

# CH20 스프링부트 인증 - 세션과 쿠키

<https://github.com/codingspecialist/Springboot-Session-Cookie.git>

## 1. 세션과 쿠키

대한 응답을 반환하기 전에 세션을 생성하고 클라이언트에게 세션 ID를 제공합니다. 클라이언트는 이 세션 ID를 사용하여 다음 요청에서 서버에 대한 자신의 식별자로 사용합니다. 세션은 기본적으로 서버 메모리에 저장되며, 필요한 경우 외부 저장소에 저장할 수 있습니다.

반면 쿠키는 클라이언트 측에서 상태를 유지합니다. 서버는 요청에 대한 응답에서 쿠키를 생성하고 클라이언트에게 제공합니다. 클라이언트는 이 쿠키를 저장하고, 다음 요청에서 해당 쿠키를 서버에게 전달하여 자신을 식별합니다.

세션과 쿠키는 모두 상태를 유지하는 데 사용됩니다. 그러나 세션은 보안성이 높으며, 클라이언트에 대한 정보를 서버 측에서 유지하기 때문에 쿠키보다 더 많은 정보를 저장할 수 있습니다. 반면 쿠키는 클라이언트 측에서 저장되기 때문에, 서버에서 세션을 생성하는 것보다 더 빠르고 간단합니다.

스프링 부트에서는 세션과 쿠키 모두를 사용할 수 있습니다. 세션은 `HttpSession` 객체를 사용하여 관리하고, 쿠키는 `javax.servlet.http.Cookie` 클래스를 사용하여 생성 및 관리할 수 있습니다.

세션은 모든 유저가 공유하는 메모리 영역이기 때문에 IoC 컨테이너에서 의존성 주입을 할 수도 있고, `DispatcherServlet`에서 메서드 파라미터로 직접 의존성 주입 받을 수 도 있다.

그림 설명 필요 (세션과 쿠키)

## 2. JSessionID

JSESSIONID는 웹 애플리케이션에서 세션을 구분하는 데 사용되는 식별자입니다. JSESSIONID는 클라이언트가 서버에 최초 요청을 보낼 때, 서버에서 생성되며, 서버에서 유지되는 세션 ID입니다. 클라이언트는 JSESSIONID를 쿠키 형태로 저장하고, 이후 요청에서는 JSESSIONID를 서버에게 전달하여 해당 클라이언트의 세션을 식별합니다.

JSESSIONID는 Java Servlet 스펙에 정의된 이름으로, Java 웹 어플리케이션 서버인 Tomcat, Jetty 등에서 기본적으로 사용됩니다. JSESSIONID는 클라이언트와 서버 간의 통신에서 중요한 역할을 합니다. 예를 들어, 사용자가 로그인하면 서버는 사용자를 구분하기 위해 JSESSIONID를 생성하고, 이후의 모든 요청에서 이 세션 ID를 사용하여 사용자의 상태를 유지합니다.

JSESSIONID는 기본적으로 쿠키를 사용하여 클라이언트에 저장됩니다. 만약 클라이언트가 쿠키를 지원하지 않는 경우에는 URL 리다이렉션을 통해 JSESSIONID를 전달합니다. 이러한 경우에는 URL 뒤에 JSESSIONID를 추가하여 전달합니다.

JSESSIONID는 기본적으로 안전하지 않은 형태로 전달됩니다. 따라서 HTTPS 프로토콜을 사용하거나, JSESSIONID를 암호화하거나, JSESSIONID를 포함한 쿠키를 HttpOnly 속성을 설정하여 XSS 공격으로부터 보호할 필요가 있습니다.

그림 설명 필요 (JSessionID)

### 3. CORS (개념만)

CORS(Cross-Origin Resource Sharing)는 웹 브라우저에서 실행되는 스크립트가 다른 도메인의 리소스에 접근할 때 적용되는 보안 정책입니다. 일반적으로, 웹 브라우저는 보안상의 이유로 자바스크립트에서 다른 도메인의 리소스에 접근하는 것을 허용하지 않습니다. 이를 Same-Origin Policy라고 부릅니다.

CORS 정책은 이러한 Same-Origin Policy를 우회할 수 있는 방법을 제공합니다. 서버 측에서는 리소스에 대한 요청 헤더에 허용할 도메인을 명시하여, 해당 도메인에서의 요청에 한해 리소스 접근을 허용할 수 있습니다.

