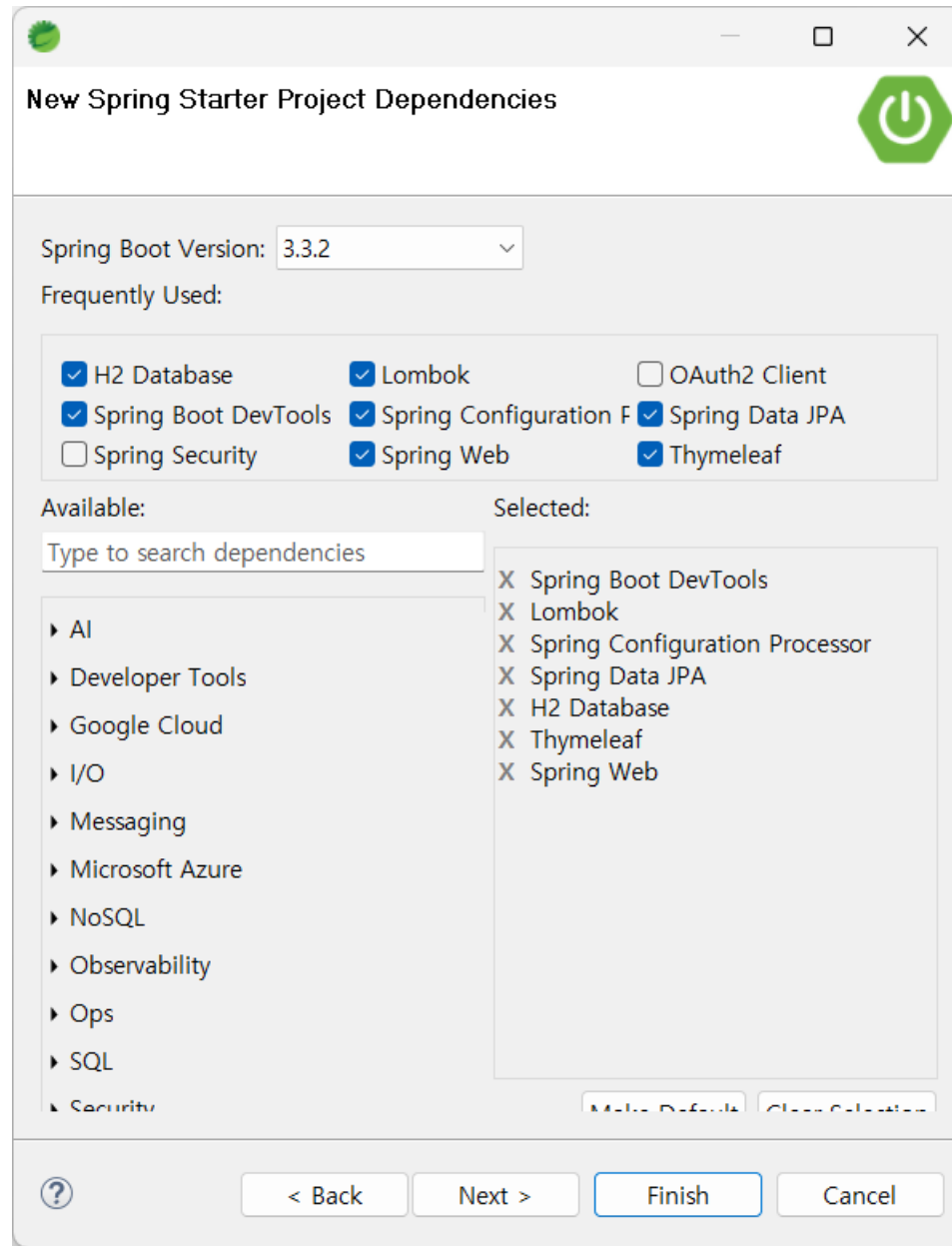


Spring Security

Spring Boot

packag명
com.shop



The image shows a 'New Spring Starter Project Dependencies' dialog box. At the top, it has a title bar with a green icon and standard window controls. Below the title bar, there's a green power button icon. The main content area is divided into several sections. The first section is 'Spring Boot Version:' with a dropdown menu set to '3.3.2'. Below that is 'Frequently Used:' which contains a grid of checkboxes for various dependencies: H2 Database, Lombok, OAuth2 Client, Spring Boot DevTools, Spring Configuration Processor, Spring Data JPA, Spring Security, Spring Web, and Thymeleaf. The next section is 'Available:' which includes a search bar labeled 'Type to search dependencies' and a list of categories: AI, Developer Tools, Google Cloud, I/O, Messaging, Microsoft Azure, NoSQL, Observability, Ops, SQL, and Security. To the right of this is a 'Selected:' list showing the currently chosen dependencies with an 'X' next to each: Spring Boot DevTools, Lombok, Spring Configuration Processor, Spring Data JPA, H2 Database, Thymeleaf, and Spring Web. At the bottom of the dialog, there are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'. A help icon (?) is also present in the bottom left corner.

New Spring Starter Project Dependencies

Spring Boot Version: 3.3.2

Frequently Used:

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

Available:

Type to search dependencies

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

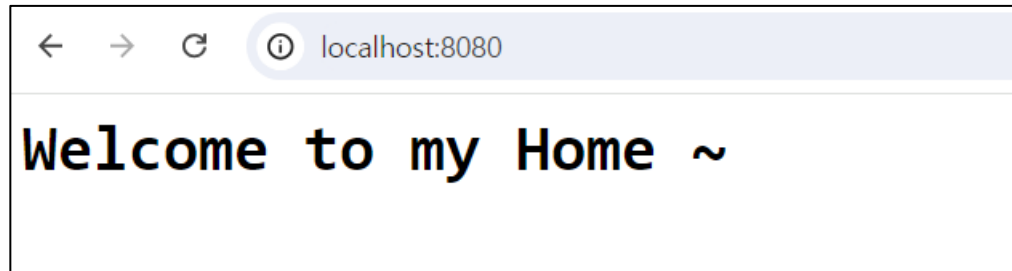
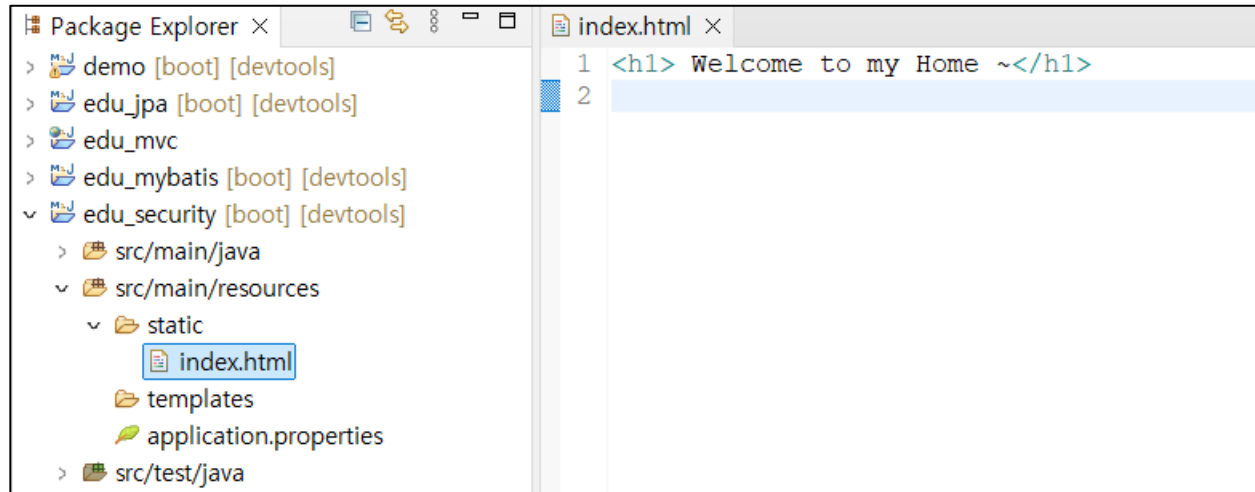
Selected:

