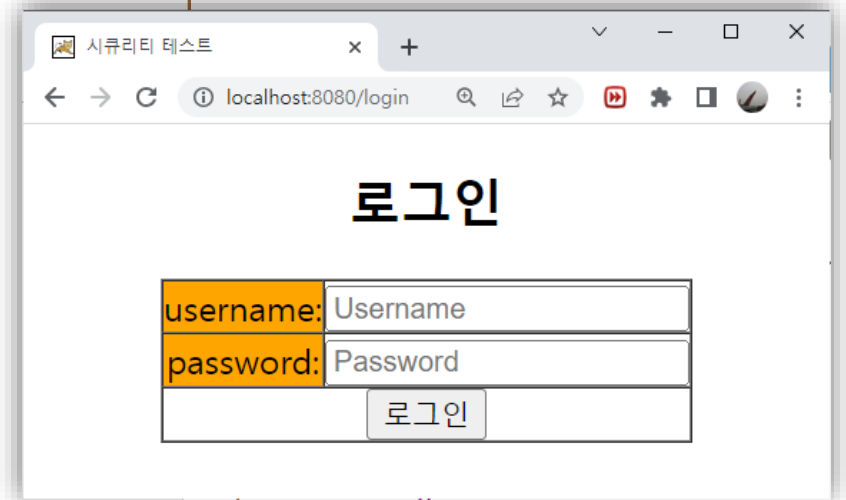
- /member를 호출해서 로그인 화면으로 왔을 때 로그인에 성공한 뒤 /loginSuccess로 이동하겠다는 의미
- 그렇지 않고 로그인에 성공한 뒤 호출한 url인 /member로 이동하려면 false로 설정하면 됨.

A.3.5 사용자 정의 로그인 화면 적용하기 (1)

서버 실행 후 테스트

template/login.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>시큐리티 테스트</title>
</head>
<body th:align="center">
  <h2>로그인</h2>
  <form action="/login" method="POST">
    <table th:align="center" border="1" th:cellpadding="0" th:cellspacing="0">
      <tr>
        <td bgcolor="orange">username:</td>
        <td><input type="text" name="username" placeholder="Username"/></td>
      </tr>
      <tr>
        <td bgcolor="orange">password:</td>
        <td><input type="password" name="password" placeholder="Password"/></td>
      </tr>
      <tr>
        <td colspan="2" align="center"><input type="submit" value="로그인"/></td>
      </tr>
    </table>
  </form>
</body>
</html>
```



A.3.6 메모리 사용자 등록하기

서버 실행 후 테스트

```
public class SecurityConfig {
```

~생략~

```
@Autowired
```

```
public void authenticate(AuthenticationManagerBuilder auth) throws Exception {
```

```
    auth.inMemoryAuthentication()
```

```
        .withUser("manager")
```

```
        .password("{noop}abcd")
```

```
        .roles("MANAGER");
```

```
    auth.inMemoryAuthentication()
```

```
        .withUser("admin")
```

```
        .password("{noop}abcd")
```

```
        .roles("ADMIN");
```

```
}
```

```
}
```

{noop} : No Operation → 비밀번호가 암호화되어 있지 않다는 의미

- 임시 테스트용 사용자 계정을 메모리에 등록해서 권한 관계가 제대로 작동하는지 확인한다.
- 완성되면 서버를 실행해서 모든 url에 대해서 정상 작동하는지 확인한다.

A.3.7 접근 권한 없음 페이지 처리

서버 실행 후 테스트

SecurityConfig.java

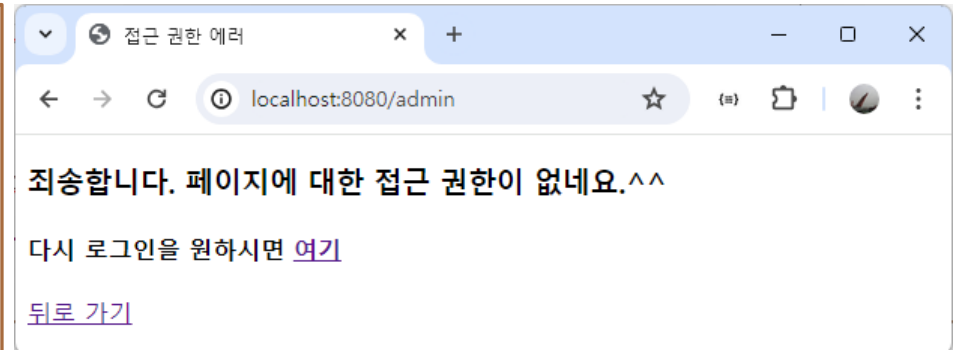
```
SecurityFilterChain securityFilterChain(HttpSecurity http)
    throws Exception {
    :
    http.exceptionHandling(ex->ex.accessDeniedPage("/accessDenied"));
    return http.build();
}
```

LoginController.java

```
class LoginController {
    :
    @GetMapping("/accessDenied")
    public void accessDenied() {
        System.out.println("accessDenied");
    }
}
```

accessDenied.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>접근 권한 에러</title>
</head>
<body>
    <h3>죄송합니다. 페이지에 대한 접근 권한이 없네요.^^</h3>
    <h4>다시 로그인을 원하시면 <a th:href="@{/Login}">여기</a></h4>
    <a th:href="@{/LoginSuccess}">뒤로 가기</a>
</body>
</html>
```



- 완성되면 서버를 실행해서 **manager**로 로그인 한 뒤에 모든 url에 접근 테스트를 해서 정상 작동하는지 확인한다.
 - **/, /member, /manager**는 정상
 - **/admin**은 접근 권한 에러 메시지 발생

A.3.8 로그아웃 처리

서버 실행 후 테스트

SecurityConfig.java

```
SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
    :  
    http.logout(logout->logout  
        .invalidateHttpSession(true)  
        .deleteCookies("JSESSIONID")  
        .logoutSuccessUrl("/login"));  
    return http.build();  
}
```

현재 브라우저와 연결된 세션 강제 종료

세션 아이디가 저장된 쿠키 삭제

로그아웃 후 이동할 URL 지정

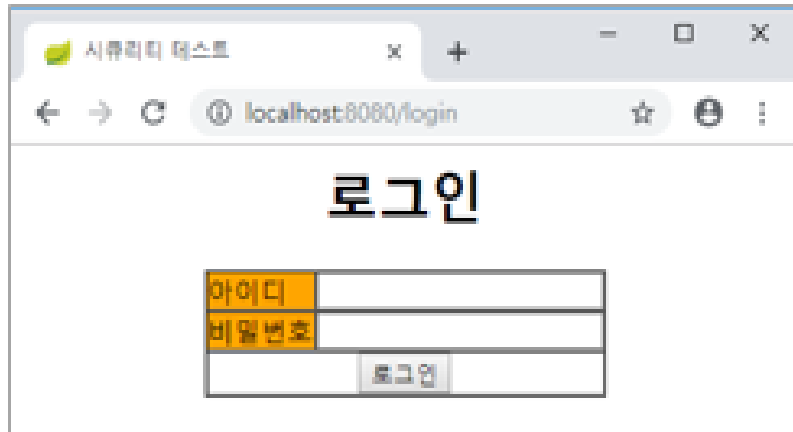
src/main/resources/templates/manager.html 수정

```
    :  
    <a th:href="@{/LoginSuccess}">뒤로 가기</a>  
    <form action="/logout" method="get">  
        <input type="submit" value="로그아웃"/>  
    </form>  
</body>  
</html>
```



- 나머지 사용자 html 파일 수정
(member.html, admin.html)

A.3.9 로그아웃 테스트 순서

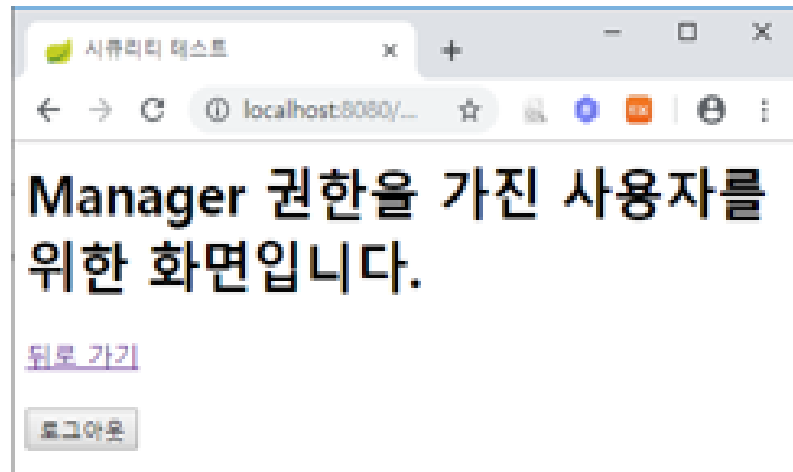


시큐리티 테스트

localhost:8080/login

로그인

아이디	<input type="text"/>
비밀번호	<input type="password"/>
<input type="button" value="로그인"/>	

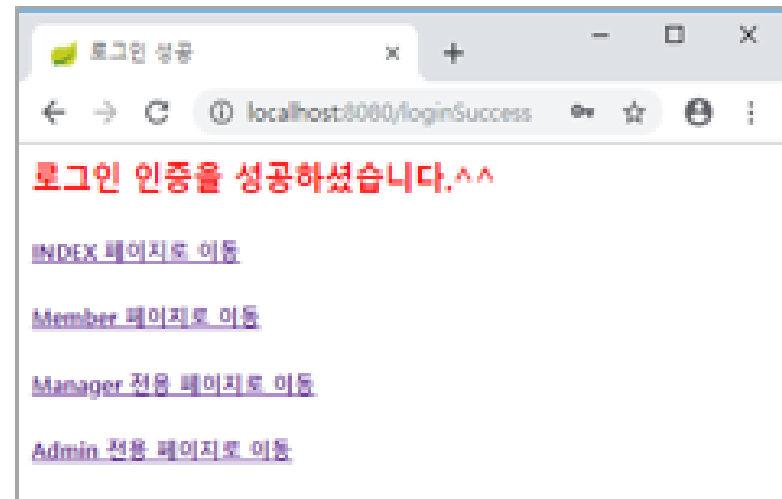


시큐리티 테스트

localhost:8080/...

Manager 권한을 가진 사용자를 위한 화면입니다.

[뒤로 가기](#)



로그인 성공

localhost:8080/loginSuccess

로그인 인증을 성공하셨습니다.^^

[INDEX 페이지로 이동](#)

[Member 페이지로 이동](#)

[Manager 전용 페이지로 이동](#)

[Admin 전용 페이지로 이동](#)



A.4. JDBC 연동 시큐리티

Skip

A.4.1 데이터베이스 준비 및 application.properties 작성

데이터베이스 준비

```
DROP TABLE IF EXISTS MEMBER;
```

```
CREATE TABLE MEMBER (  
    username varchar(10) primary key,  
    password varchar(100),  
    role varchar(12),  
    enable boolean  
);
```

```
INSERT INTO MEMBER values ('member','member123','ROLE_MEMBER', TRUE);  
INSERT INTO MEMBER values ('manager','manager123','ROLE_MANAGER', TRUE);  
INSERT INTO MEMBER values ('admin','admin123','ROLE_ADMIN', TRUE);
```

/src/main/resources/application.properties

```
spring.datasource.driver-class-name=org.h2.Driver  
spring.datasource.url=jdbc:h2:tcp://localhost/~/.h2/chapter07  
spring.datasource.username=sa  
spring.datasource.password=abcd  
spring.thymeleaf.cache=false
```

A.4.3 SecurityConfig 수정 (JDBC를 이용해서 로그인 하는 예제)

```
public class SecurityConfig {  
  
    @Autowired  
    private DataSource dataSource;  
    ~생략~  
  
    @Autowired  
    public void authenticate(AuthenticationManagerBuilder auth) throws Exception {  
  
        auth.jdbcAuthentication()  
            .dataSource(dataSource)  
            // 입력한 아이디로 사용자 정보를 조회  
            .usersByUsernameQuery("select username, concat('{noop}', password) password, "  
                                   + "enabled from member where username=?")  
            // 입력한 아이디로 사용자 권한 정보를 조회  
            .authoritiesByUsernameQuery("select username, role from member where username=?");  
    }  
}
```

A.5. JPA 연동 시큐리티

A.5.1 엔티티 클래스, Repository 작성 및 application.properties 수정

edu.pnu.domain.Role.java

```
public enum Role {  
    ROLE_ADMIN, ROLE_MANAGER, ROLE_MEMBER  
}
```

edu.pnu.domain.Member.java

```
@Getter @Setter @ToString  
@Builder  
@AllArgsConstructor  
@NoArgsConstructor  
@Entity  
public class Member {  
    @Id  
    private String username;  
    private String password;  
    @Enumerated(EnumType.STRING)  
    private Role role;  
    private boolean enabled;  
}
```

persistence.MemberRepository.java

```
package edu.pnu.persistence;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import edu.pnu.domain.Member;  
  
public interface MemberRepository extends JpaRepository<Member, String> {  
}
```

/src/main/resources/application.properties

```
spring.datasource.driver-class-name=org.h2.Driver  
spring.datasource.url=jdbc:h2:mem:test  
spring.datasource.username=sa  
spring.datasource.password=  
  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
spring.jpa.hibernate.ddl-auto=create  
spring.jpa.show-sql=true  
spring.h2.console.enabled=true
```


A.5.2 사용자 정의 UserDetailsService

- 자동 생성 사용자가 아닌 데이터베이스에 저장된 사용자 정보를 이용해서 로그인 하는 과정을 구현한다.
 - SecurityConfig에서 메모리 사용자 설정 또는 JDBC 사용자 설정코드가 있으면 삭제하거나 주석 처리한다.
- 인증 관리 필터가 인증을 처리하기 위해서는 사용자 정보가 저장된 **UserDetails** 객체와 **UserDetails** 객체에 실제 데이터베이스에서 검색한 사용자 정보를 저장하는 **UserDetailsService** 객체가 필요하다.
- 스프링 시큐리티의 인증 관리자는 **UserDetailsService** 객체를 통해 **UserDetails** 객체를 획득하고 이 **UserDetails** 객체에서 인증과 인가에 필요한 정보들을 추출하여 사용한다.
- **UserDetailsService**를 커스터마이징하고 싶으면 **UserDetailsService**를 구현한 클래스를 작성하여 등록하면 된다.
 - 스프링 부트는 **UserDetailsService**를 구현한 클래스를 기본적으로 제공하는데 이 클래스가 제공하는 **UserDetails** 객체의 아이디가 'user'이고 암호는 console에 암호화되어 나타나는 긴 문자열이다.