CORS 정책은 다음과 같은 방식으로 동작합니다.

1. 브라우저에서 웹 페이지가 로드되면, 자바스크립트에서 다른 도메인의 리소스를 요청합니다.
2. 서버는 요청 헤더에 Origin이 포함된 요청을 받습니다.
3. 서버는 해당 Origin이 허용된 도메인인지 확인합니다.
4. 허용된 도메인일 경우, 서버는 응답 헤더에 Access-Control-Allow-Origin을 포함하여, 해당 도메인에서의 요청에 한해 리소스 접근을 허용합니다.
5. 브라우저는 Access-Control-Allow-Origin 헤더를 확인하고, 허용된 도메인에서의 요청에 한해 리소스에 접근합니다.

CORS 정책은 보안상의 이유로 브라우저에서 실행되는 스크립트에만 적용되며, 서버 간 통신에는 적용되지 않습니다. 따라서, 서버 간 API 호출 등에서는 다른 방법으로 보안을 구성해야 합니다.

그림 설명 필요

### 4. XSS 공격 (개념만) - 본인사이트에서 공격

<https://github.com/codingspecialist/Springboot-Javascript-Attack.git>

XSS(Cross-Site Scripting) 공격은 웹 사이트에서 발생하는 보안 취약점 중 하나로, 공격자가 악성 스크립트를 삽입하여 사용자의 개인정보를 탈취하거나, 세션 하이재킹 등의 악의적인 행동을 할 수 있게 됩니다.

XSS 공격은 보통 웹 애플리케이션에서 입력 값 검증이나 출력 값 인코딩이 제대로 이루어지지 않을 때 발생합니다. 공격자는 이러한 취약점을 이용하여 악성 스크립트를 삽입하고, 사용자의 브라우저에서 이 스크립트가 실행될 수 있도록 합니다.

XSS 공격은 크로스 사이트 스크립팅(Cross-Site Scripting)이라고도 부릅니다. XSS 공격은 저장형(XSS Stored)과 반사형(XSS Reflected) 두 가지 종류가 있습니다.

- 저장형 XSS : 악성 스크립트가 서버에 저장되어, 해당 페이지를 방문한 모든 사용자에게 영향을 끼칩니다. 예를 들어, 게시판에 악성 스크립트를 삽입하여, 해당 게시판을 방문하는 모든 사용자가 공격을 받을 수 있습니다.
- 반사형 XSS : 악성 스크립트가 사용자의 입력 값으로부터 생성되어, 해당 페이지를 방문한 특정 사용자에게만 영향을 끼칩니다. 예를 들어, 검색어 등 사용자의 입력 값을 검색 결과로 반환하는 페이지에 악성 스크립트를 삽입하여, 해당 검색어를 입력한 사용자만 공격을 받을 수 있습니다.

XSS 공격을 방지하기 위해서는, 입력 값 검증과 출력 값 인코딩이 제대로 이루어져야 합니다. 또한, 웹 애플리케이션에서는 쿠키를 사용하여 세션 ID를 저장하는 등, XSS 공격으로부터 보호할 수 있는 다양한 보안 기능들이 제공됩니다.

그림 설명 필요

## 4.1 저장형

저장형 XSS 공격의 예시를 들어보겠습니다.

예를 들어, 사용자들이 글을 쓰고 읽을 수 있는 게시판이 있다고 가정해봅시다. 사용자들은 게시판에 글을 작성하고, 다른 사용자들도 해당 글을 읽을 수 있습니다. 이 때, 게시판에 악성 스크립트가 삽입되어 있는 경우, 해당 페이지를 방문한 모든 사용자들이 공격을 받을 수 있습니다.

공격자는 게시판의 입력 폼에 악성 스크립트를 삽입합니다. 이 악성 스크립트는 사용자들의 브라우저에서 실행되어, 사용자들의 정보를 탈취하거나, 악의적인 행동을 수행할 수 있습니다. 이후, 게시글을 작성하고 등록하면, 해당 페이지를 방문한 모든 사용자들이 악성 스크립트에 노출됩니다.