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

Make Default Clear Selection

< Back Next > Finish Cancel

index.html 생성



Security 추가

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-security</artifactId>  
</dependency>
```

사용자 자동등록

- 사용자 아이디 : user
- 사용자 비밀번호 : 서버시작시 콘솔에 출력

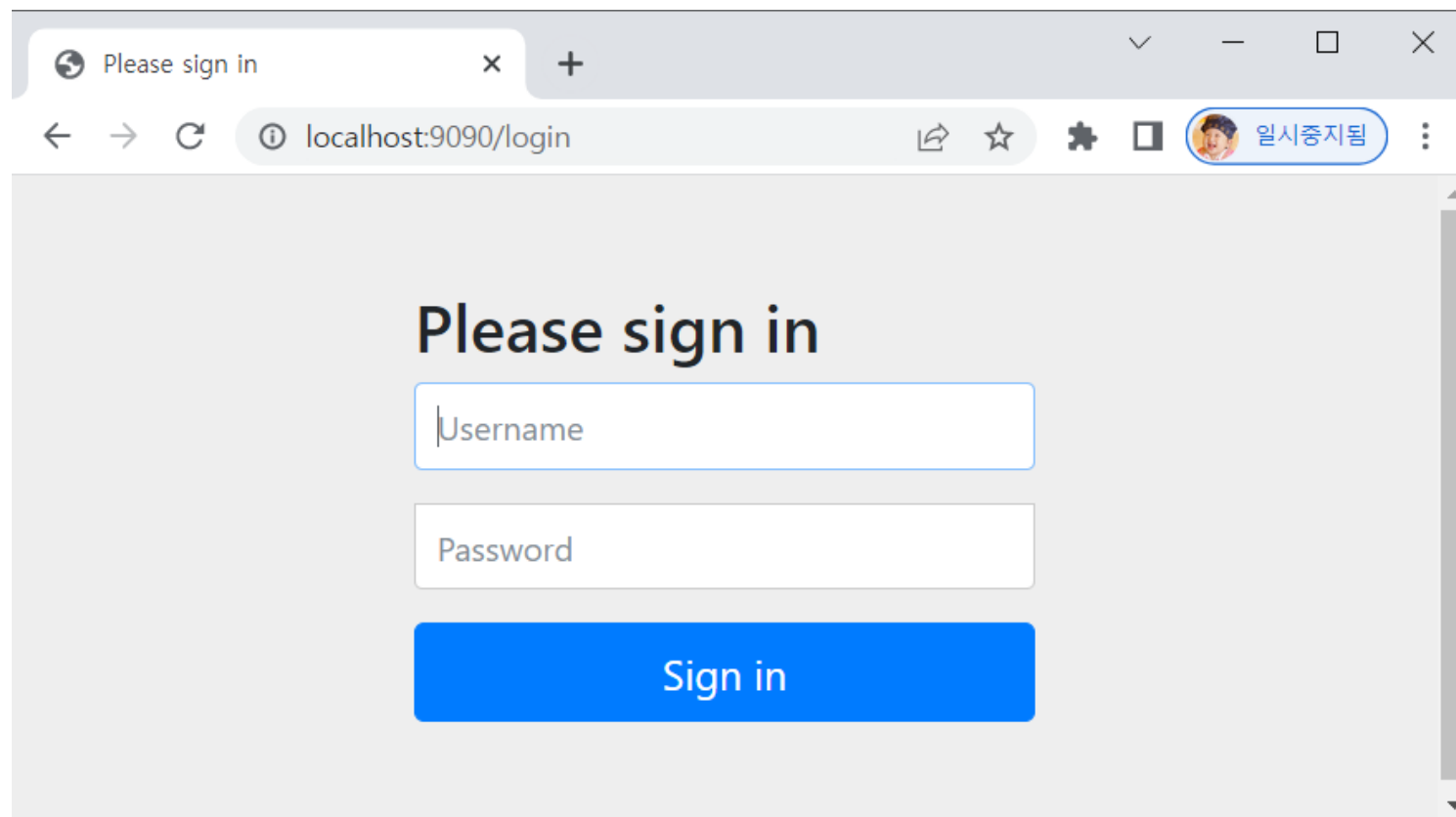
```
edu_security - EduSecurityApplication [Spring Boot App] C:\Program Files\Java\jdk-17\bin\javaw.exe (2024. 8. 15. 오후 10:17:05) [pid: 24500]
2024-08-15T22:17:11.056+09:00 INFO 24500 --- [edu_security] [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start complet
2024-08-15T22:17:11.074+09:00 INFO 24500 --- [edu_security] [ restartedMain] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2
2024-08-15T22:17:11.273+09:00 INFO 24500 --- [edu_security] [ restartedMain] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing Persis
2024-08-15T22:17:11.382+09:00 INFO 24500 --- [edu_security] [ restartedMain] org.hibernate.Version : HHH000412: Hibernate ORM cor
2024-08-15T22:17:11.447+09:00 INFO 24500 --- [edu_security] [ restartedMain] o.h.c.internal.RegionFactoryInitiator : HHH000026: Second-level cach
2024-08-15T22:17:11.904+09:00 INFO 24500 --- [edu_security] [ restartedMain] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ign
2024-08-15T22:17:12.437+09:00 INFO 24500 --- [edu_security] [ restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000489: No JTA platform a
2024-08-15T22:17:12.445+09:00 INFO 24500 --- [edu_security] [ restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManage
2024-08-15T22:17:12.510+09:00 WARN 24500 --- [edu_security] [ restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is e
2024-08-15T22:17:12.950+09:00 WARN 24500 --- [edu_security] [ restartedMain] .s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: cefb90a9-6534-4bfb-ale3-f9fb146b370f

This generated password is for development use only. Your security configuration must be updated before running your application in production.