A.5.3 사용자 정의 UserDetailsService 구현

BoardUserDetailsService.java

```
@Service
public class BoardUserDetailsService implements UserDetailsService {

    @Autowired
    private MemberRepository memRepo;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        // memRepo에서 사용자 정보를 검색해서
        Member member = memRepo.findById(username)
                                .orElseThrow(() -> new UsernameNotFoundException("Not Found"));
        System.out.println(member);    // 검색된 사용자 정보를 console에 출력해서 확인

        // UserDetails 타입의 객체를 생성해서 리턴 (o.s.s.core.userdetails.User)
        // 여기에서 리턴된 User 객체와 로그인 요청 정보를 비교한다.
        return new User(member.getUsername(), member.getPassword(),
                        AuthorityUtils.createAuthorityList(member.getRole().toString()));
    }
}
```

A.5.4 시큐리티 재정의 – 암호화 빈 객체 등록

방법1 (추천)

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

방법2 (교재 방법) – 암호화 코드 앞에 암호화 로직 정보를 헤더에 추가함.

```
public class SecurityConfig {  
  
    @Bean  
    PasswordEncoder encoder() {  
        return PasswordEncoderFactories.createDelegatingPasswordEncoder();  
    }  
}
```

상호 호환되지 않으므로
한가지만 선택해서 사용

A.5.5 암호화 적용된 사용자 생성 및 테스트

서버 실행 후 테스트

@Component

@RequiredArgsConstructor

public class MemberInitialize implements ApplicationRunner {

private final MemberRepository memRepo;

private final PasswordEncoder encoder;

@Override

public void run(ApplicationArguments args) throws Exception {

memRepo.save(Member.builder().username("member").password(encoder.encode("abcd"))
.role(Role.ROLE_MEMBER).enabled(true).build());

memRepo.save(Member.builder().username("manager").password(encoder.encode("abcd"))
.role(Role.ROLE_MANAGER).enabled(true).build());

memRepo.save(Member.builder().username("admin").password(encoder.encode("abcd"))
.role(Role.ROLE_ADMIN).enabled(true).build());

}

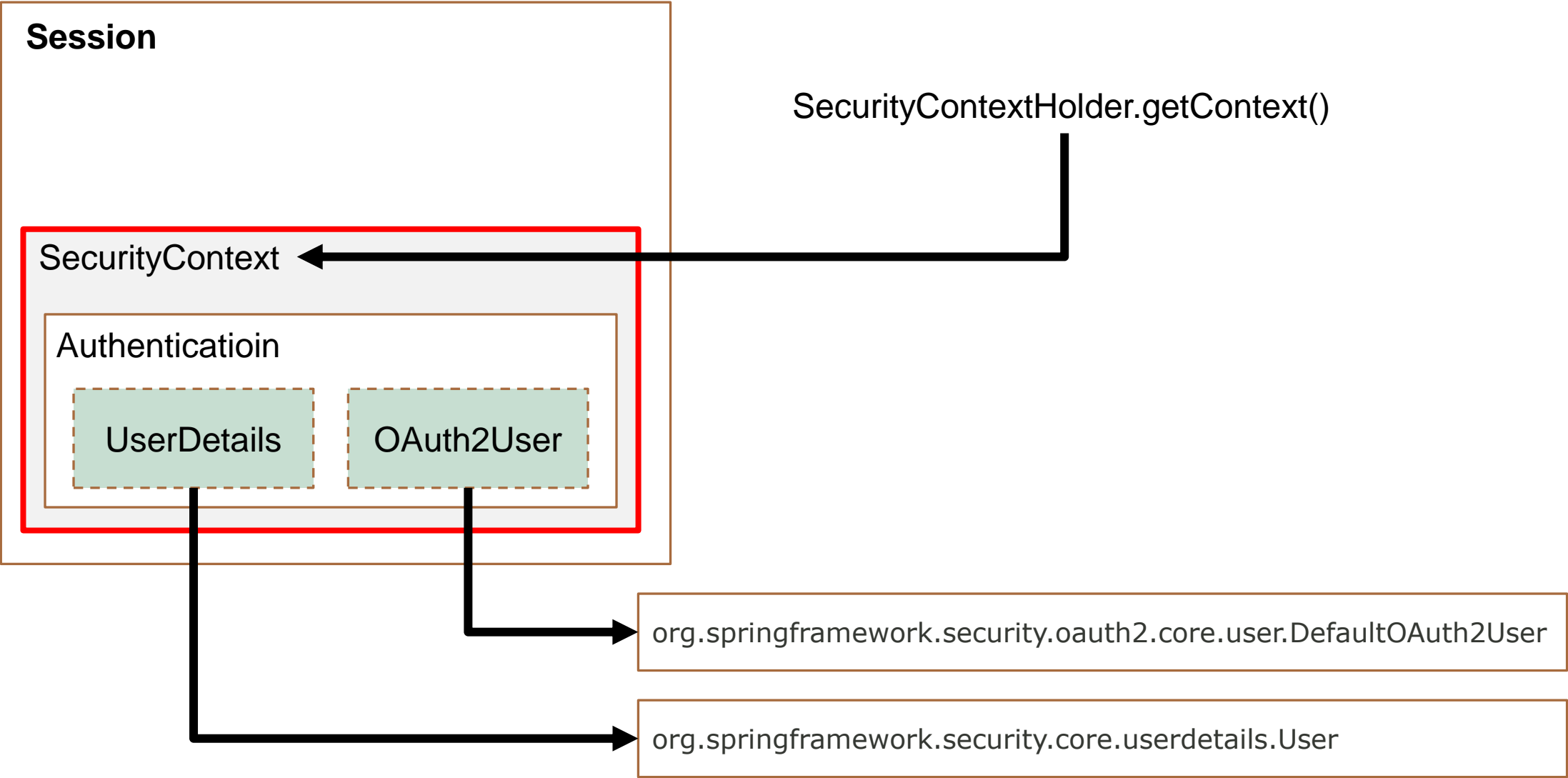
● final이 붙은 필드 변수 DI

● <http://localhost:8080/h2-console> 에서 DB에 입력된 값을 확인할 수 있다.

● 호출 시 프레임만 뜨고 실행이 안되면 SecurityConfig에 추가 : `http.headers(hr->hr.frameOptions(fo->fo.disable()));`

A.6. 추가

A.6.1 Authentication



A.6.2 로그인 세션 정보 확인 – 컨트롤러 Url 추가

LoginController.java

```
import org.springframework.security.core.userdetails.User;

@Controller
public class LoginController {
    :
    // 로그인 세션 정보 확인용 URL
    @GetMapping("/auth")
    public @ResponseBody ResponseEntity<?> auth(@AuthenticationPrincipal User user) {

        if (user == null) {
            return ResponseEntity.ok("로그인 상태가 아닙니다.");
        }
        return ResponseEntity.ok(user);
    }
}
```

A.6.3 회원 가입 기능을 추가해 봅시다.

LoginController.java

```
@Controller
public class LoginController {

    @Autowired
    private MemberService memberService;
    :

    @GetMapping ("/join")
    public void join() {}

    @PostMapping ("/join")
    public String joinProc(Member member) {
        memberService.save(member);
        return "welcome";
    }
}
```

MemberService.java

```
@Service
public class MemberService {

    @Autowired
    private MemberRepository memberRepo;

    public void save(Member member) {
        memberRepo.save(member);
    }
}
```

join.html : 가입 정보 입력 View

welcome.html : 가입 환영 View

B.Spring Boot Security with JWT

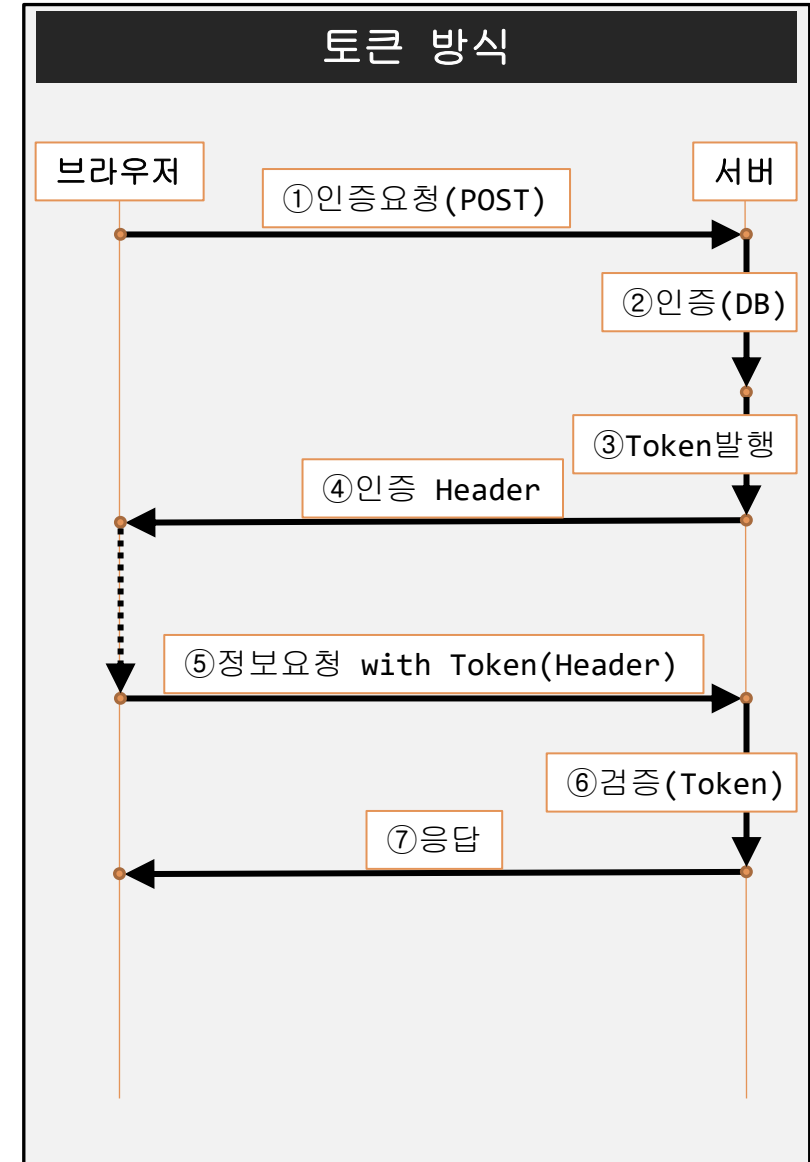
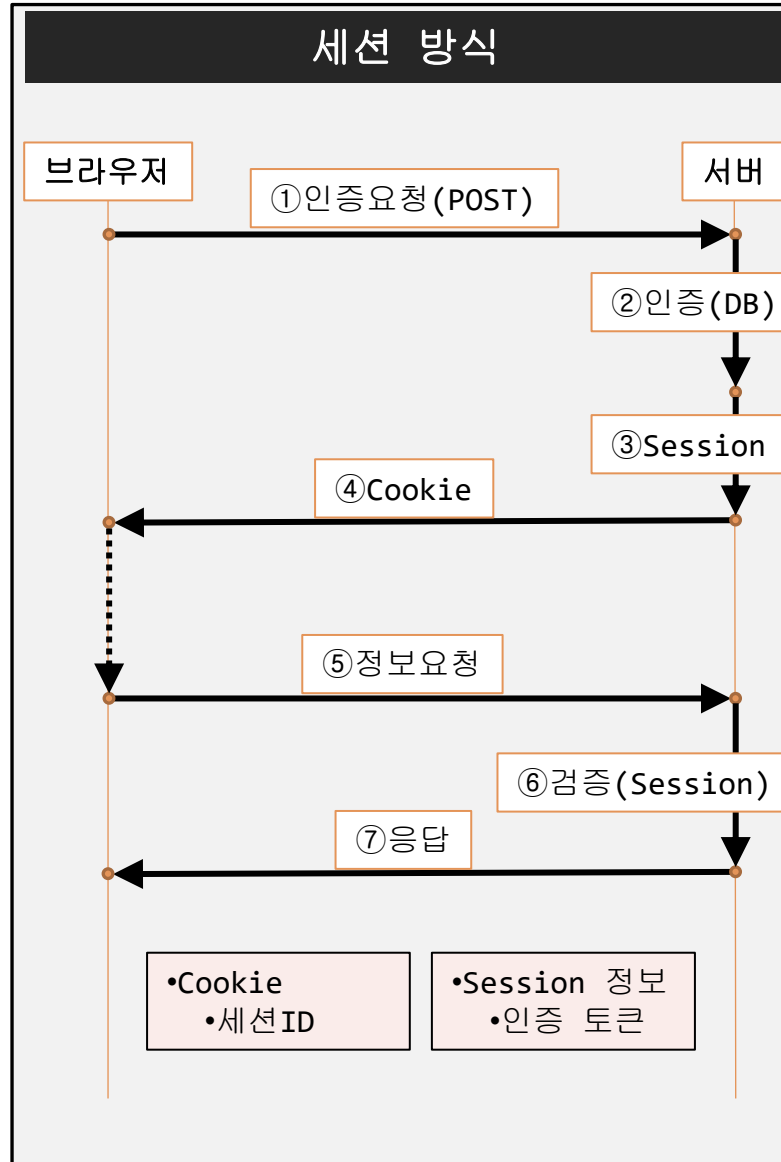
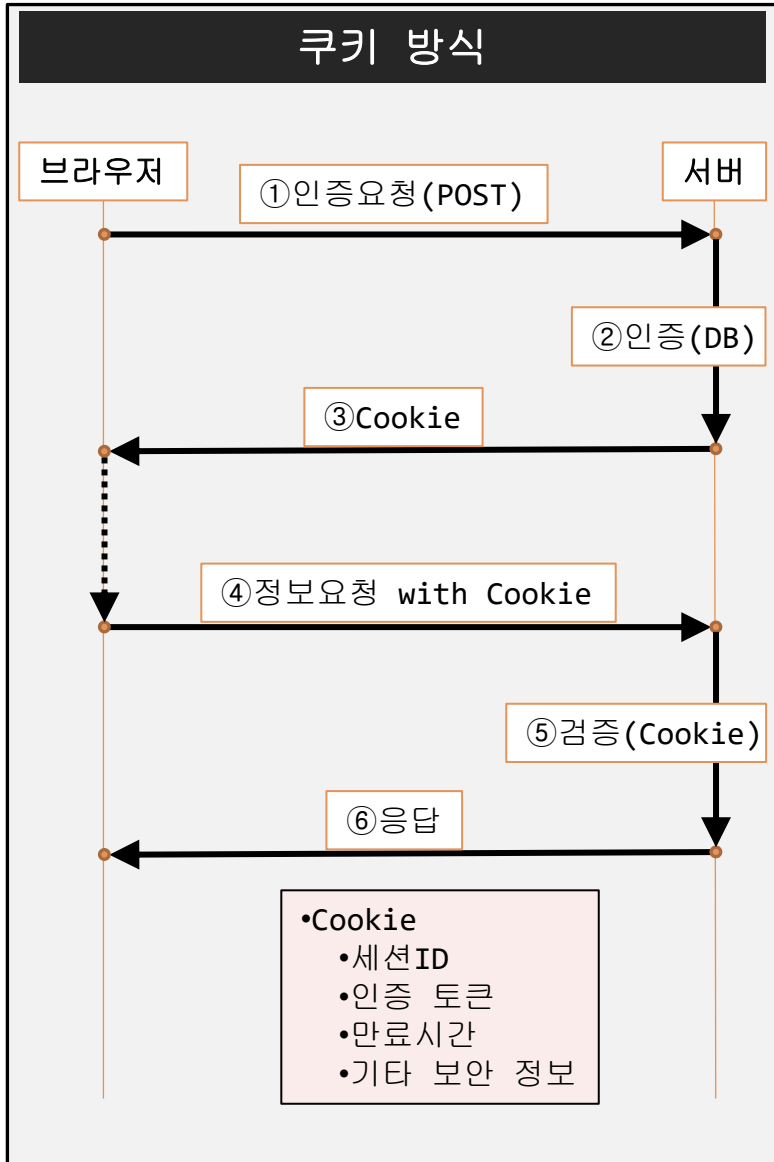
Spring-Framework 6.1.6