이러한 경우를 방지하기 위해서는, 서버에서 입력 값 검증을 철저히 하고, 입력된 내용에 대해 인코딩을 해주는 등의 보안 절차가 필요합니다. 또한, 사용자들이 입력한 내용을 저장할 때는, 스크립트를 실행할 수 없도록 필터링을 해주는 등의 방법을 사용해야 합니다.

## 4.2 반사형

반사형 XSS 공격의 예시를 들어보겠습니다.

예를 들어, 사용자들이 검색어를 입력하여 검색 결과를 반환하는 검색 엔진이 있다고 가정해봅시다. 검색 엔진은 사용자의 입력 값을 검색어로 사용하고, 해당 검색어를 포함하는 웹 페이지를 반환합니다. 이 때, 검색어에 악성 스크립트가 포함되어 있는 경우, 해당 검색어를 입력한 특정 사용자들이 공격을 받을 수 있습니다.

공격자는 검색 엔진의 검색어 입력 폼에 악성 스크립트를 삽입합니다. 이 악성 스크립트는 검색 결과 페이지에 포함되어, 해당 페이지를 방문한 특정 사용자들이 공격을 받게 됩니다. 예를 들어, 검색 결과 페이지에서 사용자의 쿠키 정보를 탈취하는 스크립트를 실행하게 됩니다.

이러한 경우를 방지하기 위해서는, 검색어를 검색하기 전에, 입력 값 검증과 출력 값 인코딩이 제대로 이루어져야 합니다. 또한, 검색 결과를 반환할 때는, 스크립트를 실행할 수 없도록 필터링을 해주는 등의 방법을 사용해야 합니다.

## 5. CSRF 토큰 (개념만) - 타사사이트에서 공격

### 5.1 영상 참고

<https://www.youtube.com/watch?v=cC1tI8c000k>

CSRF(Cross-Site Request Forgery) 공격은 공격자가 인증된 사용자의 권한을 이용하여, 악의적인 요청을 전송하는 공격입니다. 예를 들어, 사용자가 의도하지 않은 상황에서 금융 거래를 실행하거나, 개인 정보를 수정하는 등의 행동을 수행할 수 있습니다.

이러한 CSRF 공격을 방지하기 위해서는, CSRF 토큰을 사용하여 악의적인 요청을 걸러내는 등의 보안 기능이 필요합니다. CSRF 토큰은 웹 애플리케이션에서 발급하는 임시 토큰으로, 해당 토큰이 일치하는 경우에만 요청을 처리하도록 설정됩니다.

예를 들어, 웹 애플리케이션에서 사용자가 금융 거래를 실행하는 페이지가 있다고 가정해봅시다. 이 때, 사용자가 해당 페이지를 방문할 때마다 임시로 생성된 CSRF 토큰이 발급됩니다.

사용자가 금융 거래를 실행하는 경우, 해당 페이지에서는 CSRF 토큰을 함께 요청하도록 설정됩니다. 이후, 요청이 서버로 전송될 때, 서버는 CSRF 토큰의 유효성을 검사하고, 일치하는 경우에만 요청을 처리합니다. 이를 통해, 공격자가 위조한 요청은 CSRF 토큰이 일치하지 않아 걸러낼 수 있습니다.

즉, CSRF 토큰은 인증된 사용자의 세션과 결합하여, 악성 요청을 걸러내는 역할을 합니다. 이러한 보안 기능을 통해, 사용자의 권한이 무단으로 사용되는 것을 방지할 수 있습니다.

예를 들어, 은행 웹 사이트에 로그인한 사용자가 있다고 가정해봅시다. 이 사용자가 은행 계좌 이체를 실행하기 위해, 은행 웹 사이트에서 이체 페이지를 열었습니다.

이 때, 공격자는 다음과 같은 스크립트를 이메일이나 다른 웹 사이트의 링크 등을 통해 사용자에게 전송합니다.

```

```

이 스크립트는 은행 계좌 이체를 실행하는 URL을 포함하고 있습니다. 사용자가 이메일이나 링크를 클릭하여 이 스크립트가 실행될 경우, 사용자의 브라우저는 해당 URL로 요청을 전송합니다.