2024-08-15T22:17:12.979+09:00 INFO 24500 --- [edu_security] [ restartedMain] r$InitializeUserDetailsManagerConfigurer : Global AuthenticationManager
2024-08-15T22:17:13.178+09:00 INFO 24500 --- [edu_security] [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running
2024-08-15T22:17:13.243+09:00 INFO 24500 --- [edu_security] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080
2024-08-15T22:17:13.259+09:00 INFO 24500 --- [edu_security] [ restartedMain] com.shop.EduSecurityApplication : Started EduSecurityApplicati
```

모든 요청 결과 페이지



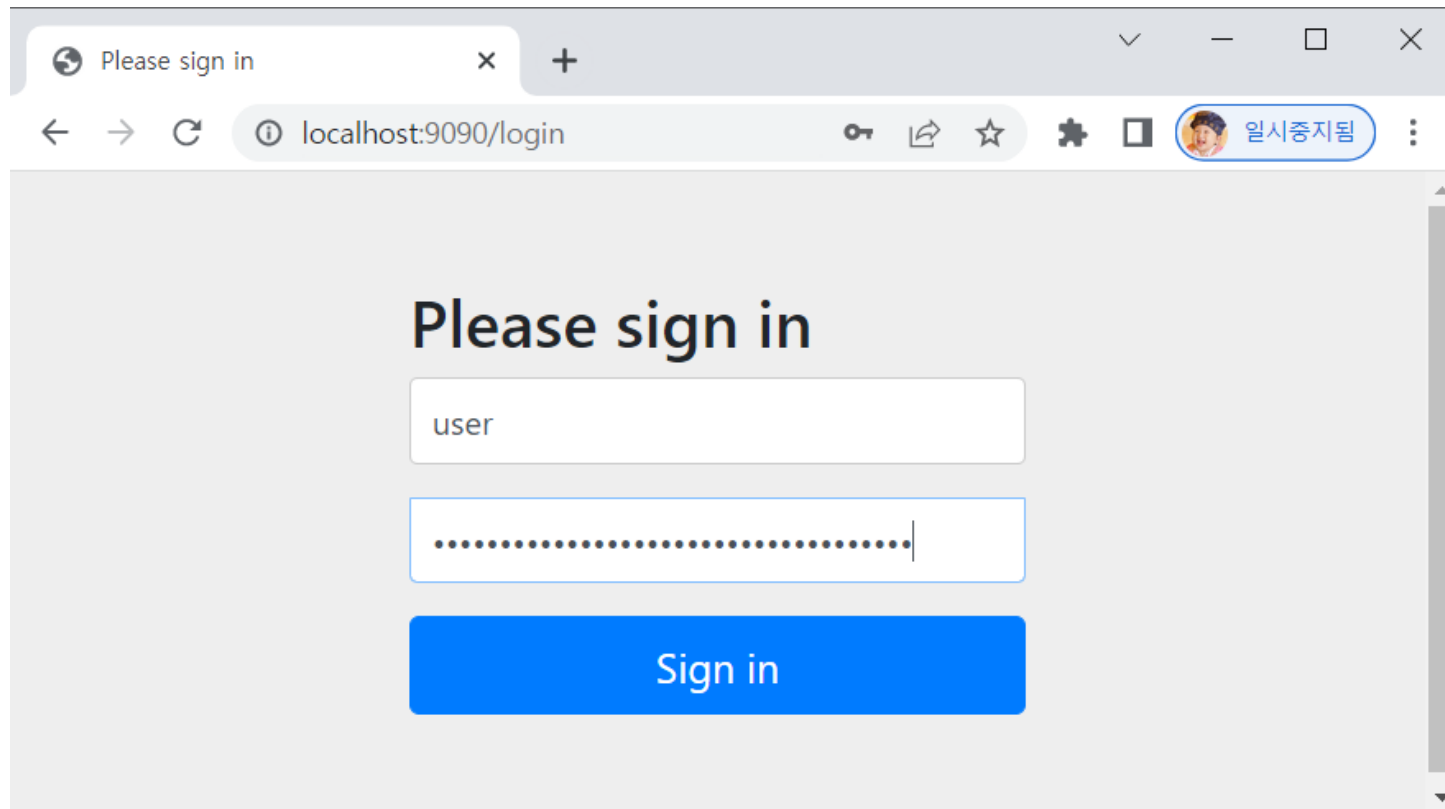
Please sign in

localhost:9090/login

일시중지됨

Please sign in

로그인



Please sign in

user

.....

Sign in

- 아이디 : user
- 비밀번호 : `console`창에서 확인

Spring Security 의존성 추가시 발생하는 기본 동작

- 서버가 기동되면 Spring Security 초기화 작업 및 보안 설정이 자동 처리됨
- 별도의 설정이나 구현을 하지 않아도 기본적인 웹 보안 기능이 현재 시스템에 연동되어 작동한다.
- 모든 요청은 인증이 되어야 자원 접근이 가능하다.
- 인증 방식으로는 기본적으로 폼 로그인 방식(Form Login)과 HTTP Basic 인증 방식(HttpBasic)을 제공한다.
- 기본 로그인 페이지를 자동으로 제공한다.
- 기본 계정이 한 개 제공되며, username: user / password: 자동 생성된 랜덤 문자열이 설정된다. 이 비밀번호는 서버 시작 시 콘솔에 출력된다.

문제점

- 계정 추가, 권한 추가, DB 연동 등의 작업이 필요할 수 있다.
- 기본적인 보안 기능 외에 시스템에서 필요로 하는 추가 보안 기능이 필요할 수 있다.

사용자 정의 보안 기능 구현

세부적인 보안 기능을 설정할 수 있는 API 제공

HttpSecurity

```
graph TD; HttpSecurity[HttpSecurity] --> 인증_API[인증 API]; HttpSecurity --> 인가_API[인가 API];
```

인증 API

- `.formLogin()`
- `.logout()`
- `.csrf()`
- `.httpBasic()`
- `.sessionManagement()`
- `.rememberMe()`
- `.exceptionHandling()`
- `.addFilter()`

인가 API

- `.authorizeHttpRequests()`
- `.requestMatchers()`
- `.hasRole()`
- `.permitAll()`
- `.authenticated()`
- `.fullyAuthentication()`
- `.access()`
- `.denyAll()`

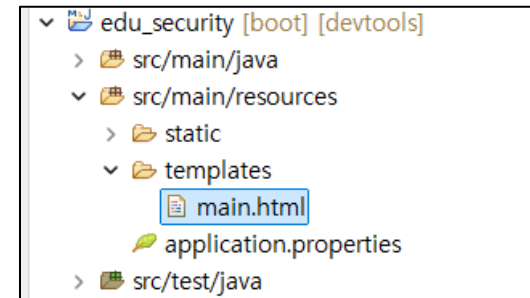
HttpSecurity

애플리케이션 자원에 대한 인증과 인가를 제어

메소드	사용할수 있는 메소드	의미
authorizeHttpRequest()		사용자 인증과 권한을 설정
	requestMatchers("URL")	매칭되는 Url 패턴들에 대한 접근 허용 permitAll() : 모든 사용자에게 접근 허용 authenticated() : 인증된 사용자만 접근 허용 hasRole(" 권한") : 특정 권한을 가진 사용자만 접근 허용
formLogin()		로그인 페이지 설정
	loginPage("/login")	로그인이 필요한 url로 접근하면 "login"화면으로 이동
logout()		로그아웃 페이지 설정
	logoutUrl("/logout")	로그아웃을 처리하는 페이지 설정
csrf()		csrf는 크로스사이트 위조 요청에 대한 설정
	csrf -> csrf.disable()	RESTfull을 사용하기 위해서는 csrf기능을 비활성화 해야 함

templates/main.html

```
<!DOCTYPE html>
<html xmlns:th="http://thymeleaf.org">
  <head>
    <meta charset="UTF-8">
    <title>main</title>
  </head>
  <body>
    <h1>Main Page</h1>
  </body>
</html>
```



com.shop.controller.MainController

```
package com.shop.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

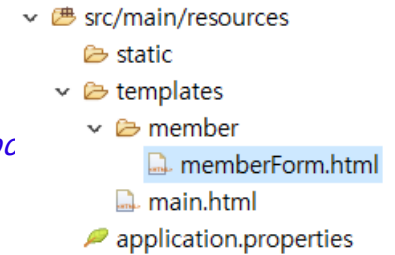
@Controller
public class MainController {

    @GetMapping("/")
    public String main() {
        return "main";
    }
}
```

```

<!DOCTYPE html>
<html xmlns:th="http://thymeleaf.org">
  <head>
    <meta charset="UTF-8">
    <title>회원 가입</title>
    <style type="text/css">
      table { width: 600px; border: 1px solid black; }
      .rh { background-color: orange; width: 80px; }
      th, td { border: 1px solid black; }
      input[type="text"], input[type="password"] { width: 100%; box-sizing: border-bc
    </style>
  </head>
  <body th:align="center">
    <h1>회원 가입</h1>
    <form action="/member/new" method="post" >
      <table th:align="center">
        <tr>
          <td class="rh">이름</td>
          <td><input type="text" name="name"/></td>
        </tr>
        <tr>
          <td class="rh">메일주소</td>
          <td><input type="text" name="email" /></td>
        </tr>
        <tr>
          <td class="rh">비밀번호</td>
          <td><input type="password" name="password" /></td>
        </tr>
        <tr>
          <td class="rh">주소</td>
          <td><input type="text" name="address"/></td>
        </tr>
        <tr>
          <td colspan="2"><input type="submit" value="회원 가입" /></td>
        </tr>
      </table>
    </form>
  </body>
</html>

```



com.shop.controller.MemberController

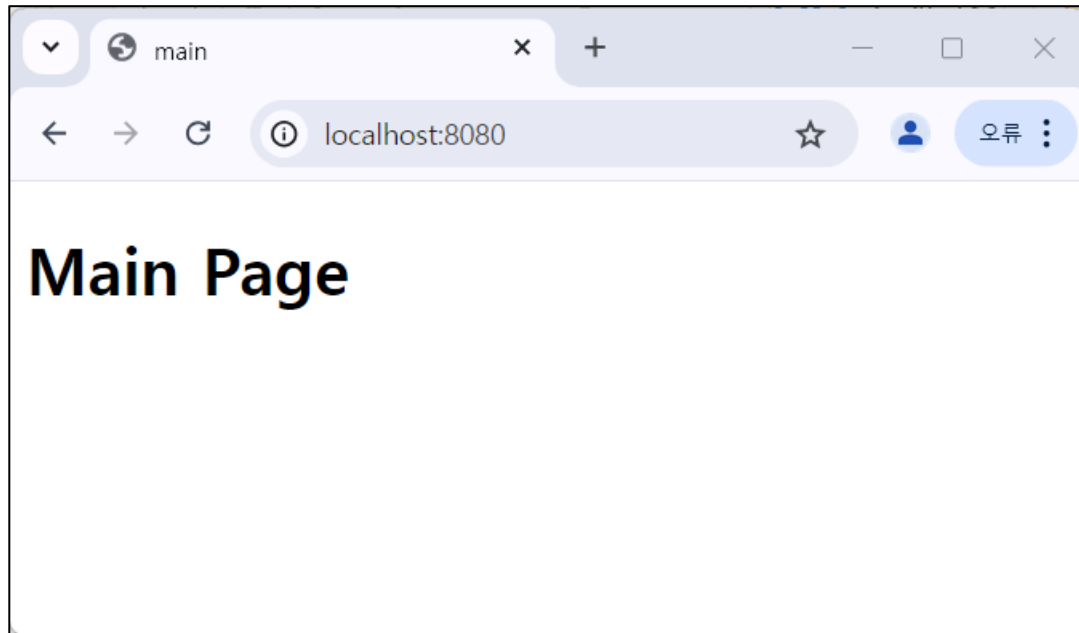
```
package com.shop.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