Spring Boot 3.2.5

Spring-Security 6.2.4

B.1. JWT

B.1.1 인증 방식 종류



B.1.2 인증 방식에 따른 장단점

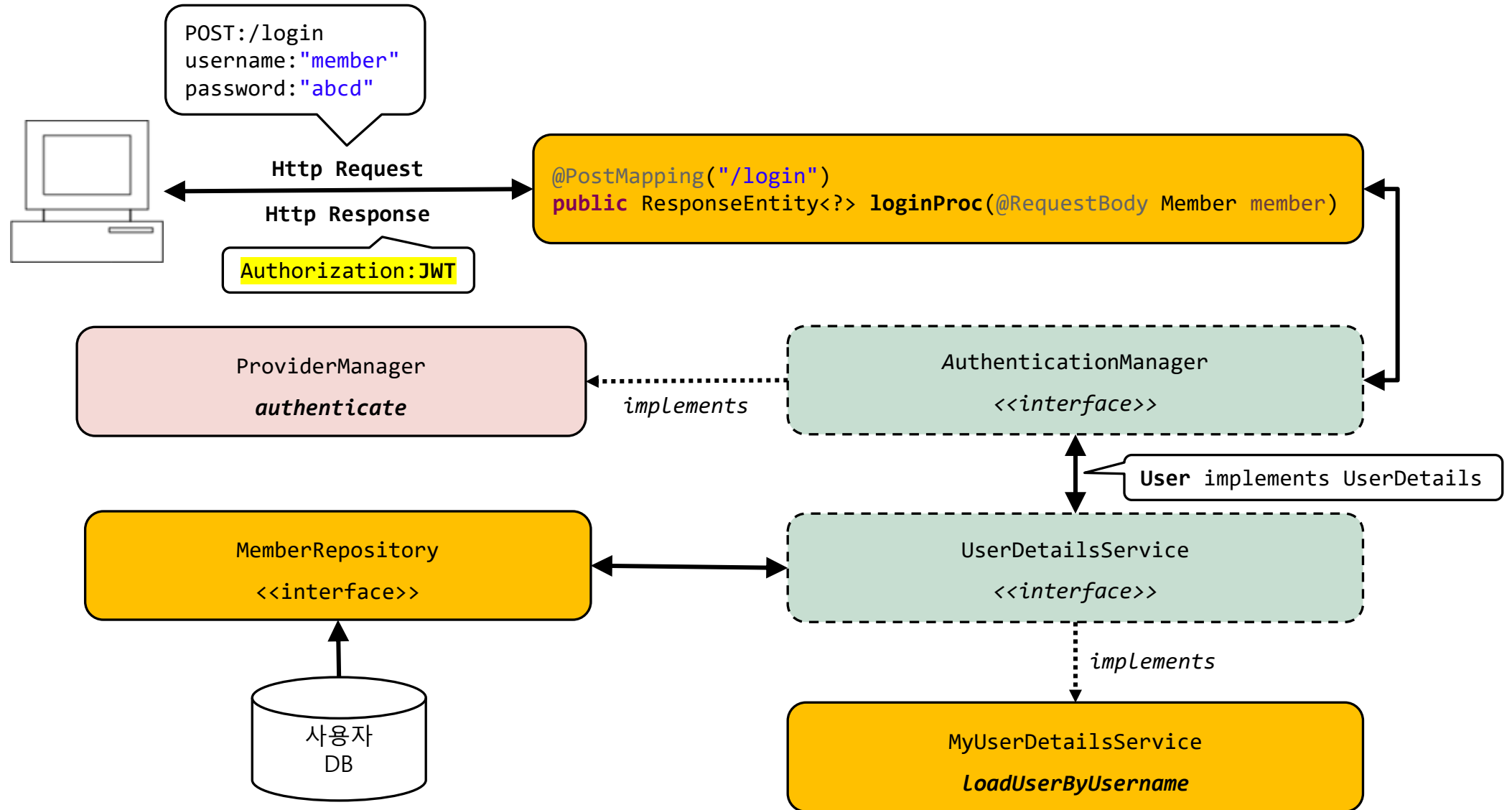
방식	장점	단점
쿠키	<ul style="list-style-type: none"> • 간단한 구현: 웹 브라우저가 자동으로 쿠키를 관리하므로 클라이언트와 서버 간 인증 정보를 쉽게 주고받을 수 있다. • 세션 관리와 통합 가능: 세션과 함께 사용할 수 있어 세션을 통해 추가적인 사용자 데이터를 저장하고 활용할 수 있다. 	<ul style="list-style-type: none"> • CSRF 공격 취약성: 추가적인 보안 조치가 필요하다. • 확장성 문제: 서버에 상태 정보를 저장하므로, 서버가 여러 대인 경우 세션 동기화가 필요해 관리가 복잡해질 수 있다.
세션	<ul style="list-style-type: none"> • 상태 유지: 서버에 사용자 상태 정보를 저장하므로 세션 동안 지속적인 사용자 상태 관리가 가능하다. • 서버 측 제어: 서버가 세션 데이터를 관리하므로 서버 측에서 세션 만료나 강제 로그아웃 등의 제어가 가능하다. 	<ul style="list-style-type: none"> • 확장성 문제: 사용자가 많아질수록 서버 메모리 사용량이 증가하며, 로드 밸런싱을 위해 세션 동기화가 필요하다. • 쿠키 의존성: 클라이언트가 세션 ID를 저장하고 서버로 보내기 위해 쿠키를 사용하므로, 쿠키 기반 인증의 단점을 공유한다.
토큰	<ul style="list-style-type: none"> • 무상태(stateless): 토큰 자체에 모든 인증 정보를 포함하므로 서버에 상태 정보를 저장할 필요가 없다. → 서버 확장성 • 편리한 클라이언트 측 저장: 토큰은 클라이언트 측에 저장할 수 있으며, 다양한 저장소(쿠키, 로컬 스토리지 등)에서 관리 가능 • 유연한 인증 정보 포함: 토큰 페이로드에 필요한 사용자 정보 및 권한 등 포함 가능 → 커스터마이징 가능 	<ul style="list-style-type: none"> • 보안 문제: 토큰이 노출되면 재사용될 수 있으므로, 토큰 보호를 위한 추가적인 보안 조치가 필요하다. (예: 토큰 만료 시간 설정, HTTPS 사용 등) • 크기 문제: 토큰의 크기가 크면 네트워크 비용이 증가할 수 있다. • 토큰 무효화: 한 번 발급된 토큰을 서버 측에서 무효화하는 것이 어렵다. 이를 해결하기 위해 별도의 블랙리스트나 리프레시 토큰 메커니즘이 필요할 수 있다.

B.1.3 JWT (JSON Web Token)

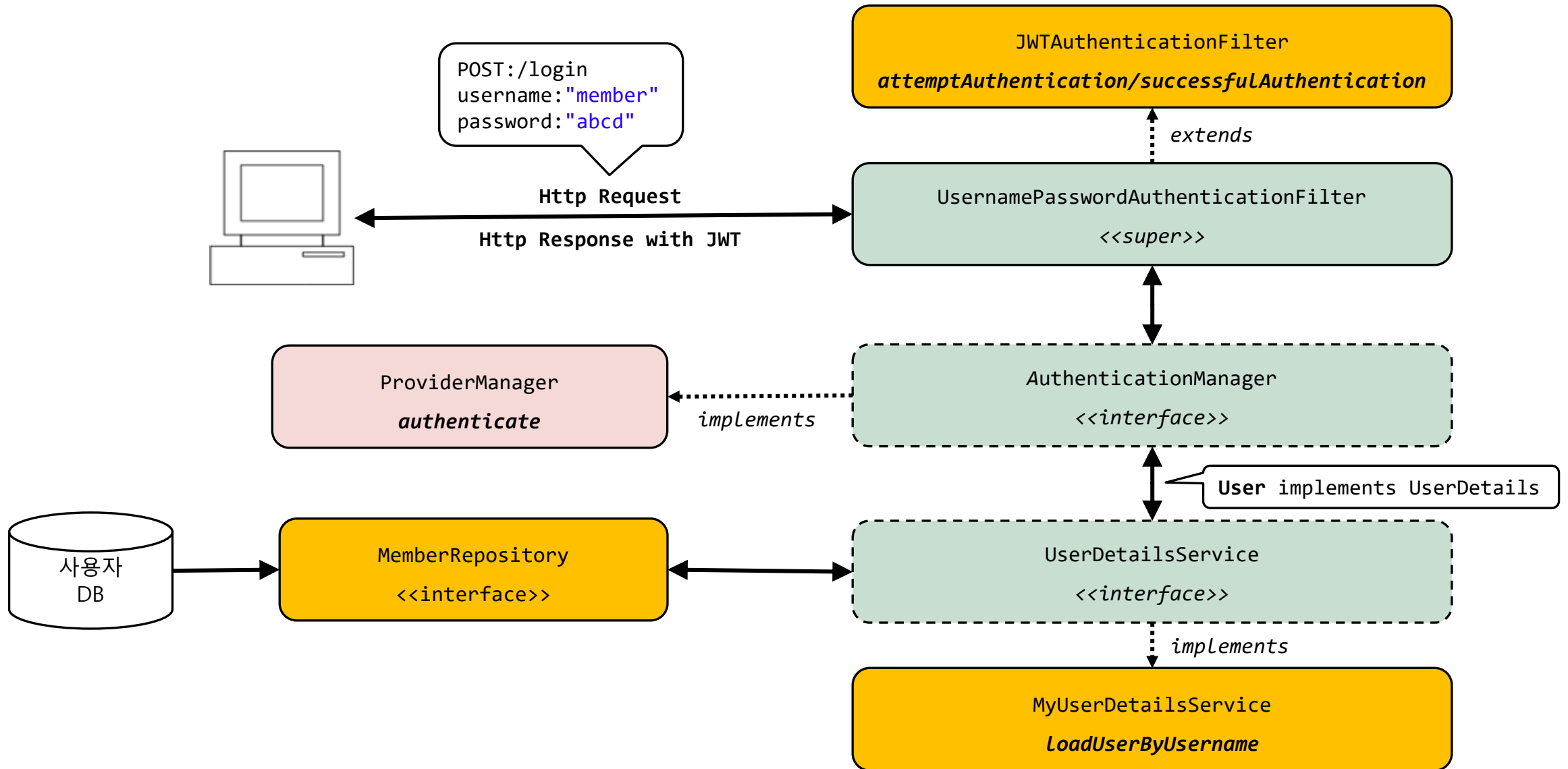
- 인증에 필요한 정보를 암호화한 JSON Token (<https://jwt.io>)
- 토큰 인증방식
- 토큰 구조 : Header + “.” + Payload + “.” + Signature

```
//Header (헤더)
{
  "alg": "HS256",           // 서명 암호와 알고리즘 ( ex: HMAC SHA256, RSA)
  "typ": "JWT",             // 토큰 유형
}
//Payload (내용)
{
  "jti" : "1234",            // Registered Claim
  "exp": "162143000000",     // Registered Claim
  "https://story.pxd.co.kr" : true, // Public Claim
  "username":"jun"          // Private Claim
}
//signature (서명)
HMACSHA256( base64UrlEncode(header) + "." + base64UrlEncode(payload), 사용자키) // 유일한 키 값
```

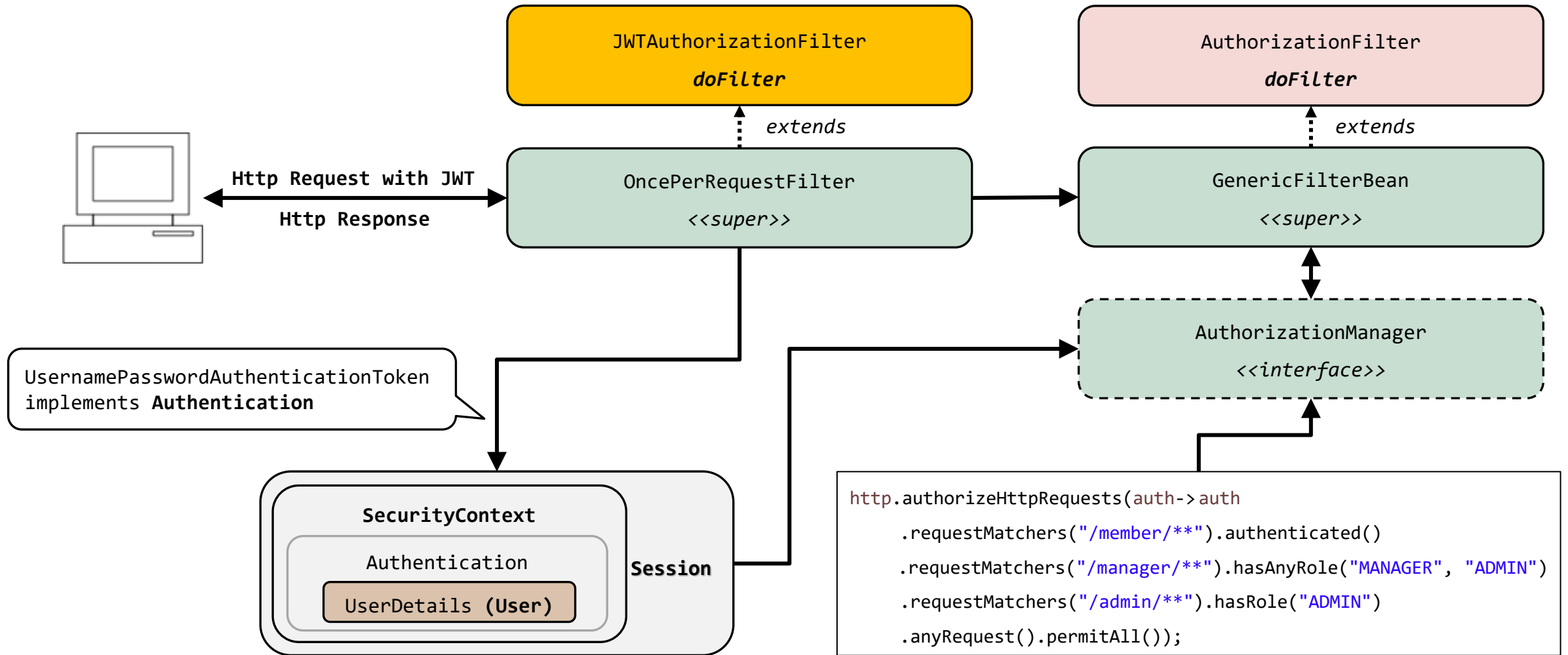
B.1.4 스프링 시큐리티 인증 절차 개략도 (토큰방식 – 컨트롤러 핸들러)



B.1.5 스프링 시큐리티 인증 절차 개략도 (토큰방식 - 필터)

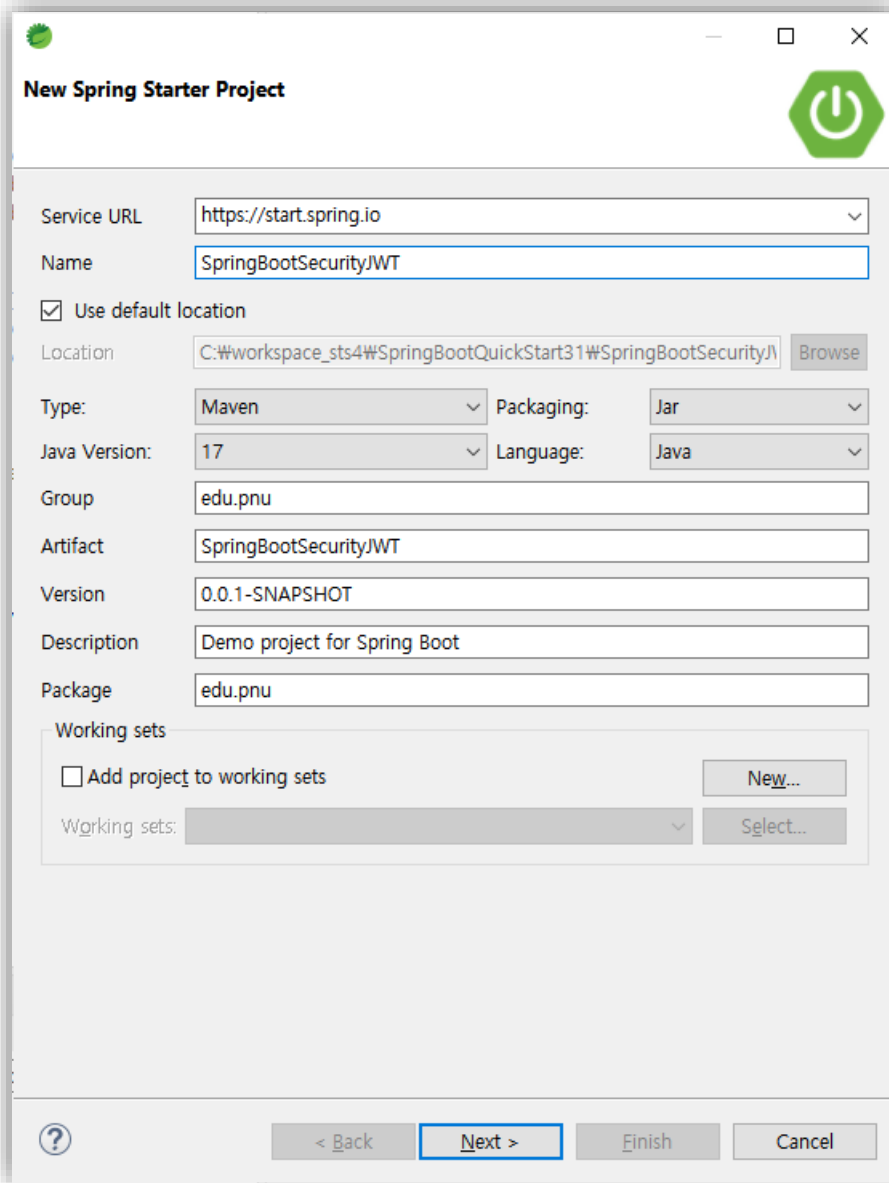


B.1.6 스프링 시큐리티 인가 절차 개략도 (토큰방식)