하지만, 이 요청은 사용자의 권한을 이용하는 것으로, 사용자의 의사와는 관계 없이 요청이 전송됩니다. 따라서, 공격자는 사용자의 은행 계좌에서 1000원을 이체할 수 있습니다.

이러한 경우를 방지하기 위해서는, CSRF 토큰을 사용하여 악의적인 요청을 걸러내는 등의 보안 기능이 필요합니다. 예를 들어, 사용자가 은행 계좌 이체 페이지를 열 때, 은행 웹 사이트에서 임시로 생성된 CSRF 토큰이 발급됩니다.

이후, 사용자가 이체를 실행하는 경우, 해당 페이지에서는 CSRF 토큰을 함께 요청하도록 설정됩니다. 이후, 요청이 서버로 전송될 때, 서버는 CSRF 토큰의 유효성을 검사하고, 일치하는 경우에만 요청을 처리합니다. 이를 통해, 공격자가 위조한 요청은 CSRF 토큰이 일치하지 않아 걸러낼 수 있습니다.

그림 설명 필요 (CSRF 토큰을 생성하고 그것을 모든 태그에 심어서 보내기 - 위조된 태그는 CSRF 토큰이 없다)

CSRF(Cross-site request forgery) 공격은 인터넷 사용자가 자신도 모르게 다른 사람의 권한을 빌려서 악의적인 행동을 하는 공격입니다. 이해를 돕기 위해 간단한 예시를 들어 설명하겠습니다.

상상해보세요. 친구와 함께 학교에서 가장 인기 있는 소셜 미디어 웹사이트를 사용하고 있다고 가정합니다. 이 웹사이트에서는 사용자들이 게시물을 작성하고 친구들과 공유할 수 있습니다. 이 사이트에 로그인하면 사용자가 자신의 계정으로 할 수 있는 모든 행동이 가능합니다.

그런데 이 사이트를 사용하면서 다른 웹사이트를 둘러보기 시작했습니다. 그 중 한 웹사이트는 게임을 소개하고 있는데, 이 게임을 시작하려면 버튼을 클릭하면 된다고 합니다. 그래서 버튼을 클릭했죠. 그런데 이 버튼을 클릭하자마자, 소셜 미디어 웹사이트에서 게시물이 삭제되거나 친구들에게 이상한 메시지가 보내지는 것을 발견했습니다.

이런 일이 어떻게 발생했을까요? 바로 CSRF 공격 때문입니다. 게임 웹사이트의 버튼은 사용자가 모르게 소셜 미디어 웹사이트에 악의적인 요청을 보내도록 만든 것이었습니다. 이 요청은 사용자가 로그인한 상태에서 이루어지기 때문에 소셜 미디어 웹사이트는 이 요청이 사용자로부터 온 것처럼 인식하고 처리해버립니다. 결국 사용자는 자신도 모르게 소셜 미디어 웹사이트에서 악의적인 행동을 한 것이 되는 것입니다.

이러한 CSRF 공격을 막기 위해서는 웹사이트에서 인증 과정에 추가적인 보안 수단을 도입해야 합니다. 예를 들어, 사용자가 요청을 보낼 때마다 서버는 이 요청이 정말로 사용자로부터 온 것인지 확인하기 위해 특별한 코드를 요구할 수 있습니다. 이렇게 하면 공격자는 사용자의 권한을 빌려 악의적인 행동을 할 수 없게 됩니다.