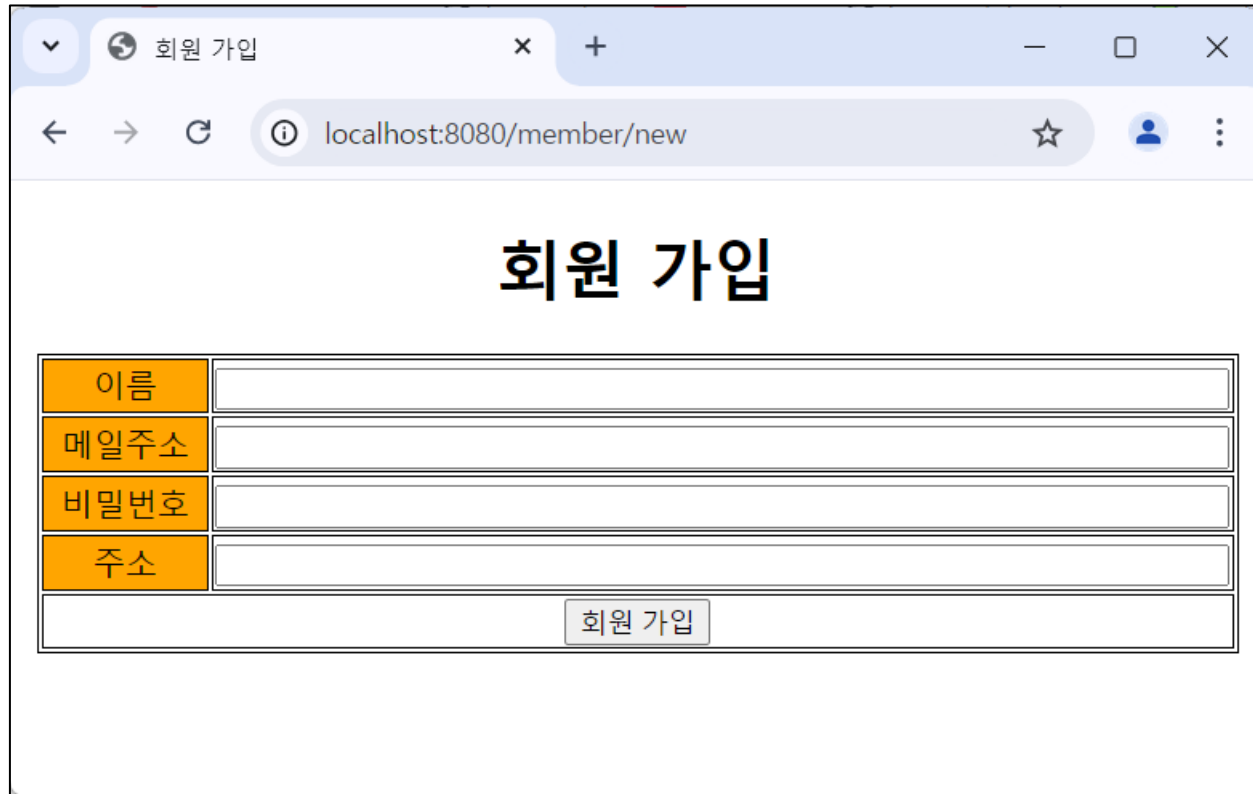
@Controller
@RequestMapping( "/member" )
public class MemberController {

    @GetMapping( "/new" )
    public String memberForm() {
        return "/member/memberForm";
    }
}
```

http://localhost:8080/



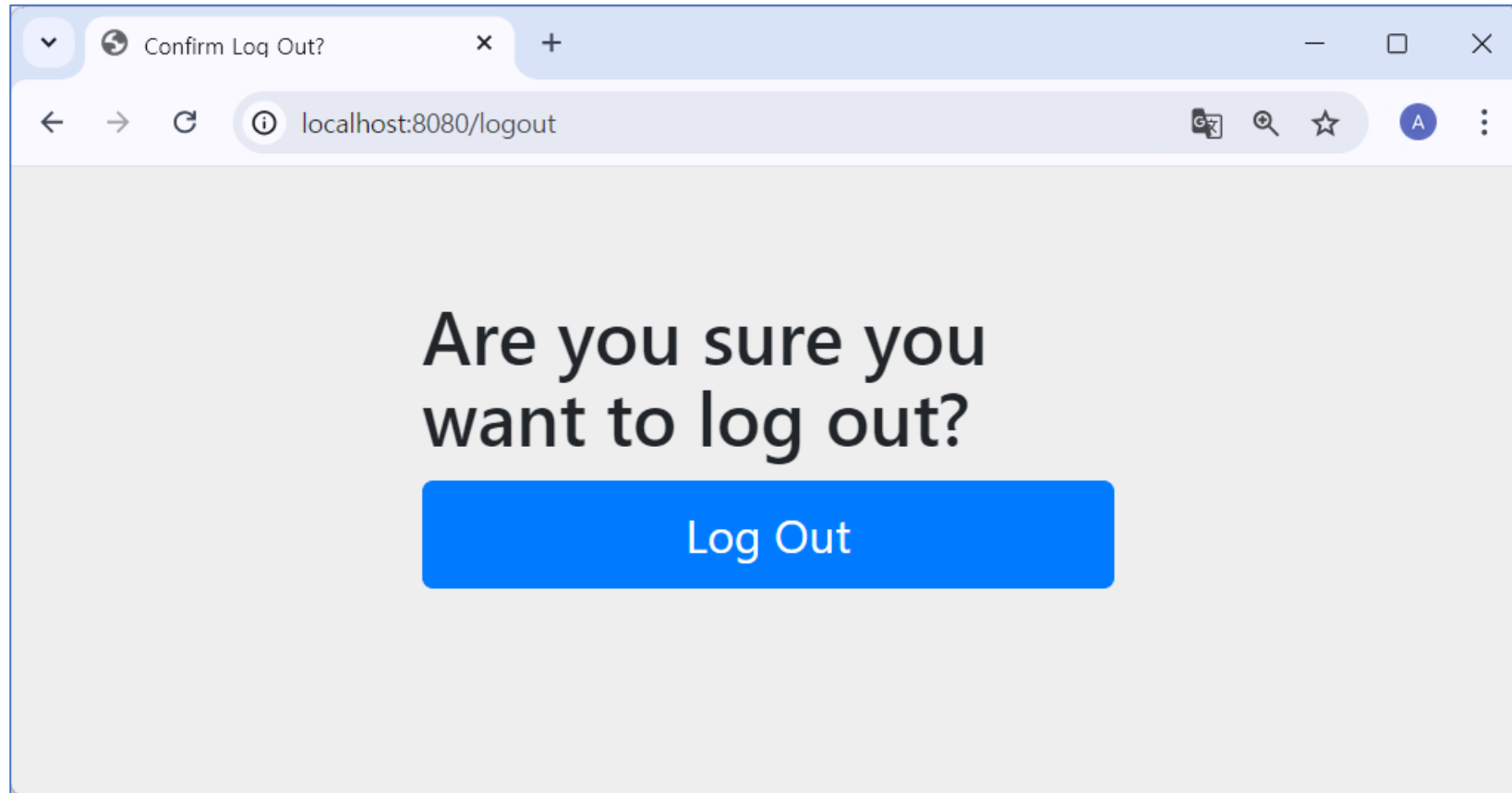
http://localhost:8080/member/new



The screenshot shows a web browser window with the title '회원 가입' and the address bar displaying 'localhost:8080/member/new'. The main content area features a large heading '회원 가입' and a sign-up form. The form consists of four input fields with labels in orange boxes: '이름' (Name), '메일주소' (Email Address), '비밀번호' (Password), and '주소' (Address). Below these fields is a '회원 가입' (Member Sign-up) button.

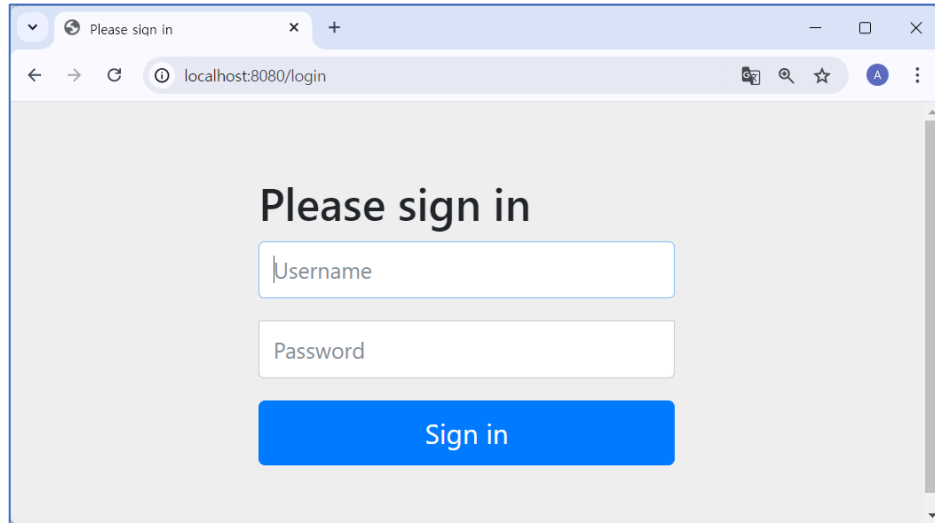
이름	<input type="text"/>
메일주소	<input type="text"/>
비밀번호	<input type="text"/>
주소	<input type="text"/>

로그 아웃 - http://localhost:8080/logout



로그아웃 후 접속

- <http://localhost:8080/>
- <http://localhost:8080/member/new>
- 모든 접속 로그인 페이지로 redirect



com.shop.config.AppConfig