B.2. 프로젝트 시작

B.2.1 프로젝트 생성



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

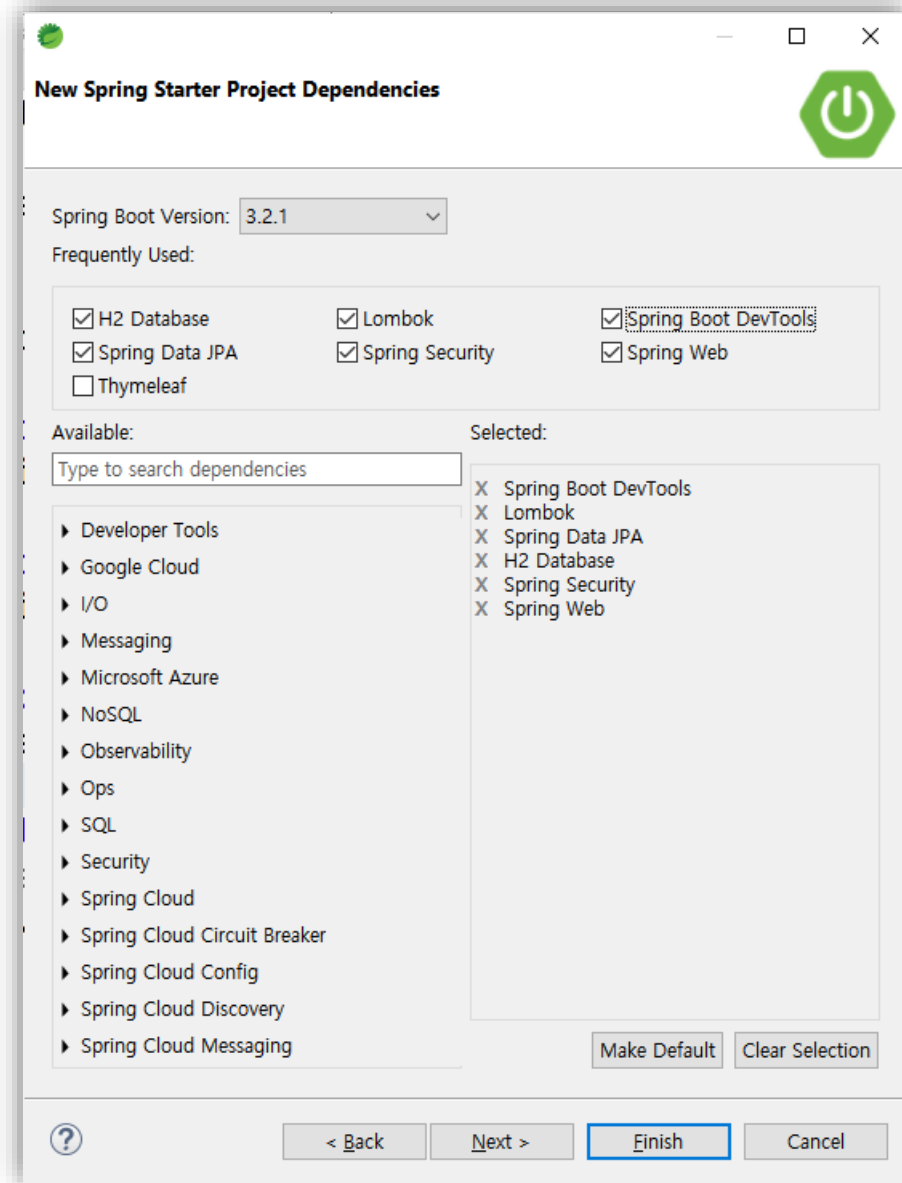
Description:

Package:

Working sets

☐ Add project to working sets

Working sets:



New Spring Starter Project Dependencies

Spring Boot Version:

Frequently Used:

☒ H2 Database ☒ Lombok ☒ Spring Boot DevTools

☒ Spring Data JPA ☒ Spring Security ☒ Spring Web

☐ Thymeleaf

Available:

- ▶ Developer Tools
- ▶ Google Cloud
- ▶ I/O
- ▶ Messaging
- ▶ Microsoft Azure
- ▶ NoSQL
- ▶ Observability
- ▶ Ops
- ▶ SQL
- ▶ Security
- ▶ Spring Cloud
- ▶ Spring Cloud Circuit Breaker
- ▶ Spring Cloud Config
- ▶ Spring Cloud Discovery
- ▶ Spring Cloud Messaging

Selected:

- X Spring Boot DevTools
- X Lombok
- X Spring Data JPA
- X H2 Database
- X Spring Security
- X Spring Web

B.2.2 엔티티 클래스, Repository, application.properties 작성

edu.pnu.domain.Role.java

```
public enum Role {  
    ROLE_ADMIN, ROLE_MANAGER, ROLE_MEMBER  
}
```

edu.pnu.domain.Member.java

```
@Getter  
@Setter  
@ToString  
@Builder  
@AllArgsConstructor  
@NoArgsConstructor  
@Entity  
public class Member {  
    @Id  
    private String username;  
    private String password;  
    @Enumerated(EnumType.STRING)  
    private Role role;  
    private boolean enabled;  
}
```

edu.pnu.persistence.MemberRepository.java

```
package edu.pnu.persistence;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import edu.pnu.domain.Member;  
  
public interface MemberRepository extends JpaRepository<Member, String> {  
}
```

src/main/resources/application.properties

```
spring.datasource.driver-class-name=org.h2.Driver  
spring.datasource.url=jdbc:h2:tcp://localhost/~/.h2/securityjwt  
spring.datasource.username=sa  
spring.datasource.password=  
  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
spring.jpa.hibernate.ddl-auto=create  
spring.jpa.show-sql=true
```

B.2.3 사용자 추가

- src/test/java에 데이터 입력을 위한 코드를 작성해서 Junit Test로 실행

```
@SpringBootTest
public class MemberInitialize {
    @Autowired
    MemberRepository memRepo;

    PasswordEncoder encoder = new BCryptPasswordEncoder();

    @Test
    public void doWork() {
        memRepo.save(Member.builder()
            .username("member")
            .password(encoder.encode("abcd"))
            .role(Role.ROLE_MEMBER)
            .enabled(true).build());
        memRepo.save(Member.builder()
            .username("manager")
            .password(encoder.encode("abcd"))
            .role(Role.ROLE_MANAGER)
            .enabled(true).build());
        memRepo.save(Member.builder()
            .username("admin")
            .password(encoder.encode("abcd"))
            .role(Role.ROLE_ADMIN)
            .enabled(true).build());
    }
}
```

Table Member CRUD 인터페이스

비밀번호 암호화 인터페이스/구현체

DB에 입력된 데이터 확인

실행	Run Selected	자동 완성	지우기	SQL 문:
SELECT * FROM MEMBER				
SELECT * FROM MEMBER;				
ENABLED	PASSWORD	ROLE	USERNAME	
TRUE	\$2a\$10\$bXu6NuTFyl5fHqGM1du4.9p1Hh1MGSNTJPha5EdSiw542JwugHLu	ROLE_MEMBER	member	
TRUE	\$2a\$10\$uM5j3BhlvRwskhtD39L9ouPhWwHnozE6jOanjdhNOgMtP5g60TTKG	ROLE_MANAGER	manager	
TRUE	\$2a\$10\$ourWx/TFvqdGz1FXPIUDp.ksCkhhTy0Uz5Wv549VPBg9tUzgqnmSG	ROLE_ADMIN	admin	
(3 행, 3 ms)				

B.2.4 시큐리티 컨트롤러 요청 url 추가

서버 실행 후 테스트

edu.pnu.controller.SecurityController.java

```
@RestController  
  
public class SecurityController {  
  
    @GetMapping("/")           public String index()      { return "index";    }  
  
    @GetMapping("/member")      public String member()   { return "member";   }  
  
    @GetMapping("/manager")     public String manager()  { return "manager";  }  
  
    @GetMapping("/admin")       public String admin()     { return "admin";    }  
  
}
```

➔ 서버를 실행하고 로그인 한 뒤 **Url**들이 제대로 호출되는지 확인

B.3. 시큐리티 인증 및 JWT 발행

B.3.1 시큐리티 설정 파일 작성

edu.pnu.config.SecurityConfig.java

```
@Configuration
@EnableWebSecurity

public class SecurityConfig {

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

    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        : ➡ 다음 페이지 코딩 계속
        return http.build();
    }
}
```

B.3.2 시큐리티 설정 파일 작성

edu.pnu.config.SecurityConfig.java

@Bean

```
SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
    http.csrf(csrf->csrf.disable()); // CSRF 보호 비활성화  
  
    http.authorizeHttpRequests(auth->auth  
        .requestMatchers("/member/**").authenticated()  
        .requestMatchers("/manager/**").hasAnyRole("MANAGER", "ADMIN")  
        .requestMatchers("/admin/**").hasRole("ADMIN")  
        .anyRequest().permitAll());  
  
    http.formLogin(frmLogin->frmLogin.disable()); // Form을 이용한 로그인을 사용하지 않겠다는 설정  
    http.httpBasic(basic->basic.disable()); // Http Basic인증 방식을 사용하지 않겠다는 설정  
  
    // 세션을 유지하지 않겠다고 설정 → Url 호출 뒤 응답할 때까지는 유지되지만 응답 후 삭제된다는 의미.  
    http.sessionManagement(sm->sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS));  
  
    return http.build();  
}
```

1. 서버가 정상적으로 실행되는지 확인.
2. localhost:8080 호출하면 index가 출력되는지 확인
3. localhost:8080/login 호출하면 404 에러가 발생하는지 확인
4. localhost:8080/member 호출하면 403 에러가 발생하는지 확인

B.3.3 시큐리티 인증 필터(1)

edu.pnu.config.filter.JWTAuthenticationFilter.java

```
@RequiredArgsConstructor
public class JWTAuthenticationFilter extends UsernamePasswordAuthenticationFilter {

    // 인증 객체
    private final AuthenticationManager authenticationManager;

    // POST/login 요청이 왔을 때 인증을 시도하는 메소드
    @Override
    public Authentication attemptAuthentication(HttpServletRequest request,
                                                HttpServletResponse response) throws AuthenticationException {

        ~~ 55page 코딩 ~~

    }

    // 인증이 성공했을 때 실행되는 후처리 메소드
    @Override
    protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response,
                                            FilterChain chain, Authentication authResult) throws IOException, ServletException {

        ~~ 57page 코딩 ~~

    }
}
```

1. POST /login 요청이 들어오면 이 필터가 실행

B.3.3 시큐리티 인증 필터(2)

edu.pnu.config.filter.JWTAuthenticationFilter.java

```
public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response)
    throws AuthenticationException {

    // request에서 json 타입의 [username/password]를 읽어서 Member 객체를 생성한다.
    ObjectMapper mapper = new ObjectMapper();

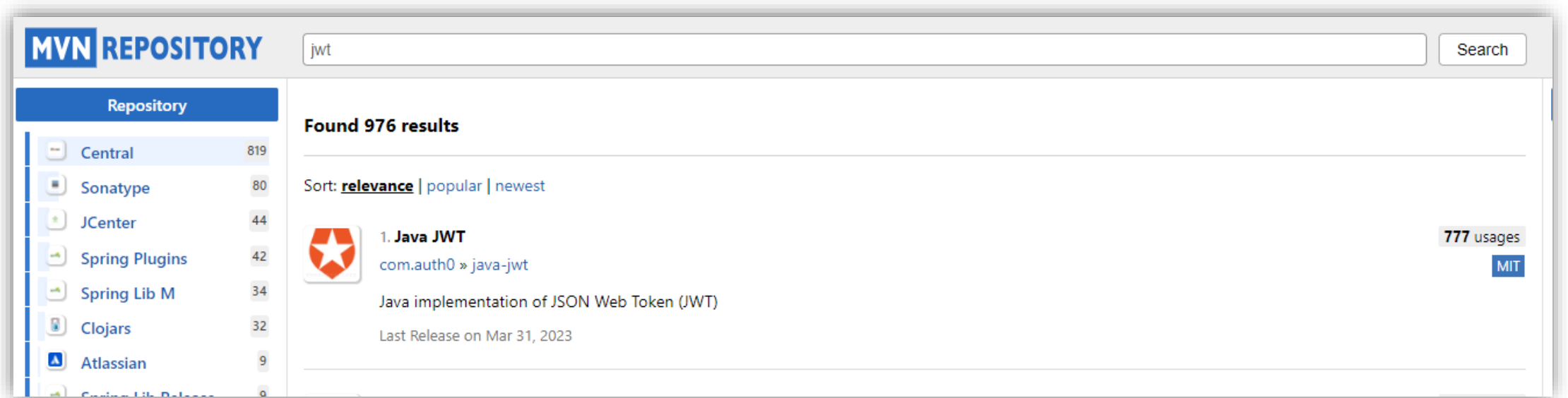
    try {
        Member member = mapper.readValue(request.getInputStream(), Member.class);
        // Security에게 자격 증명 요청에 필요한 객체 생성
        Authentication authToken = new UsernamePasswordAuthenticationToken(
            member.getUsername(), member.getPassword());

        // 인증 진행 -> UserDetailsService의 loadUserByUsername에서 DB로부터 사용자 정보를 읽어온 뒤
        // 사용자 입력 정보와 비교한 뒤 자격 증명에 성공하면 Authenticaiton객체를 만들어서 리턴한다.
        return auth = authenticationManager.authenticate(authToken);
    } catch (Exception e) {
        Log.info(e.getMessage()); // “자격 증명에 실패하였습니다.” 로그 출력
    }
    response.setStatus(HttpStatus.UNAUTHORIZED.value()); // 자격 증명에 실패하면 응답코드 리턴
    return null;
}
```

B.3.4 JWT 라이브러리 설정

pom.xml

```
<!-- https://mvnrepository.com/artifact/com.auth0/java-jwt -->
<dependency>
  <groupId>com.auth0</groupId>
  <artifactId>java-jwt</artifactId>
  <version>4.4.0</version>
</dependency>
```