## CSRF와 XSS 차이

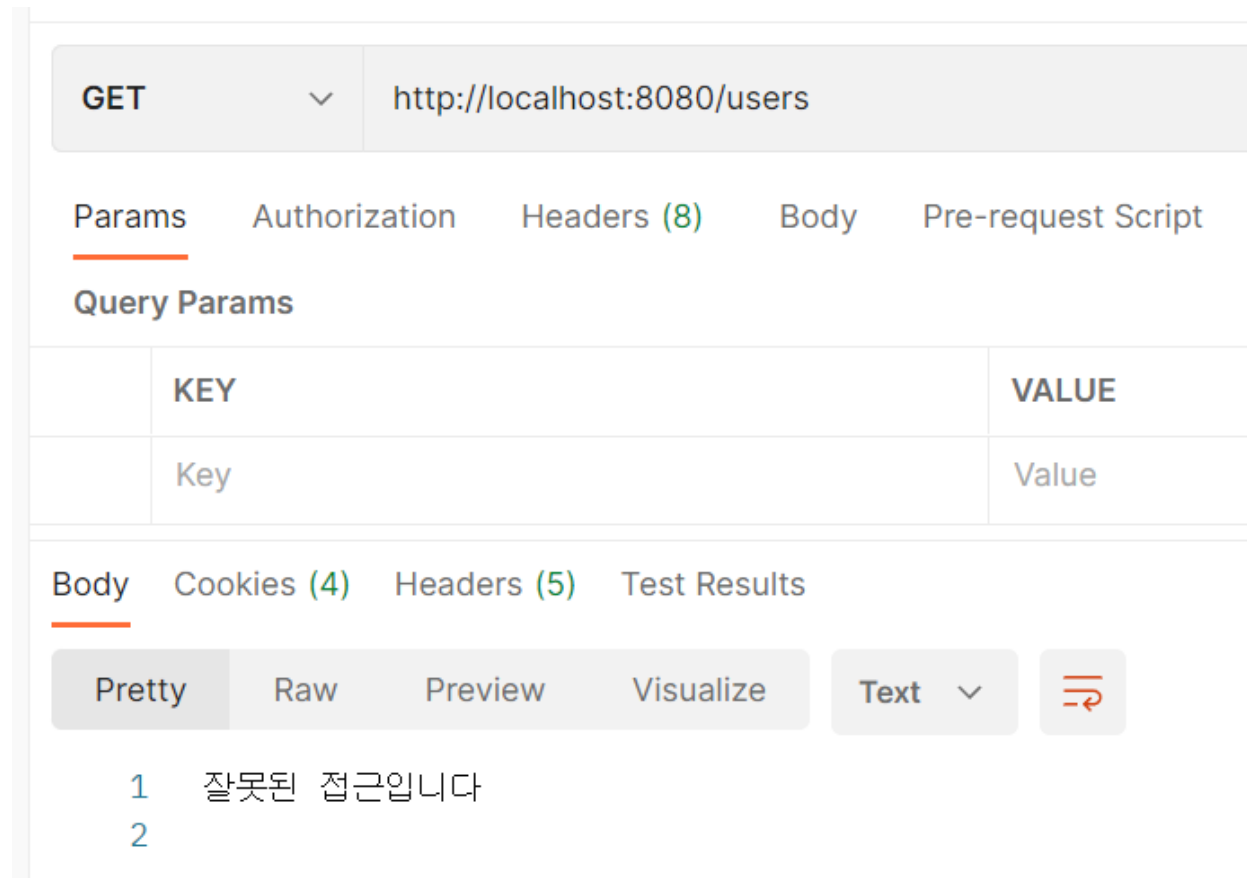
CSRF(Cross-site request forgery)와 XSS(Cross-site scripting)는 모두 웹 보안 취약점이지만, 공격 방법과 목표가 다릅니다.

1. CSRF(Cross-site request forgery): CSRF 공격은 사용자가 로그인한 상태에서, 공격자가 다른 웹사이트를 통해 사용자로 하여금 악의적인 요청을 보내게 만드는 공격입니다. 이렇게 함으로써 공격자는 사용자의 권한을 빌려 특정 행동을 수행하게 됩니다. 예를 들어, 사용자가 로그인한 상태에서 공격자가 만든 페이지에 접근하면, 공격자가 원하는 행동(게시물 삭제, 패스워드 변경 등)이 사용자의 권한으로 수행되는 것입니다.
2. XSS(Cross-site scripting): XSS 공격은 공격자가 웹사이트에 악의적인 스크립트를 삽입하여, 다른 사용자가 그 스크립트를 실행하게 하는 공격입니다. 이 공격은 주로 웹사이트의 입력 폼이나 댓글 기능 등을 통해 이루어집니다. 사용자가 악의적인 스크립트가 포함된 페이지를 열면, 스크립트가 실행되어 사용자의 정보를 탈취하거나 다른 악의적인 행동을 할 수 있습니다.

두 공격 모두 웹사이트의 보안을 해치기 때문에, 웹 개발자는 CSRF 토큰, 콘텐츠 보안 정책(Content Security Policy) 등의 기술을 활용하여 이러한 취약점을 방어해야 합니다.