```
package com.shop.config;
```

```
import org.modelmapper.ModelMapper;
```

```
import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.Configuration;
```

```
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
```

```
import org.springframework.security.crypto.password.PasswordEncoder;
```

```
@Configuration
```

```
public class AppConfig {
```

```
    @Bean
```

```
    public ModelMapper modelMapper() {
```

```
        return new ModelMapper();
```

```
    }
```

```
    @Bean
```

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

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

```
    }
```

```
}
```

```
<dependency>
```

```
    <groupId>org.modelmapper.extensions</groupId>
```

```
    <artifactId>modelmapper-spring</artifactId>
```

```
    <version>3.1.1</version>
```

```
</dependency>
```

com.shop.config.SecurityConfig

```
package com.shop.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(authorize -> authorize
                .requestMatchers("/", "/member/new").permitAll()
                .anyRequest().authenticated());

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

회원 가입

Spring Security

```
package com.shop.constant;

public enum Role {
    USER, ADMIN
}
```

```
package com.shop.dto;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class MemberFormDto {
    private String name;
    private String email;
    private String password;
    private String address;
}
```

```
package com.shop.entity;
```

```
@Entity
```

```
@Table(name = "member2")
```

```
@Getter
```

```
@Setter
```

```
@ToString
```

```
public class Member {
```

```
    @Id
```

```
    @Column(name = "member_id")
```

```
    @GeneratedValue(strategy = GenerationType.AUTO)
```

```
    private Long id;
```

```
    private String name;
```

```
    @Column(unique = true)
```

```
    private String email;
```

```
    private String password;
```

```
    private String address;
```

```
    @Enumerated(EnumType.STRING)
```

```
    private Role role;
```

```
}
```



```
package com.shop.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.shop.entity.Member;

public interface MemberRepository extends JpaRepository<Member, Long> {
    Member findByEmail(String email);
}
```

```
package com.shop.service;
```

```
@Service
```

```
@Transactional
```

```
@RequiredArgsConstructor
```

```
public class MemberService {
```

```
    private final MemberRepository memberRepository;
```

```
    @Autowired
```

```
    private PasswordEncoder passwordEncoder;
```

```
    @Autowired
```

```
    private ModelMapper modelMapper;
```

```
    public Member saveMember(MemberFormDto memberFormDto){
```

```
        Member member = modelMapper.map(memberFormDto, Member.class);
```

```
        String password = passwordEncoder.encode(memberFormDto.getPassword());
```

```
        member.setPassword(password);
```

```
        member.setRole(Role.USER);
```

```
        validateDuplicateMember(member);
```

```
        return memberRepository.save(member);
```

```
    }
```

```
    private void validateDuplicateMember(Member member) {
```

```
        Member findMember = memberRepository.findByEmail(member.getEmail());
```

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

```
            throw new IllegalArgumentException("이미 가입된 회원입니다.");
```

```
        }
```

```
    }
```

```
}
```

```
package com.shop.service;

import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.security.crypto.password.PasswordEncoder;
import com.shop.constant.Role;
import com.shop.dto.MemberFormDto;
import com.shop.entity.Member;
```

```
@SpringBootTest
```

```
public class MemberServiceTest {
```

```
    @Autowired
```

```
    private MemberService memberService;
```

```
    @Autowired
```

```
    private PasswordEncoder passwordEncoder;
```

```
    private MemberFormDto createMember(String email) {
        MemberFormDto memberFormDto = new MemberFormDto();
        memberFormDto.setEmail(email);
        memberFormDto.setName("홍길동");
        memberFormDto.setAddress("서울 강남구 언주로");
        memberFormDto.setPassword("1234");
        return memberFormDto;
    }
```

```

@Test
@DisplayName("회원 가입 테스트")
public void saveMemberTest() {
    // Given: 테스트에 필요한 데이터 생성
    MemberFormDto memberFormDto = createMember("test@email.com");

    // When: 회원을 저장하는 메서드 호출
    Member savedMember = memberService.saveMember(memberFormDto);

    // Then: 저장된 회원의 필드가 예상대로 설정되었는지 확인
    assertMemberDetails(savedMember, memberFormDto);
}

private void assertMemberDetails(Member savedMember, MemberFormDto memberFormDto) {
    assertThat(savedMember.getEmail()).isEqualTo(memberFormDto.getEmail());
    assertThat(savedMember.getName()).isEqualTo(memberFormDto.getName());
    assertThat(savedMember.getAddress()).isEqualTo(memberFormDto.getAddress());
    assertThat(passwordEncoder.matches("1234", savedMember.getPassword())).isTrue();
    assertThat(savedMember.getRole()).isEqualTo(Role.USER);
}

```

```

@Test
@DisplayName("중복 회원 가입 테스트")
public void saveDuplicateMemberTest() {
    // Given: 동일한 이메일로 두 명의 회원 생성
    MemberFormDto firstMember = createMember("test2@email.com");
    MemberFormDto duplicateMember = createMember("test2@email.com");

    // When: 첫 번째 회원을 저장
    memberService.saveMember(firstMember);

    // Then: 두 번째 회원을 저장 시도할 때 중복 회원 예외가 발생하는지 확인
    IllegalArgumentException exception = assertThrows(IllegalArgumentException.class, () -> {
        memberService.saveMember(duplicateMember);
    });

    // 예외 메시지가 예상한 대로인지 검증
    assertEquals("이미 가입된 회원입니다.", exception.getMessage());
}
}

```

```
package com.shop.controller;
```

~ 생략 ~

```
@Controller
```

```
@RequestMapping( "/member" )
```

```
@RequiredArgsConstructor
```

```
public class MemberController {
```

```
    private final MemberService memberService;
```

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

```
    public String memberForm( ) {  
        return "/member/memberForm";  
    }
```

```
    @PostMapping( "/new" )
```

```
    public String memberSave(MemberFormDto memberFormDto) {  
        memberService.saveMember(memberFormDto);  
        return "redirect:/";  
    }
```

```
}
```

com.shop.config.SecurityConfig

```
package com.shop.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(authorize -> authorize
                .requestMatchers("/", "/member/new").permitAll()
                .anyRequest().authenticated());

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