B.3.5 시큐리티 인증 필터(3)

edu.pnu.config.filter.JWTAuthenticationFilter.java

```
protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response, FilterChain
chain, Authentication authResult) throws IOException, ServletException {

    // 자격 증명이 성공하면 loadUserByUsername에서 만든 객체가 authResult에 담겨져 있다.
    User user = (User)authResult.getPrincipal();

    // username으로 JWT를 생성해서 Response Header - Authorization에 담아서 돌려준다.
    // 이것은 하나의 예시로서 필요에 따라 추가 정보를 담을 수 있다.
    String token = JWT.create()
        .withExpiresAt(new Date(System.currentTimeMillis()+1000*60*10))
        .withClaim("username", user.getUsername())
        .sign(Algorithm.HMAC256("edu.pnu.jwt"));
    response.addHeader(HttpHeaders.AUTHORIZATION, "Bearer " + token);
    response.setStatus(HttpStatus.OK.value());
}
```

B.3.6 시큐리티 인증 필터 등록

edu.pnu.config.SecurityConfig.java

```
public class SecurityConfig {  
  
    ~ 생략 ~  
  
    @Autowired  
    private AuthenticationConfiguration authenticationConfiguration;  
  
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
  
        ~ 생략 ~  
  
        // 스프링 시큐리티가 등록한 필터체인에 뒤에 작성한 필터를 추가한다.  
        http.addFilter(new JWTAuthenticationFilter(  
            authenticationConfiguration.getAuthenticationManager()  
        ));  
  
        return http.build();  
    }  
}
```

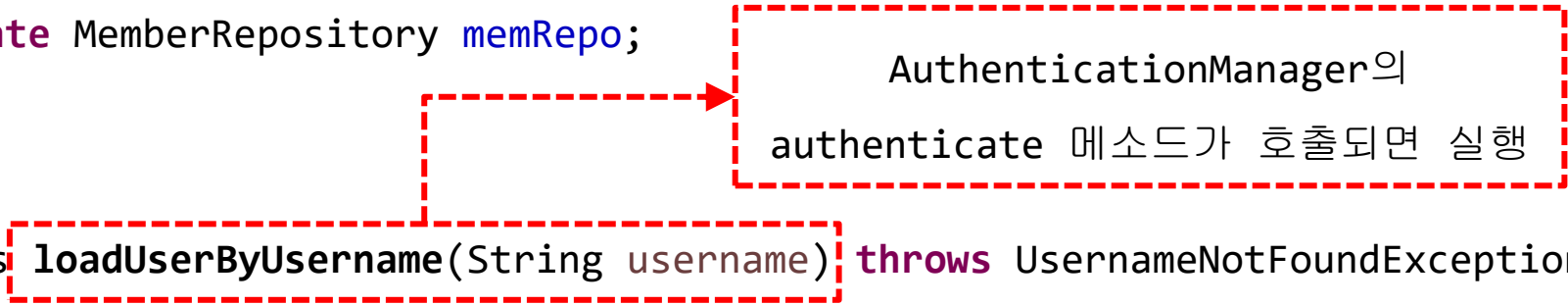
B.3.7 인증 서비스

edu.pnu.service.SecurityUserDetailsService.java

```
@Service
public class SecurityUserDetailsService implements UserDetailsService {

    @Autowired private MemberRepository memRepo;

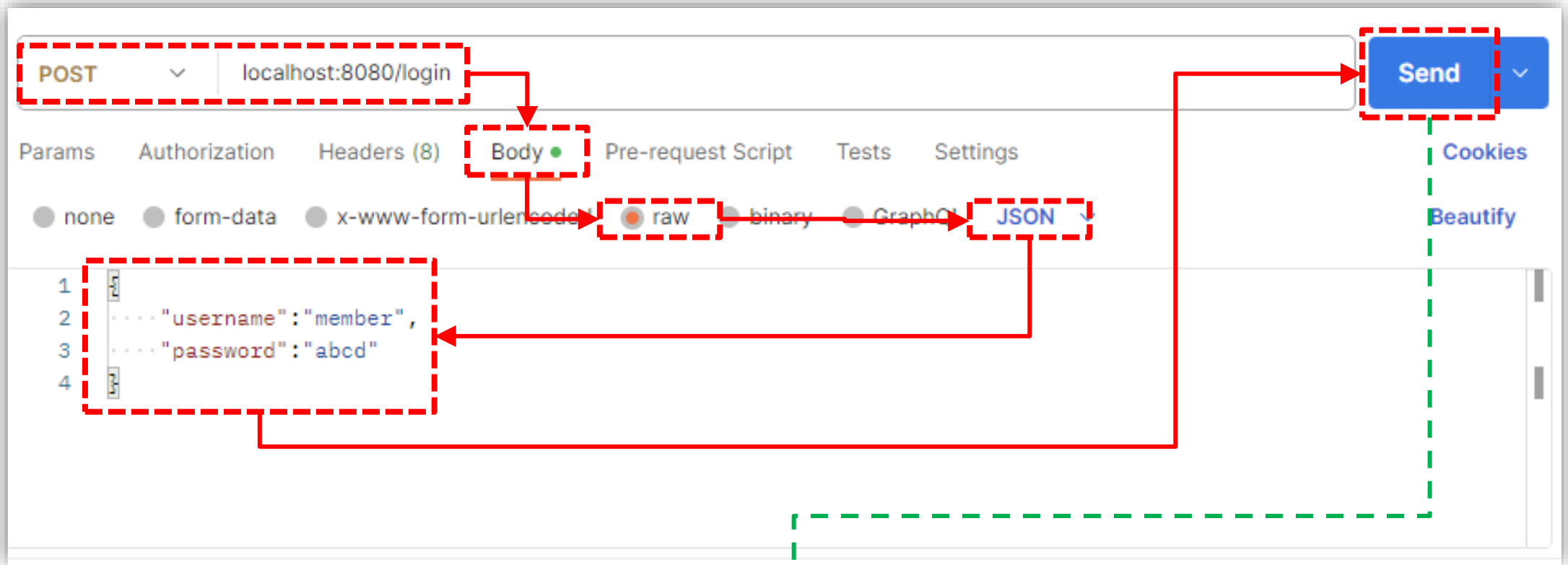
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException
    {
        Member member = memRepo.findById(username)
                                .orElseThrow(() -> new UsernameNotFoundException("Not Found!"));
        return new User(member.getUsername(), member.getPassword(),
                        AuthorityUtils.createAuthorityList(member.getRole().toString()));
    }
}
```



AuthenticationManager의
authenticate 메소드가 호출되면 실행

B.3.8 PostMan 테스트

서버 실행 후 테스트



Console 창에 아래와 같이 출력되면 성공

```
auth:UsernamePasswordAuthenticationToken [Principal=org.springframework.security.core.userdetails.User [Username=member, Password=[PROTECTED], Enabled=true, AccountNonExpired=true, credentialsNonExpired=true, AccountNonLocked=true, Granted Authorities=[ROLE_MEMBER]], Credentials=[PROTECTED], Authenticated=true, Details=null, Granted Authorities=[ROLE_MEMBER]]
```

B.3.9 PostMan 테스트

서버 실행 후 테스트

The screenshot shows the Postman interface for a POST request to `localhost:8080/login`. The request body is a JSON object: `{ "username": "member", "password": "abcd" }`. The request is configured with the `raw` body type and `JSON` format. The `Send` button is highlighted. The response is shown in the bottom panel, indicating a `200 OK` status. The response headers are listed in a table below.

Key	Value
Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOiJlE2OTg2MzM2OTQsInVzZ...
X-Content-Type-Options	nosniff
X-XSS-Protection	0
Cache-Control	no-cache, no-store, max-age=0, must-revalidate

B.4. JWT 인가 필터

B.4.1 시큐리티 인가 필터(1)

edu.pnu.config.filter.JWTAuthorizationFilter.java

```
@RequiredArgsConstructor
public class JWTAuthorizationFilter extends OncePerRequestFilter {
    // 인가 설정을 위해 사용자의 Role 정보를 읽어 들이기 위한 객체 설정
    private final MemberRepository memberRepository;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse
        response, FilterChain filterChain) throws IOException, ServletException {

        // ~~ 다음 페이지 ~~

        filterChain.doFilter(request, response);
    }
}

// OncePerRequestFilter 를 상속받게 되면 하나의 요청에 대해서 단 한번만 필터를 거치게 된다.
// 예를 들어 forwarding 되어 다른 페이지로 이동하게 되더라도 다시 이 필터를 거치지 않게 한다.
```

B.4.2 시큐리티 인가 필터(2)

edu.pnu.config.filter. JWTAuthorizationFilter.java

@Override

```
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) ... {  
    String srcToken = request.getHeader("Authorization");           // 요청 헤더에서 Authorization을 얻어온다.  
    if (srcToken == null || !srcToken.startsWith("Bearer ")) {      // 없거나 "Bearer "로 시작하지 않는다면  
        filterChain.doFilter(request, response);                  // 필터를 그냥 통과  
        return;  
    }  
    String jwtToken = srcToken.replace("Bearer ", "");              // 토큰에서 "Bearer "를 제거  
  
    // 토큰에서 username 추출  
    String username = JWT.require(Algorithm.HMAC256("edu.pnu.jwt")).build().verify(jwtToken).getClaim("username").asString();  
  
    Optional<Member> opt = memberRepository.findById(username);      // 토큰에서 얻은 username으로 DB를 검색해서 사용자를 검색  
    if (!opt.isPresent()) {                                          // 사용자가 존재하지 않는다면  
        filterChain.doFilter(request, response);                  // 필터를 그냥 통과  
        return;  
    }  
    ~ 계속 ~  
}
```

B.4.3 시큐리티 인가 필터(3)

edu.pnu.config.filter. JWTAuthorizationFilter.java

~ 계속 ~

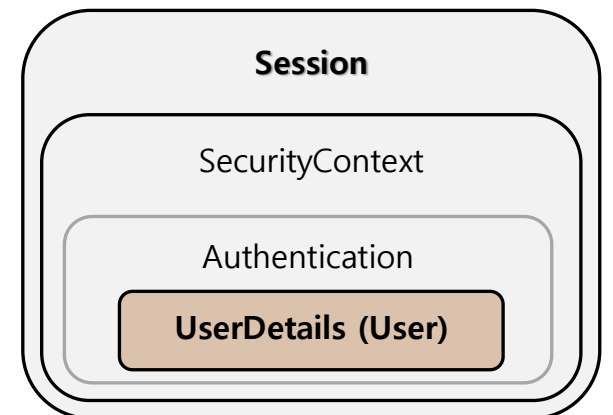
```
Member findmember = opt.get();
```

```
// DB에서 읽은 사용자 정보를 이용해서 UserDetails 타입의 객체를 생성
User user = new User(findmember.getUsername(), findmember.getPassword(),
    AuthorityUtils.createAuthorityList(findmember.getRole().toString()));
```

```
// Authentication 객체를 생성 : 사용자명과 권한 관리를 위한 정보를 입력(암호는 필요 없음)
Authentication auth = new UsernamePasswordAuthenticationToken(user, null, user.getAuthorities());
```

```
// 시큐리티 세션에 등록한다.
SecurityContextHolder.getContext().setAuthentication(auth);
```

```
filterChain.doFilter(request, response);
}
```



B.4.4 시큐리티 인가 필터 등록

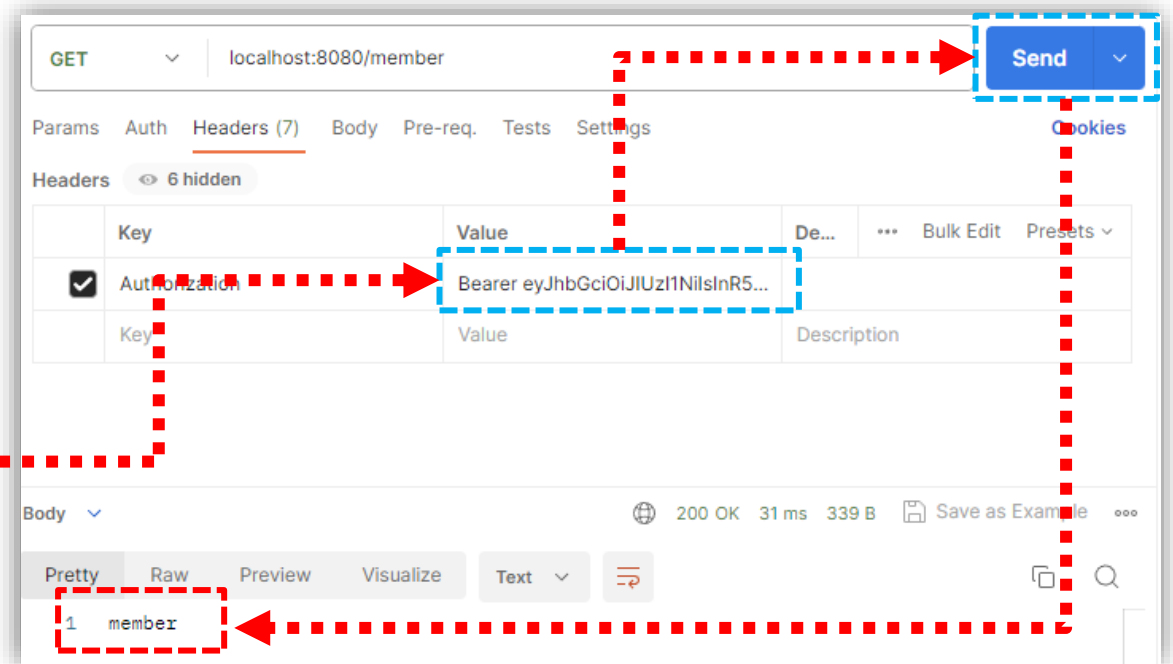
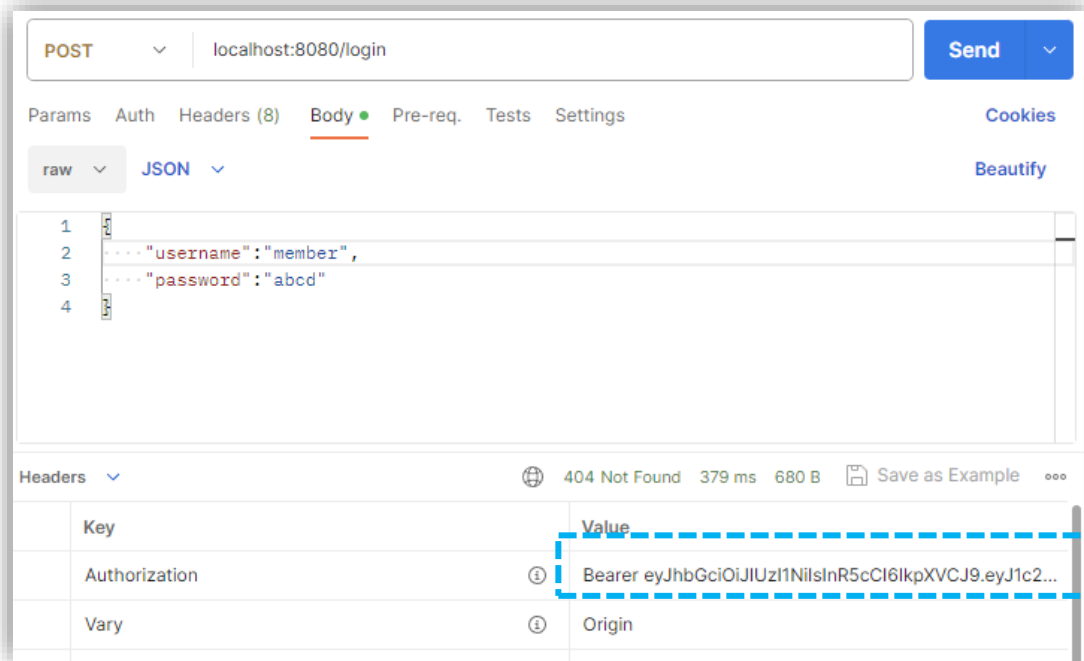
edu.pnu.config.SecurityConfig.java

```
public class SecurityConfig {  
  
    @Autowired  
    private MemberRepository memberRepository;  
  
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
  
        ~ 생략 ~  
  
        // 스프링 시큐리티가 등록한 필터들 중에서 AuthorizationFilter 앞에 앞에서 작성한 필터를 삽입한다.  
        http.addFilterBefore(new JWTAuthorizationFilter(memberRepository), AuthorizationFilter.class);  
  
        return http.build();  
    }  
}
```