## 6. 인증 서버 실습

### 6.1 로그인 하지 않은 상태



The screenshot shows a web client interface with the following details:

- Method: GET
- URL: http://localhost:8080/users
- Params: Query Params table with columns KEY and VALUE, containing a single entry: Key, Value.
- Body: Selected tab showing the response body as '잘못된 접근입니다' (Access is denied).
- Other tabs: Cookies (4), Headers (5), Test Results.
- Response format: Pretty, Raw, Preview, Visualize.
- Response type: Text.

### 6.2 로그인 한 상태

인터셉터로 처리

```
package shop.mtcoding.hiberpc.config.auth;

import org.springframework.web.servlet.HandlerInterceptor;
import shop.mtcoding.hiberpc.model.User;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```

import javax.servlet.http.HttpSession;

public class LoginInterceptor implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
Object handler) throws Exception {
        HttpSession session = request.getSession();
        User loginUser = (User) session.getAttribute("loginUser");
        if(loginUser == null){
            response.setContentType("text/plain; charset=utf-8");
            response.getWriter().println("잘못된 접근입니다");
            return false;
        }else{
            return true;
        }
    }
}

```

```

package shop.mtcoding.hiberpc.config;

import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.method.support.HandlerMethodArgumentResolver;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import shop.mtcoding.hiberpc.config.auth.LoginInterceptor;
import shop.mtcoding.hiberpc.config.auth.MyLoginArgumentResolver;

import java.util.List;

@RequiredArgsConstructor
@Configuration
public class WebMvcConfig implements WebMvcConfigurer {

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(new LoginInterceptor())
            .addPathPatterns("/users/**");
    }
}

```

GET

▼

http://localhost:8080/users

ParamsAuthorizationHeaders (8)BodyPre-request ScriptTestsS

Query Params

	KEY	VALUE
	Key	Value

BodyCookies (4)Headers (5)Test Results

PrettyRawPreviewVisualizeJSON▼

```
1  [  
2    {  
3      "id": 1,  
4      "username": "ssar",  
5      "email": "ssar@nate.com",  
6      "createdAt": "2023-03-25T03:40:41.954+00:00"  
7    },  
8    {  
9      "id": 2,  
10     "username": "cos",  
11     "email": "cos@nate.com",  
12     "createdAt": "2023-03-25T03:40:46.909+00:00"  
13   }  
14 ]
```

### 6.3 권한이 없는 상태



GET

⌵

http://localhost:8080/users/1/v1

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Query Params

	KEY	VALUE
	Key	Value

Body

Cookies (4)

Headers (4)

Test Results

Pretty

Raw

Preview

Visualize

Text ⌵

↺

1

고객 정보를 볼 수 있는 권한이 없습니다

## 6.4 권한이 있는 상태

HandlerMethodArgumentResolver로 처리

-> 리플렉션으로 처리하는 방식을 스프링부트에서 쉽게 제공해준다.

```

package shop.mtcoding.hiberpc.config.auth;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
public @interface LoginUser {
}

```

```

package shop.mtcoding.hiberpc.config.auth;

import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.MethodParameter;
import org.springframework.web.bind.support.WebDataBinderFactory;
import org.springframework.web.context.request.NativeWebRequest;
import org.springframework.web.method.support.HandlerMethodArgumentResolver;
import org.springframework.web.method.support.ModelAndViewContainer;
import shop.mtcoding.hiberpc.config.auth.LoginUser;
import shop.mtcoding.hiberpc.model.User;

import javax.servlet.http.HttpSession;

@RequiredArgsConstructor
@Configuration
public class MyLoginArgumentResolver implements HandlerMethodArgumentResolver {

    private final HttpSession session;

    @Override
    public boolean supportsParameter(MethodParameter parameter) {
        boolean check1 = parameter.getParameterAnnotation(LoginUser.class) != null;
        boolean check2 = User.class.equals(parameter.getParameterType());
        return check1 && check2;
    }

    @Override
    public Object resolveArgument(MethodParameter parameter, ModelAndViewContainer
mavContainer, NativeWebRequest webRequest, WebDataBinderFactory binderFactory) throws
Exception {