```
<input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}" />
```

회원 가입

이름	<input type="text"/>
메일주소	<input type="text"/>
비밀번호	<input type="text"/>
주소	<input type="text"/>



회원 가입

이름	홍길동
메일주소	hong@kt.com
비밀번호
주소	경기도 성남시 분당구



main

localhost:8080

Main Page

Validation

Spring Security

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-validation</artifactId>  
</dependency>
```

```

package com.shop.dto;

import org.hibernate.validator.constraints.Length;

import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotEmpty;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Getter @Setter @ToString
public class MemberFormDto {
    @NotBlank(message = "이름은 필수 입력 값입니다.")
    private String name;

    @NotEmpty(message = "이메일은 필수 입력 값입니다.")
    @Email(message = "이메일 형식으로 입력해주세요")
    private String email;

    @NotEmpty(message = "비밀번호는 필수 입력 값입니다.")
    @Length(min = 8, max = 16, message = "비밀번호는 8자이상, 16자 이하로 입력해주세요")
    private String password;

    @NotEmpty(message = "주소는 필수 입력 값입니다.")
    private String address;
}

```

```

package com.shop.controller;

@Controller
@RequestMapping("/member")
@RequiredArgsConstructor
public class MemberController {

    private final MemberService memberService;

    @GetMapping("/new")
    public String memberForm(MemberFormDto memberFormDto) {
        return "/member/memberForm";
    }

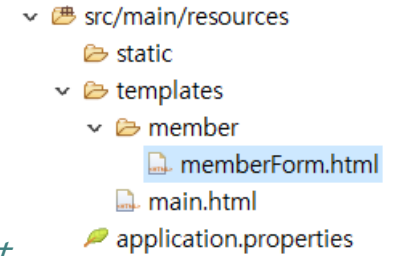
    @PostMapping("/new")
    public String memberSave(@Valid MemberFormDto memberFormDto, BindingResult br, Model model) {
        if (br.hasErrors()) {
            return "/member/memberForm";
        }
        try {
            memberService.saveMember(memberFormDto);
        } catch (Exception e) {
            model.addAttribute("errorMessage", e.getMessage());
            return "/member/memberForm";
        }
        return "redirect:/";
    }
}

```

```

<body th:align="center">
    <h1>회원 가입</h1>
    <p th:text="${errorMessage}"></p>
    <!-- 에러 메시지 표시 -->
    <form action="/member/new" method="post" th:object="${memberFormDto}">
        <table th:align="center">
            <tr>
                <td class="rh">이름</td>
                <td><input type="text" th:field="*{name}" />
                <p th:if="${#fields.hasErrors('name')}" th:errors="*{name}"></p></td>
            </tr>
            <tr>
                <td class="rh">메일주소</td>
                <td><input type="text" th:field="*{email}" />
                <p th:if="${#fields.hasErrors('email')}" th:errors="*{email}"></p>
                </td>
            </tr>
            <tr>
                <td class="rh">비밀번호</td>
                <td><input type="password" th:field="*{password}" />
                <p th:if="${#fields.hasErrors('password')}" th:errors="*{password}"></p>
                </td>
            </tr>
            <tr>
                <td class="rh">주소</td>
                <td><input type="text" th:field="*{address}" />
                <p th:if="${#fields.hasErrors('address')}" th:errors="*{address}"></p>
                </td>
            </tr>
            <tr>
                <td colspan="2"><input type="submit" value="회원 가입" /></td>
            </tr>
        </table>
        <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}" />
    </form>
</body>