B.4.5 시큐리티 인가 필터 테스트

서버 실행 후 테스트

- 서버 구동 – PostMan 로그인 후 응답 JWT를 이용해서 다른 URL 접근 테스트



- member로 로그인한 뒤에 /manager를 요청하면 아래와 같은 응답을 받게 된다.
- {"timestamp":"2023-06-30T12:59:31.620+00:00","status":403,"error":"Forbidden","path":"/manager"}

C.Spring Boot Security with OAuth2

Spring-Framework 6.0.13

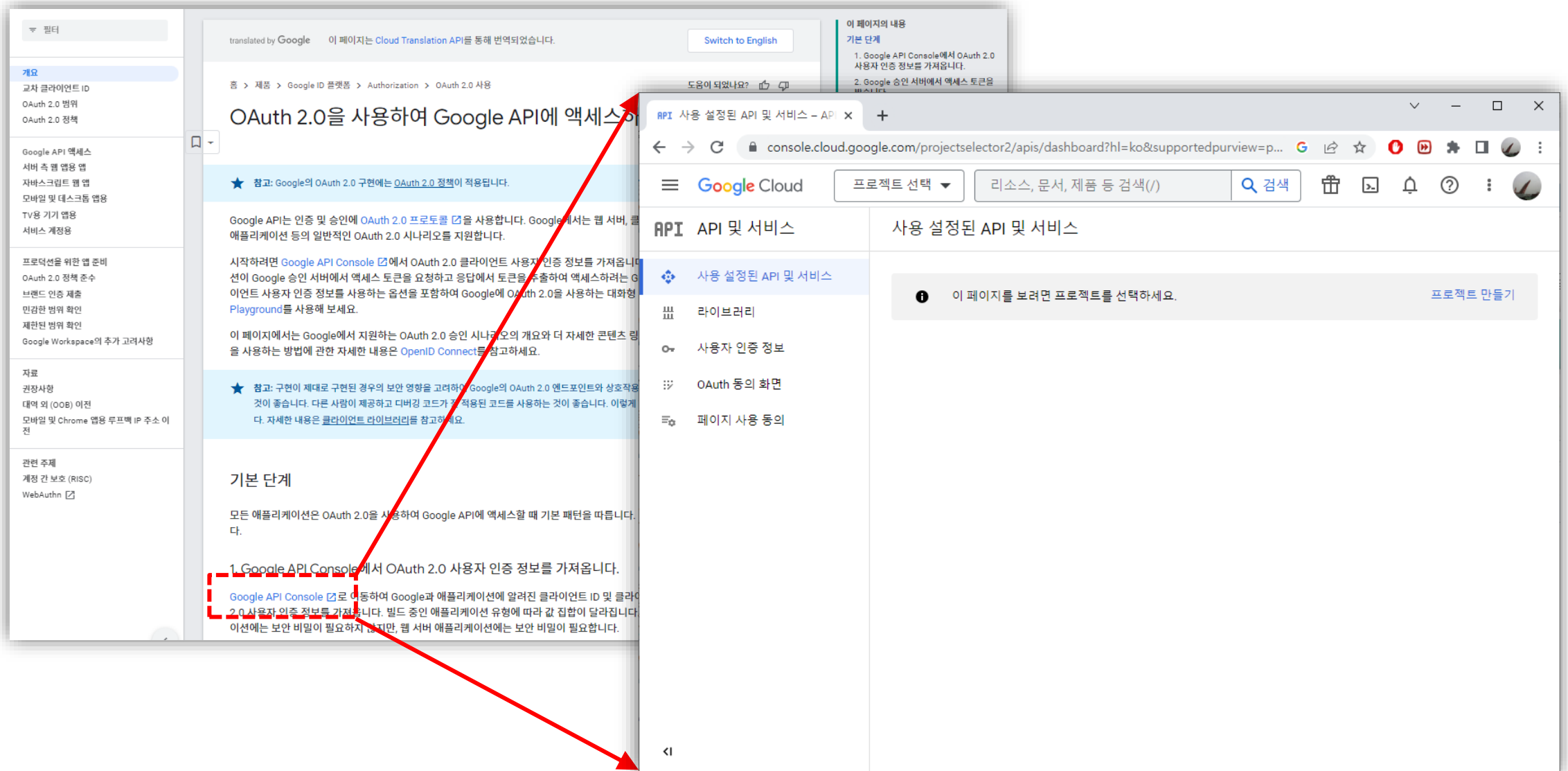
Spring Boot 3.1.5

Spring-Security 6.1.5

C.1 OAuth2 with Google

C.1.1 설정 준비

- <https://developers.google.com/identity/protocols/oauth2> → 구글 검색창에서 “google oauth2 api” 검색

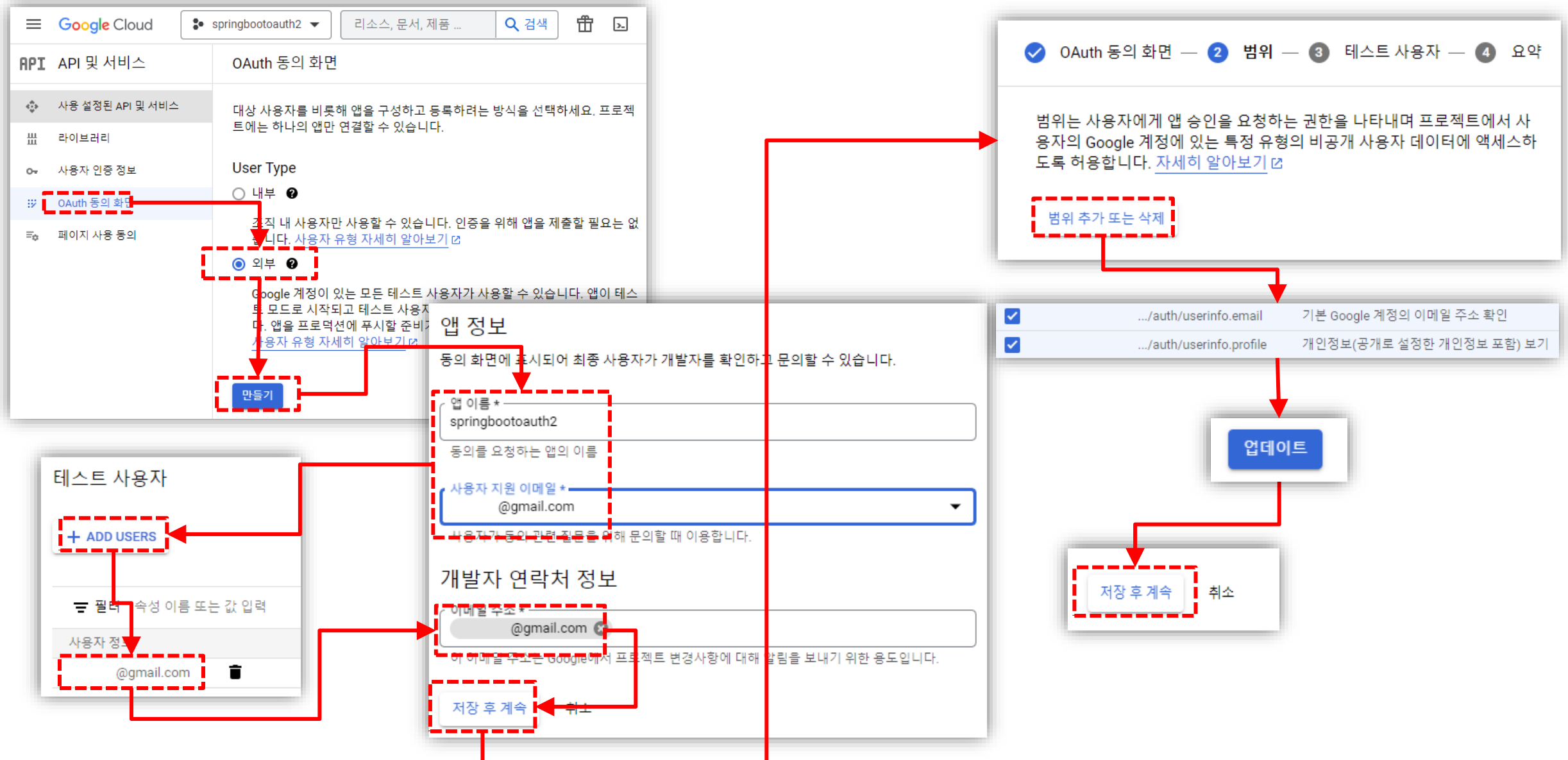


C.1.2 프로젝트 생성

The image illustrates the process of creating a new project in the Google Cloud console through four sequential screenshots:

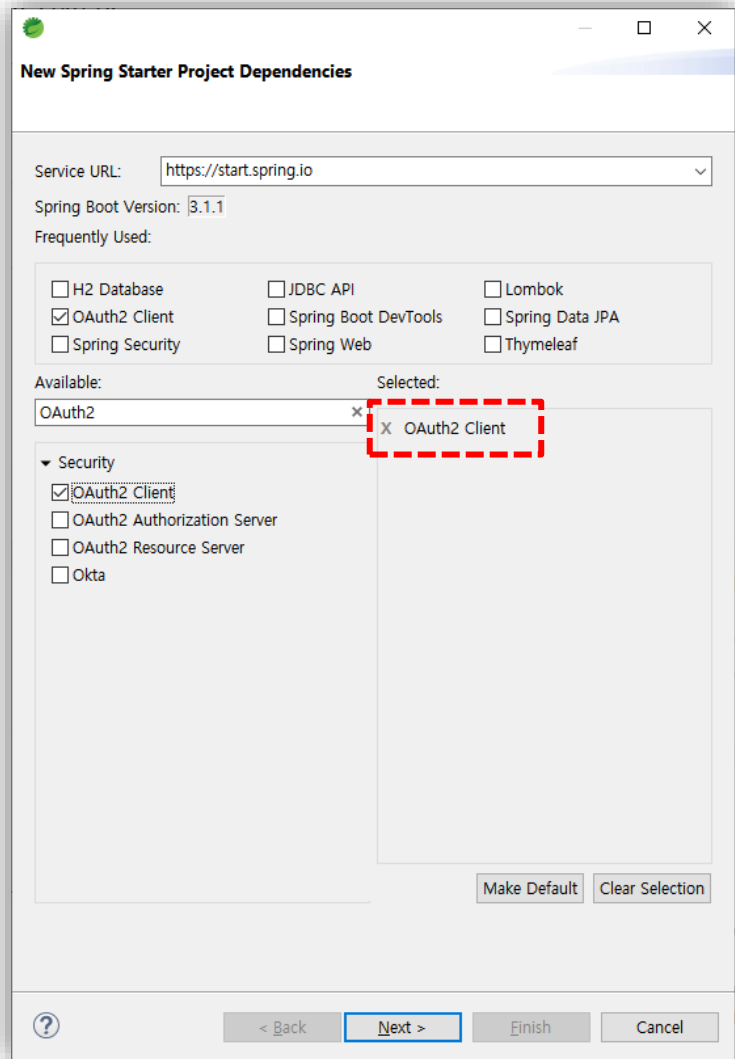
- Top Left Screenshot:** The 'API 및 서비스' (APIs & Services) page. A red dashed box highlights the '프로젝트 선택' (Select Project) dropdown menu in the top navigation bar.
- Top Right Screenshot:** The '프로젝트 선택' (Select Project) modal. A red dashed box highlights the '새 프로젝트' (New Project) button.
- Bottom Left Screenshot:** The 'API 및 서비스' page with a specific project selected (springbooth2). The 'API 및 서비스' list is visible, and a red arrow points from the '새 프로젝트' button in the previous step to this screenshot.
- Bottom Right Screenshot:** The '새 프로젝트' (New Project) dialog. A red dashed box highlights the '프로젝트 이름' (Project Name) field, which contains the text 'springbooth2'. A red arrow points from the '새 프로젝트' button in the previous step to this dialog.

C.1.3 OAuth 동의 화면



C.1.5 OAuth2 Client Dependency 추가 및 설정

Add Starters



application.properties

```
spring.security.oauth2.client.registration.google.client-id=*****  
spring.security.oauth2.client.registration.google.client-secret=*****  
spring.security.oauth2.client.registration.google.scope=email,profile
```

→ 앞에서 다운받아 둔 JSON 파일에서 client-id와 client-secret를 찾아 입력

login.html

```
<body th:align="center">  
    <h2>로그인</h2>  
    <form action="/login" method="POST">  
        <table th:align="center" border="1" th:cellpadding="0">  
            :  
        </table>  
    </form>  
    <a href="http://localhost:8080/oauth2/authorization/google">구글 로그인</a>  
</body>
```

OAuth2 Client 사용 시 URL은 고정

Front-End 로그인 창에 [구글 로그인] 링크 추가 → 호스트 주소(<http://localhost:8080>)는 Front-End 와 스프링 부트 서버의 주소가 동일하면 생략가능 ("/oauth2/authorization/google")

C.1.6 OAuth2 with Session

- Session을 유지하는 경우의 OAuth2 로그인

SecurityConfig.java

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

~ 생략 ~

```
// 구글 로그인을 실행하면 DefaultOAuth2UserService가 실행됨.
```

```
// 로그인에 성공했을 때 추가적인 작업이 필요하면 DefaultOAuth2UserService를 상속한 클래스의
```

```
// loadUser 메소드에서 하면 됨.
```

```
http.oauth2Login(oauth2->oauth2
```

```
    .loginPage("/login")
```

```
    .defaultSuccessUrl("/loginSuccess", true);
```

```
});
```

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

```
}
```

[구글 로그인]링크가 있는 로그인 페이지 설정. 생략하면 스프링 시큐리티가 제공하는 OAuth2 로그인 화면이 뜬다.