```

```

        return session.getAttribute("loginUser");
    }
}

```

```

package shop.mtcoding.hiberpc.config;

import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.method.support.HandlerMethodArgumentResolver;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import shop.mtcoding.hiberpc.config.auth.LoginInterceptor;
import shop.mtcoding.hiberpc.config.auth.MyLoginArgumentResolver;

import java.util.List;

@RequiredArgsConstructor
@Configuration
public class WebMvcConfig implements WebMvcConfigurer {

    private final MyLoginArgumentResolver myLoginArgumentResolver;

    @Override
    public void addArgumentResolvers(List<HandlerMethodArgumentResolver> resolvers) {
        resolvers.add(myLoginArgumentResolver);
    }
}

```

```

// 인증과 권한 - 리졸버
@GetMapping("/users/{id}/v2")
public ResponseEntity<?> userDetailsV2(@PathVariable Integer id, @LoginUser User
loginUser){
    if(!loginUser.getId().equals(id)){
        throw new MyException("고객 정보를 볼 수 있는 권한이 없습니다");
    }
    User userPS = userRepository.findById(id).orElseThrow(
        ()-> new MyException("해당 유저를 찾을 수 없습니다")
    );
    return new ResponseEntity<>(userPS, HttpStatus.OK);
}

```


GET ⌵ http://localhost:8080/users/2/v1

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Query Params

	KEY	VALUE
	Key	Value

Body Cookies (4) Headers (5) Test Results

Pretty Raw Preview Visualize JSON ⌵ 

```

1  {
2      "id": 2,
3      "username": "ssar",
4      "email": "ssar@nate.com",
5      "createdAt": "2023-03-25T03:42:51.524+00:00"
6  }

```

## 7. 쿠키 실습

```

@PostMapping("/products/{id}/cart")
public ResponseEntity<> addCart(@PathVariable Integer id, HttpServletRequest request,
    HttpServletResponse response){
    String productNames = "";
    Cookie[] cookies = request.getCookies();

    // 장바구니가 존재할 때
    boolean checkCookie = false;
    for (Cookie c: cookies) {
        if(c.getName().equals("cart")){
            productNames += c.getValue()+" "+id.toString();
            checkCookie = true;
        }
    }

    // 장바구니가 존재하지 않을 때
    if(checkCookie == false){

```

```

        productNames = id.toString();
    }

    Cookie cookie = new Cookie("cart", productNames);
    cookie.setPath("/");
    cookie.setMaxAge(1000*60*60); // 1시간
    cookie.setHttpOnly(false); // document.cookie
    response.addCookie(cookie);
    return new ResponseEntity<>("쿠키등록됨", HttpStatus.OK);
}

```

POST

http://localhost:8080/products/1/cart

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

### Query Params

	KEY	VALUE
	Key	Value

Body

Cookies (5)

Headers (6)

Test Results

Pretty

Raw

Preview

Visualize

Text



1 쿠키등록됨

Body	Cookies (5)	Headers (6)	Test Results
Status: 200 OK Time: 25 ms Size: 263 B Save Response			
KEY	VALUE		
Set-Cookie	①	cart=1; Max-Age=3600000; Expires=Fri, 05 May 2023 19:48:20 GMT; Path=/	
Content-Type	①	text/plain; charset=UTF-8	
Content-Length	①	15	
Date	①	Sat, 25 Mar 2023 03:48:20 GMT	
Keep-Alive	①	timeout=60	
Connection	①	keep-alive	

쿠키를 추가할 때 마다 Response Headers에 Set-Cookie에 /로 구분하여 하나씩 추가된다.

POST ⌵ http://localhost:8080/products/3/cart

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies (5) Headers (6) Test Results ⊕ Status: 200 OK Time: 22 ms Size: 267 B ⌵

KEY	VALUE
Set-Cookie	<span>ⓘ</span> cart=1/2/3; Max-Age=3600000; Expires=Fri, 05 May 2023 19:48:56 GMT; Path=/ <span>ⓘ</span>
Content-Type	<span>ⓘ</span> text/plain; charset=UTF-8
Content-Length	<span>ⓘ</span> 15
Date	<span>ⓘ</span> Sat, 25 Mar 2023 03:48:56 GMT
Keep-Alive	<span>ⓘ</span> timeout=60
Connection	<span>ⓘ</span> keep-alive