```



com.shop.config.SecurityConfig

```
package com.shop.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            // .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(authorize -> authorize
                .requestMatchers("/", "/member/new").permitAll()
                .anyRequest().authenticated());

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

회원 가입

이름	<input type="text"/>
메일주소	<input type="text"/>
비밀번호	<input type="password"/>
주소	<input type="text"/>

회원 가입

이름	이름은 필수 입력 값입니다.
메일주소	이메일은 필수 입력 값입니다.
비밀번호	비밀번호는 8자이상, 16자 이하로 입력해주세요. 비밀번호는 필수 입력 값입니다.
주소	주소는 필수 입력 값입니다.

회원 가입

localhost:8080/member/new

회원 가입

이미 가입된 회원입니다.

이름	홍길동
메일주소	test@email.com
비밀번호	
주소	서울

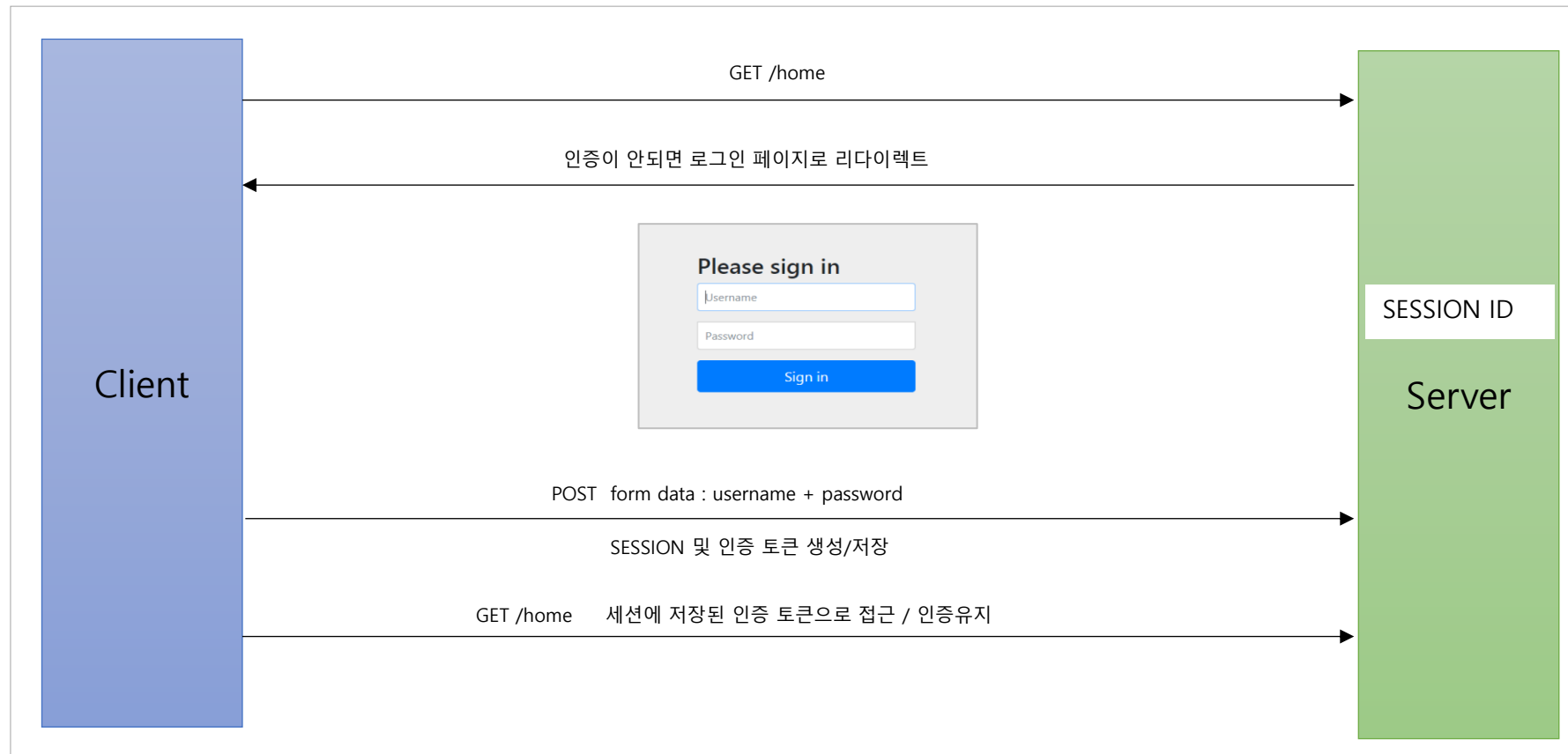
회원 가입

로그인/아웃

Spring Security

인증 API - Form 인증

Login

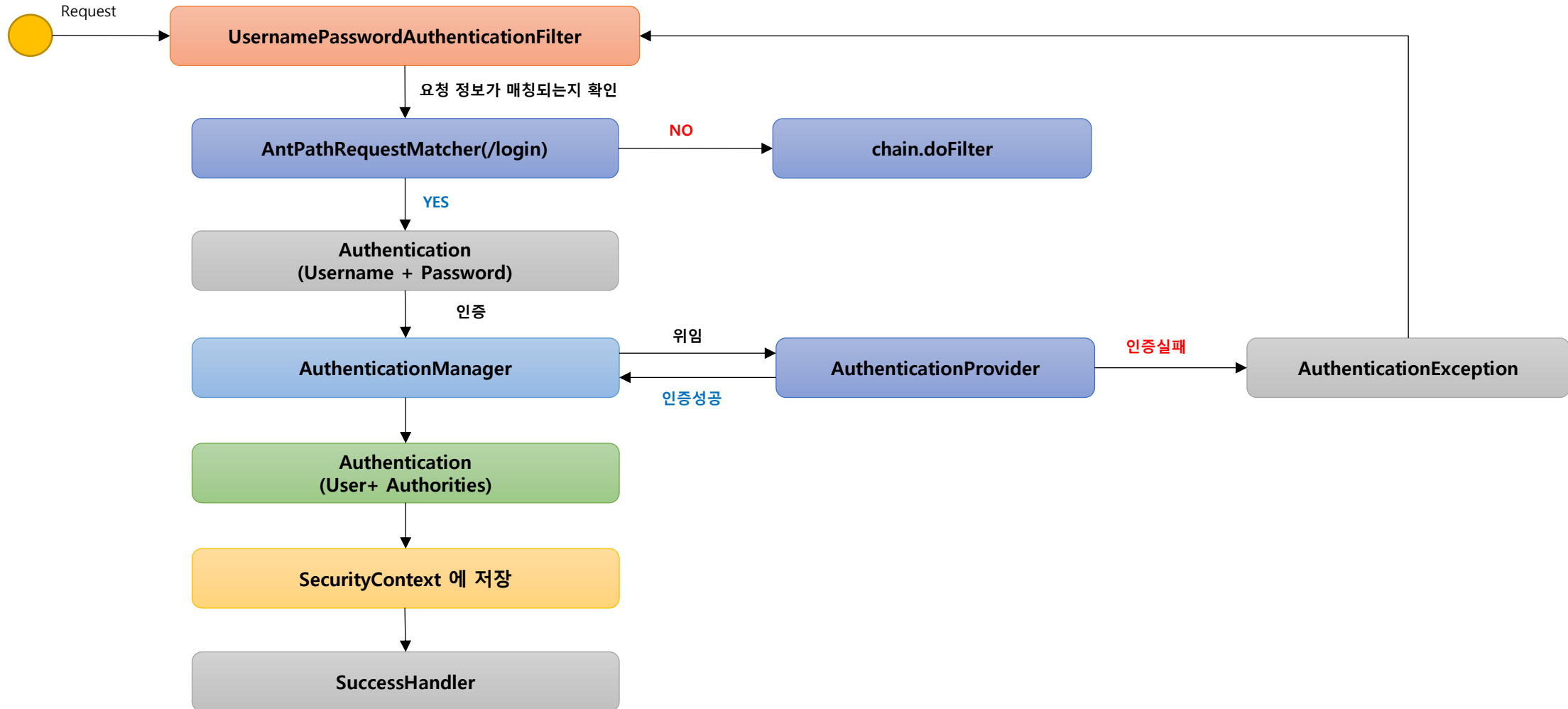


인증 API - Form Login 인증

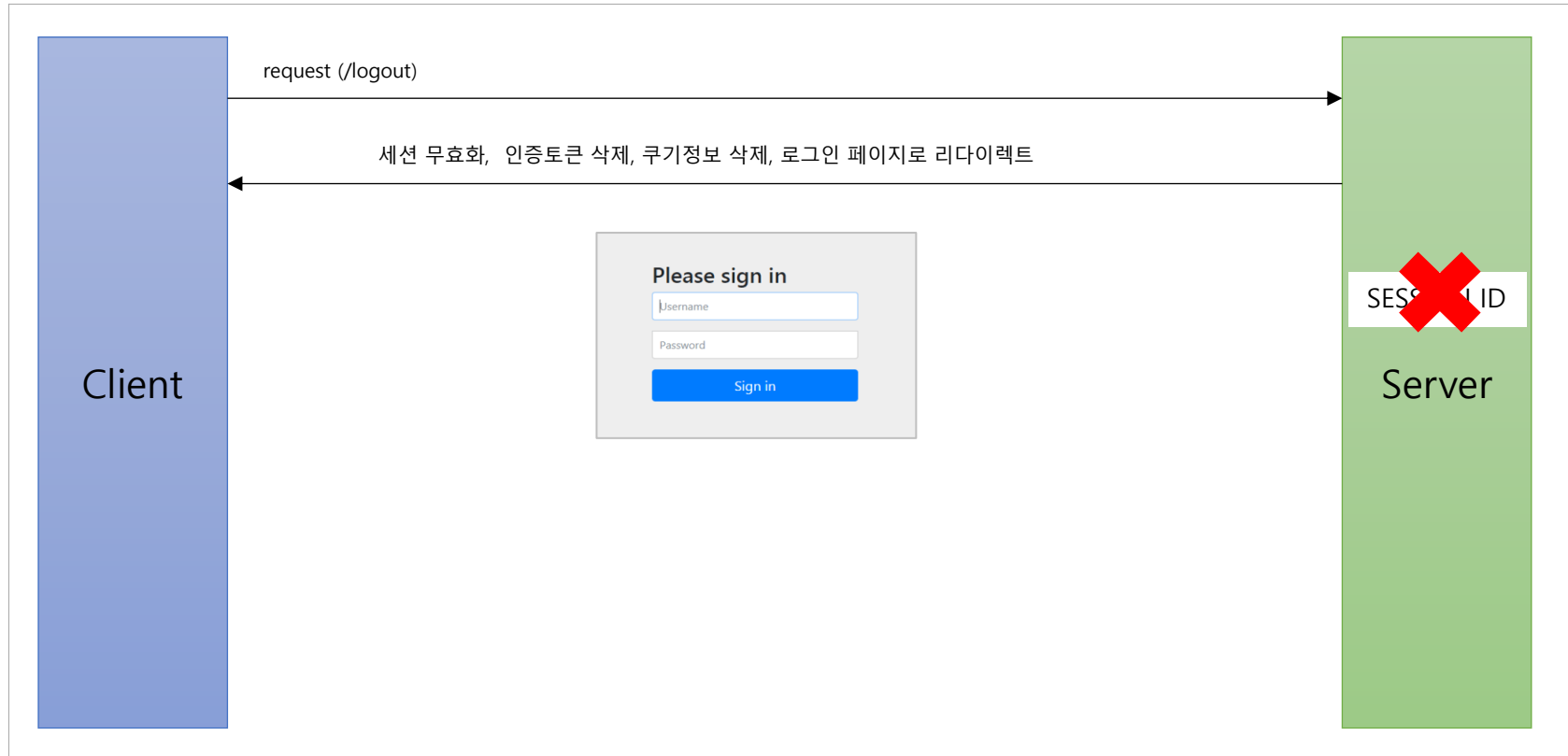
HttpSecurity.formLogin() : Form 로그인 인증 기능이 작동함

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .formLogin(formLogin -> formLogin
            .loginPage("/login.html")           // 사용자 정의 로그인 페이지
            .defaultSuccessUrl("/home")         // 로그인 성공 후 이동할 페이지
            .failureUrl("/login.html?error=true") // 로그인 실패 후 이동할 페이지
            .usernameParameter("username")      // 아이디 파라미터명 설정
            .passwordParameter("password")      // 비밀번호 파라미터명 설정
            .loginProcessingUrl("/login")        // 로그인 Form Action URL
            .successHandler(loginSuccessHandler()) // 로그인 성공 후 핸들러
            .failureHandler(loginFailureHandler()) // 로그인 실패 후 핸들러
        );
    return http.build();
}
```