수정이 완료되면 제대로 로그인이 되는지 서버 구동 및 테스트

C.1.7 OAuth2 로그인 세션 정보 확인 – 컨트롤러에 URL 추가

LoginController.java

```
import org.springframework.security.core.userdetails.User;

@Controller
public class LoginController {

    // 로그인 세션 정보 확인용 URL
    @GetMapping("/oauth")
    public @ResponseBody String auth(@AuthenticationPrincipal OAuth2User user) {
        if (user == null) return "OAuth2:null";

        // 자동 회원가입을 한다면 이용할 정보 확인
        System.out.println("attributes:" + user.getAttributes());

        return "OAuth2:" + user;
    }
}
```

C.1.8 OAuth2 without Session(1)

- Session 생성 정책이 stateless인 경우의 OAuth2 로그인

SecurityConfig.java

```
@RequiredArgsConstructor
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    private final OAuth2SuccessHandler successHandler;

    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

        ~ 생략 ~

        http.oauth2Login(oauth2->oauth2.successHandler(successHandler));

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

로그인에 성공하면 임의의 사용자를 생성해서 DB에 저장하고 JWT 토큰을 만들어서 응답 헤더에 설정하는 핸들러

OAuth2 로그인이 성공하면 실행되는 successHandler, defaultSuccessUrl 두 메소드가 동시에 설정되어 있으면 successHandler가 우선한다.

C.1.9 OAuth2 without Session(2)

OAuth2SuccessHandler.java

```
@Slf4j
@RequiredArgsConstructor
@Component
public class OAuth2SuccessHandler extends SimpleUrlAuthenticationSuccessHandler {
    private final MemberRepository memRepo;
    private final PasswordEncoder encoder;
    @Override
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response,
                                       Authentication authentication) throws IOException, ServletException {
        Log.info("OAuth2SuccessHandler:onAuthenticationSuccess");
        OAuth2User user = (OAuth2User)authentication.getPrincipal();
        // 임의의 사용자를 만들어서 서버에 저장
        String username = CustomMyUtil.getUsernameFromOAuth2User(user);
        if (username == null) {
            Log.error("onAuthenticationSuccess:Cannot generate username from oauth2user!");
            throw new ServletException("Cannot generate username from oauth2user!");
        }
        Log.info("onAuthenticationSuccess:" + username);
        memRepo.save(Member.builder()
                        .username(username)
                        .password(encoder.encode("1a2s3d4f"))
                        .role("ROLE_MEMBER")
                        .enabled(true).build());

        String jwtToken = JWTUtil.getJWT(username);
        response.addHeader(HttpHeaders.AUTHORIZATION, jwtToken);
    }
}
```

JWT 토큰을 응답 헤더에 추가하는 핸들러

수정이 완료되면 제대로 로그인인지 서버 구동 및 테스트
<http://localhost:8080/login> → OAuth2 login 화면이 뜸

C.1.10 Custom Class(1)

CustomMyUtil.java

```
package edu.pnu.util;

public class CustomMyUtil {

    // OAuth2User 정보를 이용해서 임의의 사용자 아이디를 생성하는 메소드
    public static String getUsernameFromOAuth2User(OAuth2User user) {

        String userString = user.toString();
        String regName = null;
        // userString 조사
        if (userString.contains("google"))        regName = "Google";
        else if (userString.contains("facebook")) regName = "Facebook";
        else if (userString.contains("naver"))    regName = "Naver";
        else if (userString.contains("kakao"))    regName = "Kakao";
        else {
            if (userString.contains("id=") && userString.contains("resultcode=") && userString.contains("response="))
                regName = "Naver";
            else
                return null;
        }
        String name = user.getName();
        if (name == null) return null;

        return regName + "_" + name;
    }
}
```

C.1.11 Custom Class(2)

JWTUtil.java

```
package edu.pnu.util;

public class JWTUtil {
    private static final long ACCESS_TOKEN_MSEC = 10 * (60 * 1000); // 10분
    private static final String JWT_KEY = "edu.pnu.jwtkey";           // 인코딩을 위한 secret key
    private static final String claimName = "username";               // 토큰에 담을 정보의 key값
    private static final String prefix = "Bearer ";                  // JWT 토큰 헤더 문자열

    private static String getJWTSource(String token) {
        if (token.startsWith(prefix)) return token.replace(prefix, "");
        return token;
    }

    public static String getJWT(String username) {
        String src = JWT.create()
            .withClaim(claimName, username)
            .withExpiresAt(new Date(System.currentTimeMillis()+ACCESS_TOKEN_MSEC))
            .sign(Algorithm.HMAC256(JWT_KEY));

        return prefix + src;
    }

    public static String getClaim(String token) { // 토큰에 담긴 정보 중 key가 "username"인 데이터 가져오기
        String tok = getJWTSource(token);
        return JWT.require(Algorithm.HMAC256(JWT_KEY)).build().verify(tok).getClaim(claimName).asString();
    }

    public static boolean isExpired(String token) { // 유효기간 만료 여부
        String tok = getJWTSource(token);
        return JWT.require(Algorithm.HMAC256(JWT_KEY)).build().verify(tok).getExpiresAt().before(new Date());
    }
}
```

C.2 OAuth2 with Naver

C.2.1 설정 준비

- <https://developers.naver.com/main/> ➔ 구글 검색창에서 “naver oauth2 api” 검색

The left screenshot shows the Naver Developers main page. The 'Application' menu item in the top navigation bar is highlighted with a red dashed box. A red arrow points from this menu item to the 'Application' page shown in the right screenshot.

The right screenshot shows the 'Application' page. It features a green 'Application 등록' button and a table titled 'Application 목록'.

Client ID	Application 명	Action		
API 상태 보기				
상태	API URL	API 명	응답시간	업데이트 시각
✓	datalab/**	데이터랩 (쇼핑인사이드)	27 ms	2023. 11. 18 AM 11:58:16
✓	datalab/search	데이터랩 (검색어트렌드)	28 ms	2023. 11. 18 AM 11:58:16
✓	vision/**	얼굴인식	274 ms	2023. 11. 18 AM 11:58:15

C.2.2 Application 등록

하도 API 설정을 할 수 있습니다.

Application 등록

Application 명 Action

응답시간 업데이트 시각

애플리케이션 등록 (API 이용신청)

애플리케이션의 기본 정보를 등록하면, 좌측 내 애플리케이션 메뉴의 서브 메뉴에 등록하신 애플리케이션 이름으로 서브 메뉴가 만들어집니다.

애플리케이션 이름 ⇄

myApp

- 네이버 로그인할 때 사용자에게 표시되는 이름이므로 서비스 브랜드를 대표할 수 있는 이름으로 가급적 10자 이내로 간결하게 설정해주세요.
- 40자 이내의 영문, 한글, 숫자, 공백문자, 점표(.), "/", "-", "_", 만 입력 가능합니다.

사용 API ⇄

네이버 로그인

사용할 API를 추가해 주세요.

- [애플리케이션 이름] 설정을 확인해 주세요.
- [사용 API] 설정을 확인해 주세요.

등록하기 취소

네이버 로그인

제공 정보 선택(이용자 식별자는 기본 정보로 제공) ①

필수 항목은 개인정보보호법 제3조 제1항, 제16조 제1항 등에 따라 서비스 제공을 위해 필요한 최소한의 개인정보만을 선택해야 합니다.

권한	필수	추가
회원이름	<input checked="" type="checkbox"/>	<input type="checkbox"/>
연락처 이메일 주소	<input checked="" type="checkbox"/>	<input type="checkbox"/>
별명	<input type="checkbox"/>	<input type="checkbox"/>
프로필 사진	<input type="checkbox"/>	<input type="checkbox"/>
성별	<input type="checkbox"/>	<input type="checkbox"/>
생일	<input type="checkbox"/>	<input type="checkbox"/>
연령대	<input type="checkbox"/>	<input type="checkbox"/>
출생연도	<input type="checkbox"/>	<input type="checkbox"/>
휴대전화번호	<input type="checkbox"/>	<input type="checkbox"/>

로그인 오픈 API 서비스 환경 ②

반드시 검수에 통과되어야 네이버 로그인 사용이 가능합니다.

환경 추가

- 안드로이드
- iOS
- Mobile 웹
- PC 웹
- Windows App

확인하세요.

네이버 로그인 Callback URL (최대 5개)

http://localhost:8080/login/oauth2/code/naver

+ ✓

서비스 URL

http://localhost:8080

✓

등록하기 취소

C.2.3 OAuth2 Client 설정

application.properties

```
spring.security.oauth2.client.registration.naver.client-id=****
spring.security.oauth2.client.registration.naver.client-secret=****
spring.security.oauth2.client.registration.naver.scope=name,email
spring.security.oauth2.client.registration.naver.client-name=Naver
spring.security.oauth2.client.registration.naver.redirect-uri=http://localhost:8080/login/oauth2/code/naver
spring.security.oauth2.client.registration.naver.authorization-grant-type=authorization_code

spring.security.oauth2.client.provider.naver.authorization-uri=https://nid.naver.com/oauth2.0/authorize
spring.security.oauth2.client.provider.naver.token-uri=https://nid.naver.com/oauth2.0/token
spring.security.oauth2.client.provider.naver.user-info-uri=https://openapi.naver.com/v1/nid/me
spring.security.oauth2.client.provider.naver.user-name-attribute=response
```

- ➔ “****”는 네이버에서 애플리케이션 등록이 완료된 뒤에 설정된 client-id와 client-secret를 찾아 입력
- ➔ 나머지 항목을 그대로 사용하면 되고, Dependency 설정은 Google과 동일

C.2.4 링크 추가 및 테스트

index.html

```
<body th:align="center">
  <h2>로그인</h2>
  <form action="/login" method="POST">
    <table th:align="center" border="1" th:cellpadding="0">
      :
    </table>
  </form>
  <a href="http://localhost:8080/oauth2/authorization/google">구글 로그인</a><br>
  <a href="http://localhost:8080/oauth2/authorization/naver">네이버 로그인</a>
</body>
```

OAuth2 Client 사용 시 URL은 고정

Front-End 로그인 창에 [네이버 로그인] 링크 추가 → 호스트 주소(<http://localhost:8080>)는 Front-End와 스프링 부트 서버의 주소가 동일하면 생략가능 ("/oauth2/authorization/naver")

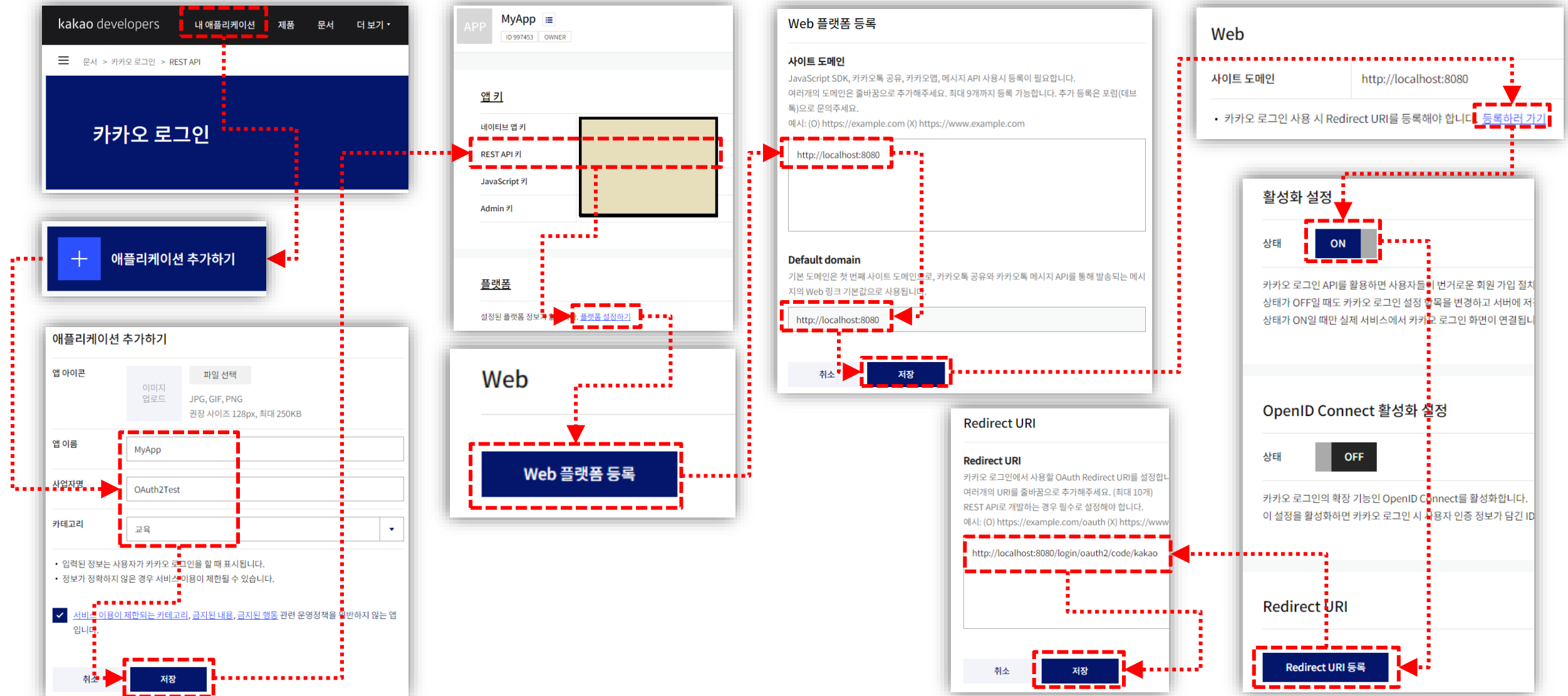
수정이 완료되면 제대로 로그인이 되는지 서버 구동 및 테스트

<http://localhost:8080/login> → OAuth2 login 화면에 구글과 네이버 로그인이 뜸

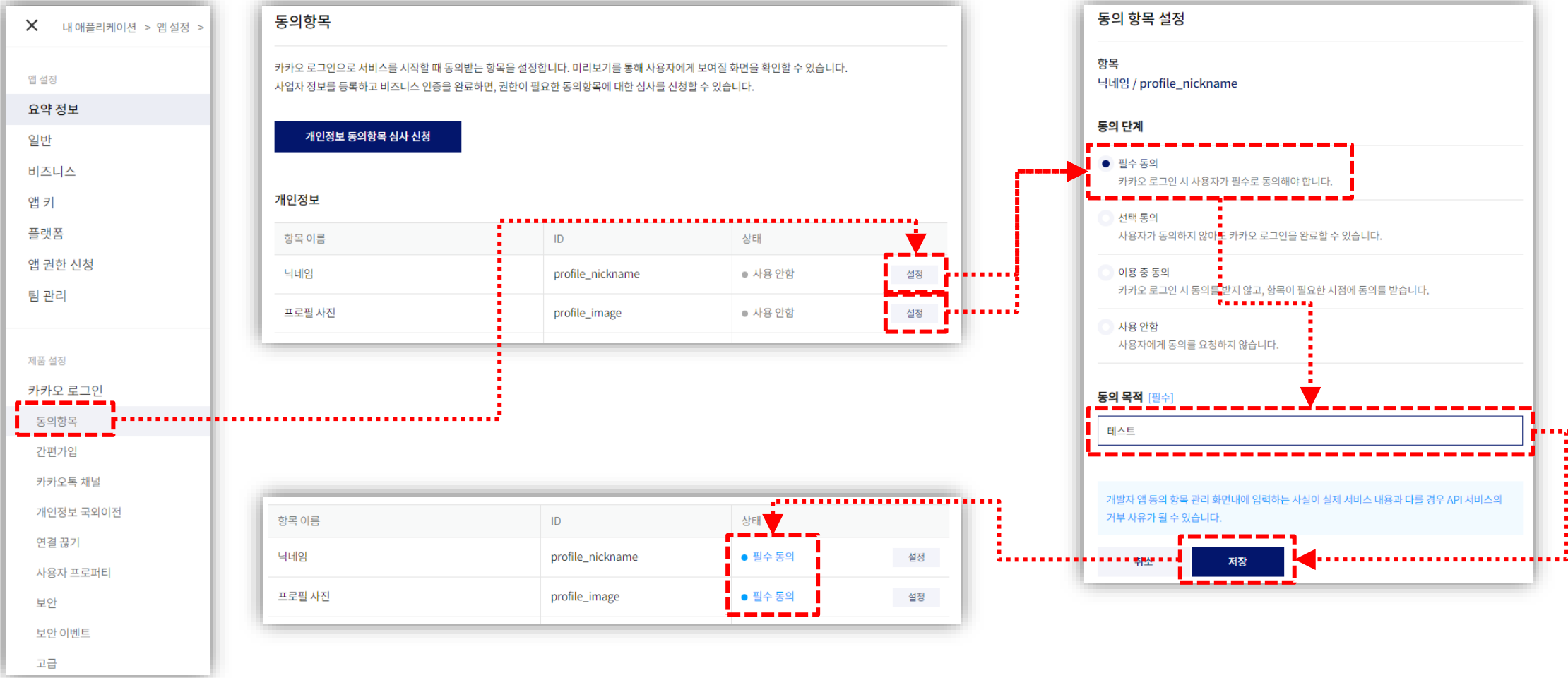
C.3 OAuth2 with Kakao

C.3.1 설정 준비

- <https://developers.kakao.com/docs/latest/ko/kakaologin/rest-api> → 구글 검색창에서 “kakao oauth2 api” 검색



C.3.2 설정 준비



C.3.3 OAuth2 Client 설정

application.properties

```
spring.security.oauth2.client.registration.kakao.client-id={REST API 키}
spring.security.oauth2.client.registration.kakao.client-authentication-method=client_secret_post
spring.security.oauth2.client.registration.kakao.scope=profile_nickname,profile_image
spring.security.oauth2.client.registration.kakao.client-name=Kakao
spring.security.oauth2.client.registration.kakao.redirect-uri=http://localhost:8080/login/oauth2/code/kakao
spring.security.oauth2.client.registration.kakao.authorization-grant-type=authorization_code

spring.security.oauth2.client.provider.kakao.authorization-uri=https://kauth.kakao.com/oauth/authorize
spring.security.oauth2.client.provider.kakao.token-uri=https://kauth.kakao.com/oauth/token
spring.security.oauth2.client.provider.kakao.user-info-uri=https://kapi.kakao.com/v2/user/me
spring.security.oauth2.client.provider.kakao.user-name-attribute=id
```

- ➔ {REST API 키}는 카카오에서 애플리케이션 추가하면 보이는 앱키에서 “REST API 키”를 입력
- ➔ 나머지 항목을 그대로 사용하면 되고, Dependency 설정은 Google과 동일

C.3.4 링크 추가 및 테스트

index.html

```
<body th:align="center">
  <h2>로그인</h2>
  <form action="/login" method="POST">
    <table th:align="center" border="1" th:cellpadding="0">
      :
    </table>
  </form>
  <a href="http://localhost:8080/oauth2/authorization/google">구글 로그인</a><br>
  <a href="http://localhost:8080/oauth2/authorization/naver">네이버 로그인</a><br>
  <a href="http://localhost:8080/oauth2/authorization/kakao">카카오 로그인</a>
</body>
```

OAuth2 Client 사용 시 URL은 고정

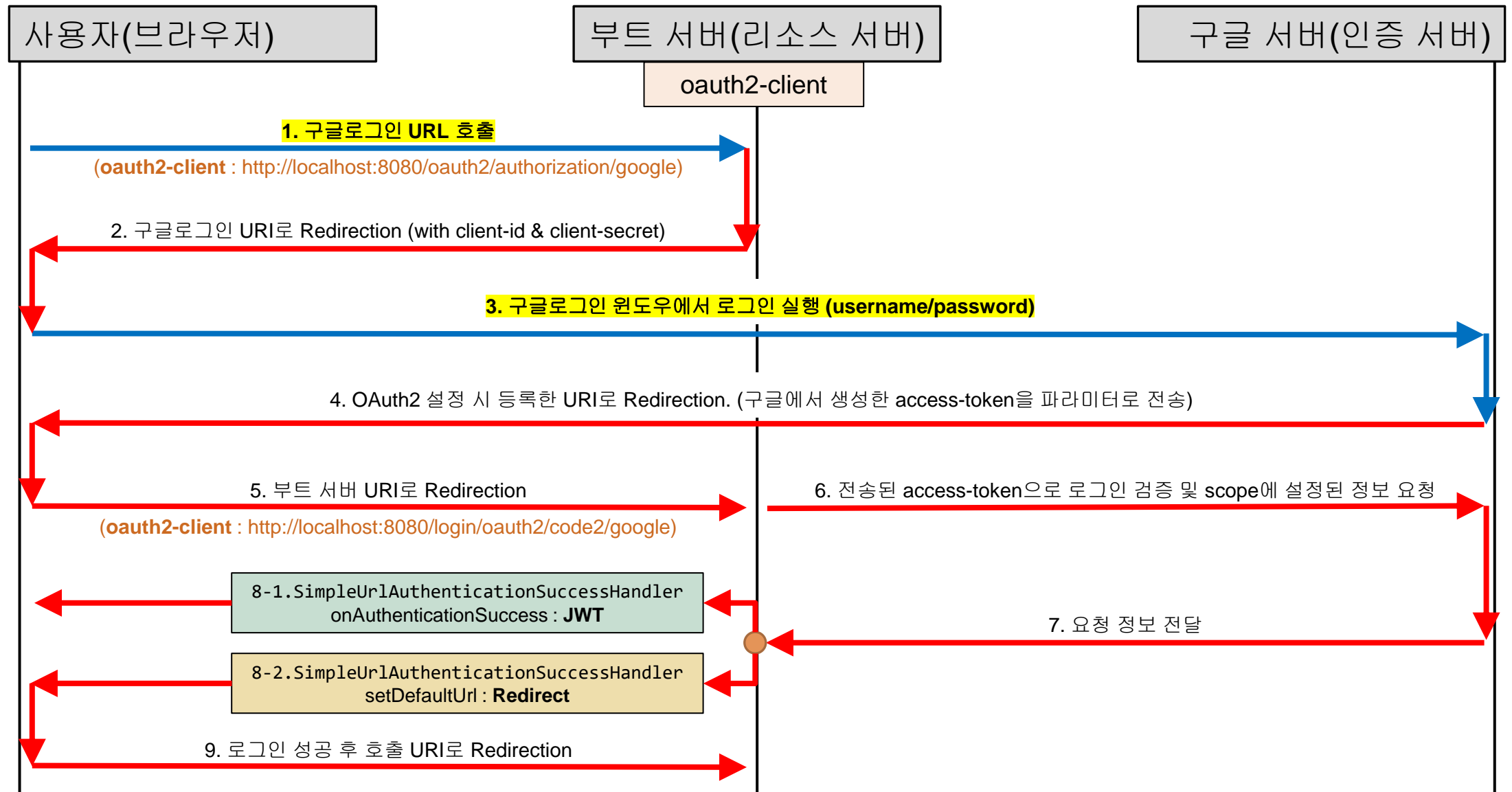
Front-End 로그인 창에 [카카오 로그인] 링크 추가 → 호스트 주소(<http://localhost:8080>)는 Front-End와 스프링 부트 서버의 주소가 동일하면 생략가능 ("/oauth2/authorization/kakao")

수정이 완료되면 제대로 로그인이 되는지 서버 구동 및 테스트

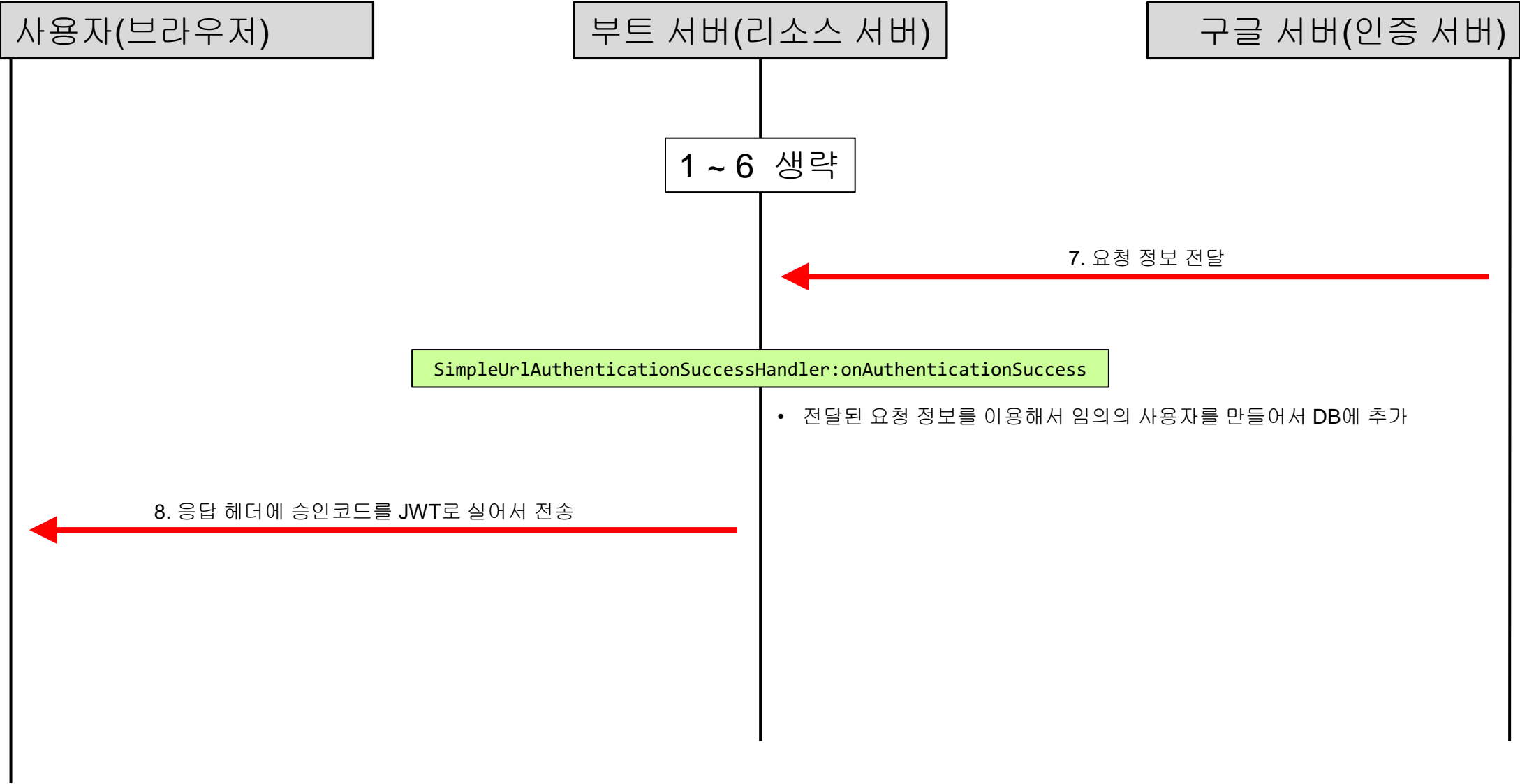
<http://localhost:8080/login> → OAuth2 login 화면에 구글과 네이버와 카카오 로그인이 뜸

C.4 OAuth2 with spring-boot-starter-oauth2-client

C.4.1 OAuth2 진행과정 (with spring-boot-starter-oauth2-client)



C.4.2 OAuth2 with JWT 진행과정 (with spring-boot-starter-oauth2-client)



D. CORS

(Cross-Origin Resource Sharing : 교차 출처 리소스 공유)

기본적으로 스프링 부트는 CORS 정책을 허용하지 않음.

D.1 CorsConfigurationSource를 메서드를 이용해서 등록하는 방법

SecurityConfig.java

```
public class SecurityConfig {  
    @Bean  
    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
        http.cors(cors->cors.configurationSource(corsSource()));  
        ~ 생략 ~  
    }  
    private CorsConfigurationSource corsSource() {  
        CorsConfiguration config = new CorsConfiguration();  
        config.addAllowedOriginPattern(CorsConfiguration.ALL);  
        config.addAllowedMethod(CorsConfiguration.ALL);  
        config.addAllowedHeader(CorsConfiguration.ALL);  
        config.setAllowCredentials(true);  
  
        config.addExposedHeader(CorsConfiguration.ALL);  
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();  
        source.registerCorsConfiguration("/**", config);  
        return source;  
    }  
}
```

Spring Security를 사용하는 경우 사용

// 요청을 허용할 서버
// 요청을 허용할 Method
// 요청을 허용할 Header
// 요청/응답에 자격증명정보 포함을 허용
// true인 경우 addAllowedOrigin("*")는 사용 불가 → Pattern으로 변경
// Header에 Authorization을 추가하기 위해서는 필요
// 위 설정을 적용할 Rest 서버의 URL 패턴 설정

D.2 WebMvcConfigurer를 상속한 클래스를 Bean 객체로 등록하는 방법

CustomConfig.java

```
@Configuration
public class CustomConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
            .allowCredentials(true)
            .allowedHeaders(HttpHeaders.AUTHORIZATION)
            .allowedMethods(HttpMethod.GET.name(),
                HttpMethod.POST.name(),
                HttpMethod.PUT.name(),
                HttpMethod.DELETE.name())
            .allowedOrigins("http://localhost:3000", "http://127.0.0.1:3000")
            .exposedHeader(CorsConfiguration.ALL);
    }
}
```

Spring Security가
없는 경우 사용

D.3 @CrossOrigin 어노테이션을 이용하는 방법

1. Controller 수준에서 설정

```
@RestController
@CrossOrigin(origins={"http://localhost:3000", "http://127.0.0.1:3000"})
public class MyRestApiController {
    ~ 생략 ~
}
```

2. Method 수준에서 설정

```
@RestController
public class MyRestApiController {
    @GetMapping("/api/data")
    @CrossOrigin(origins = {"http://localhost:3000", "http://127.0.0.1:3000"})
    public ResponseEntity<?> getData() {
        List<Car> list = dataRepo.findAll();
        return ResponseEntity.ok(list);
    }
}
```

- 두 방법 모두 로그인が必要な 메소드에 대해서는 적용 불가
- 로그인이 필요하지 않은 컨트롤러/메소드에 대해서만 적용 가능

E. User & OAuth2 통합

토큰 방식이 아니라 세션을 유지하는 형태로 구현할 때,
User와 OAuth2를 동시에 사용하면서 두 객체 타입을 하
나로 통합하고자 하는 경우

E.1 AuthUser

AuthUser.java : UserDetails와 OAuth2User를 동시에 구현한 클래스

```
public class AuthUser implements UserDetails, OAuth2User{
    private static final long serialVersionUID = 1L;

    private Member member;
    private Map<String, Object> attributes;

    public AuthUser(Member member) {
        this.member = member;
    }

    public AuthUser(Member member, Map<String, Object> attributes){
        this.member = member;
        this.attributes = attributes;
    }

    @Override
    public Map<String, Object> getAttributes() {
        return attributes;
    }

    @Override
    public String getName() {
        return null;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities(){
        return AuthorityUtils.createAuthorityList(member.getRole());
    }
}
```

```
@Override
public String getPassword() {
    return member.getPassword();
}

@Override
public String getPassword() {
    return member.getPassword();
}

@Override
public String getUsername() {
    return member.getUsername();
}

@Override
public boolean isAccountNonExpired() { return true; }

@Override
public boolean isAccountNonLocked() { return true; }

@Override
public boolean isCredentialsNonExpired() { return true; }

@Override
public boolean isEnabled() { return true; }

@Override
public String toString() {
    return "AuthUser [" + member + "]";
}
}
```

E.2 UserDetailsService 구현체

SecurityUserDetailsService.java

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import edu.pnu.domain.Member;
import edu.pnu.persistence.MemberRepository;

@Service
public class SecurityUserDetailsService implements UserDetailsService {

    @Autowired
    private MemberRepository memRepo;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        Member member = memRepo.findById(username).orElseThrow();
        //return new User(member.getUsername(), member.getPassword(), member.getAuthorities());
        return new AuthUser(member);
    }
}
```

E.3 DefaultOAuth2UserService 상속구현

OAuth2UserDetailsService.java

```
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.Component;

@RequiredArgsConstructor
@Component
public class OAuth2UserDetailService extends DefaultOAuth2UserService {
    @Override
    public OAuth2User loadUser(OAuth2UserRequest userRequest) throws OAuth2AuthenticationException {
        OAuth2User user = super.loadUser(userRequest);
        Map<String, Object> map = user.getAttributes();
        Member member = Member.builder()
            .username((String)map.get("sub"))
            .password("[PROTECTED]")
            .enabled(true)
            .role("ROLE_USER")
            .build();
        return new AuthUser(member, map);
    }
}
```


F. 구현 실습

F.1 시큐리티 적용 시나리오 (JWT)

요청 URL	의미
/public	인증을 하지 않은 모든 사용자가 접근할 수 있다.
/intra/marketing	MARKET 권한을 가진 사용자가 접근할 수 있다.
/intra/develop	DEVELOP 권한을 가진 사용자가 접근할 수 있다.
/intra/finance	FINANCE 권한을 가진 사용자가 접근할 수 있다.
/admin	ADMIN 권한을 가진 사용자가 접근할 수 있다. ADMIN 권한을 가진 사람을 모든 URL에 접근할 수 있다.