인증 API – Login Form 인증



인증 API - Logout



인증 API - Logout

HttpSecurity.logout() : 로그아웃 기능이 작동함

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .logout(logout -> logout
            .logoutUrl("/logout") // 로그아웃 처리 URL
            .logoutSuccessUrl("/login") // 로그아웃 성공 후 이동할 페이지
            .deleteCookies("JSESSIONID", "remember-me") // 로그아웃 시 삭제할 쿠키
            .addLogoutHandler(logoutHandler()) // 로그아웃 핸들러 추가
            .logoutSuccessHandler(logoutSuccessHandler()) // 로그아웃 성공 핸들러 추가
        );
    return http.build();
}
```

인증 API - Logout

HttpSecurity.logout() : 로그아웃 기능이 작동함

```
@Bean
public LogoutHandler logoutHandler() {
    return (request, response, authentication) -> {
        // 로그아웃 핸들러 로직 구현
    };
}
```

```
@Bean
public LogoutSuccessHandler logoutSuccessHandler() {
    return (request, response, authentication) -> {
        // 로그아웃 성공 후 처리 로직 구현
        response.sendRedirect("/login");
    };
}
```

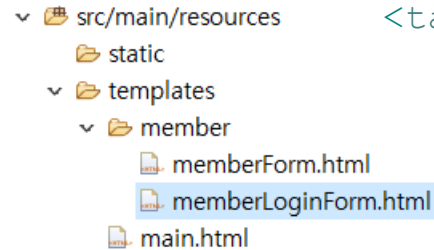
memberLoginForm.html

```
<!DOCTYPE html>
<html xmlns:th="http://thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>회원 가입</title>
  <style type="text/css">
    table {
      width: 600px;
      border: 1px solid black,
    }

    .rh {
      background-color: orange;
      width: 80px;
    }

    th, td {
      border: 1px solid black;
    }

    input[type="text"], input[type="password"] {
      width: 100%;
      box-sizing: border-box;
    }
  </style>
</head>
```



```
<body th:align="center">
  <h1>로그인</h1>
  [[${loginErrorMsg}]]
  <form action="/member/login" method="post">
    <input type="hidden" th:name="${_csrf.parameterName}"
      th:value="${_csrf.token}" />
    <table th:align="center">
      <tr>
        <td class="rh">이메일</td>
        <td><input type="text" name="email" />
      </tr>
      <tr>
        <td class="rh">비밀번호</td>
        <td><input type="password" name="password" />
      </tr>
      <tr>
        <td colspan="2"><input type="submit" value="로그인" />
          <button type="button"
            onClick="location.href='/member/new'">회원가입</button>
        </td>
      </tr>
    </table>
  </form>
</body>
</html>
```

```

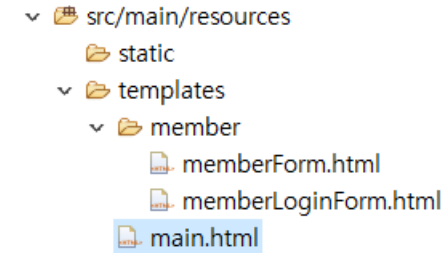
<!DOCTYPE html>
<html xmlns:th="http://thymeleaf.org">
  <head>
    <meta charset="UTF-8">
    <title>main</title>
  </head>
  <body>
    <h1>Main Page</h1>

    <a href="/member/new">회원 가입 </a>
    <br>
    <a href="/member/login">로그인 </a>
    <br>

    <form action="/member/logout" method="post">
      <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}" />
      <input type="submit" value="로그아웃" />
    </form>

  </body>
</html>

```




```
package com.shop.controller;
```

~ 생략 ~

```
@Controller
```

```
@RequestMapping("/member")
```

```
@RequiredArgsConstructor
```

```
public class MemberController {
```

```
private final MemberService memberService;
```

~ 생략 ~

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

```
public String loginMember() {  
    return "/member/memberLoginForm";  
}
```

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

```
public String loginError(Model model) {  
    model.addAttribute("loginErrorMsg", "아이디 또는 비밀번호를 확인해주세요.");  
    return "/member/memberLoginForm";  
}
```

```
}
```

```
package com.shop.config;
```

```
@Configuration
```

```
@EnableWebSecurity
```

```
public class SecurityConfig {
```

```
    @Bean
```

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

```
        http
```

```
            .authorizeHttpRequests(authz -> authz
```

```
                .requestMatchers("/", "/member/new", "/member/login").permitAll()
```

```
                .anyRequest().authenticated()
```

```
            )
```

```
            .formLogin(form -> form
```

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

```
                .defaultSuccessUrl("/")
```

```
                .usernameParameter("email")
```

```
                .failureUrl("/member/login/error")
```

```
            )
```

```
            .logout(logout -> logout
```

```
                .logoutUrl("/member/logout")
```

```
                .logoutSuccessUrl("/")
```

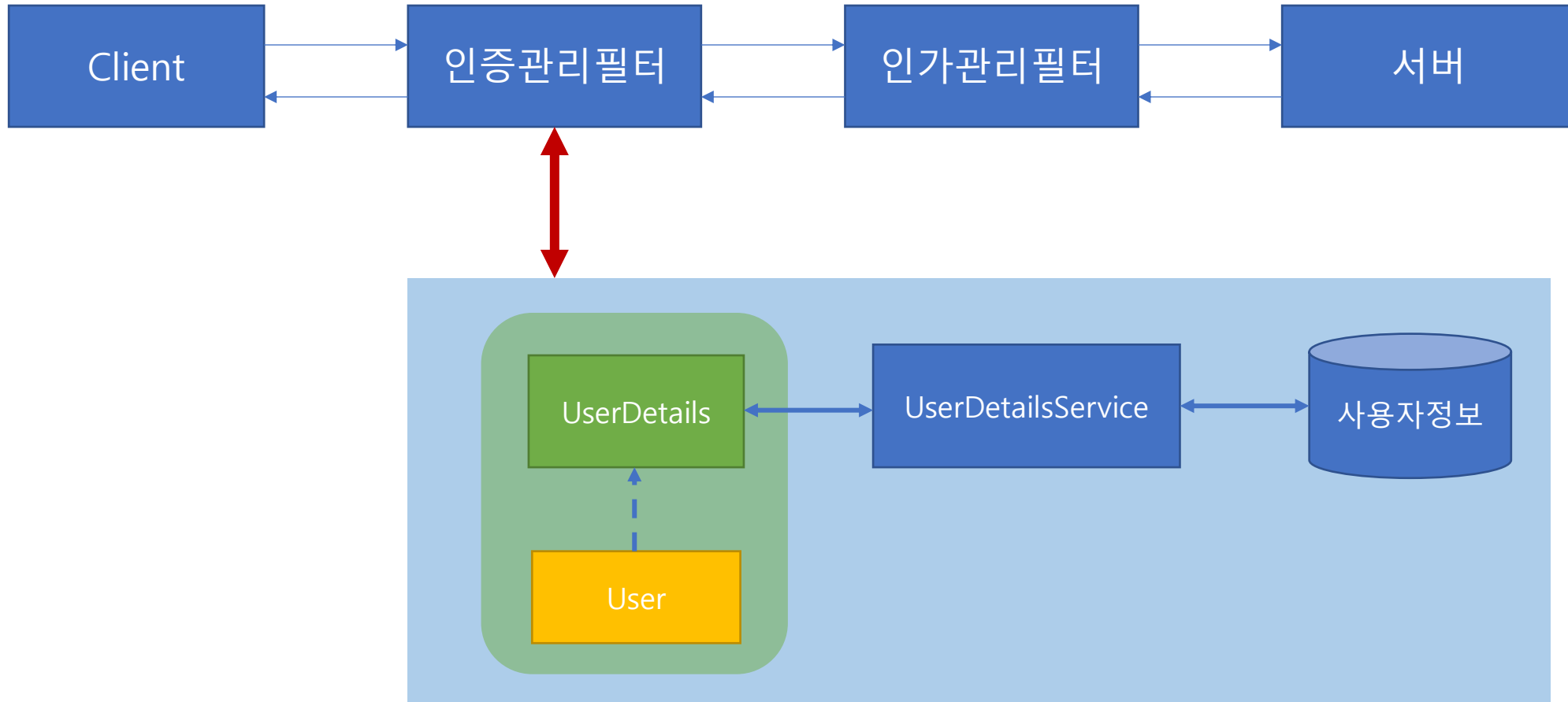
```
            );
```

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

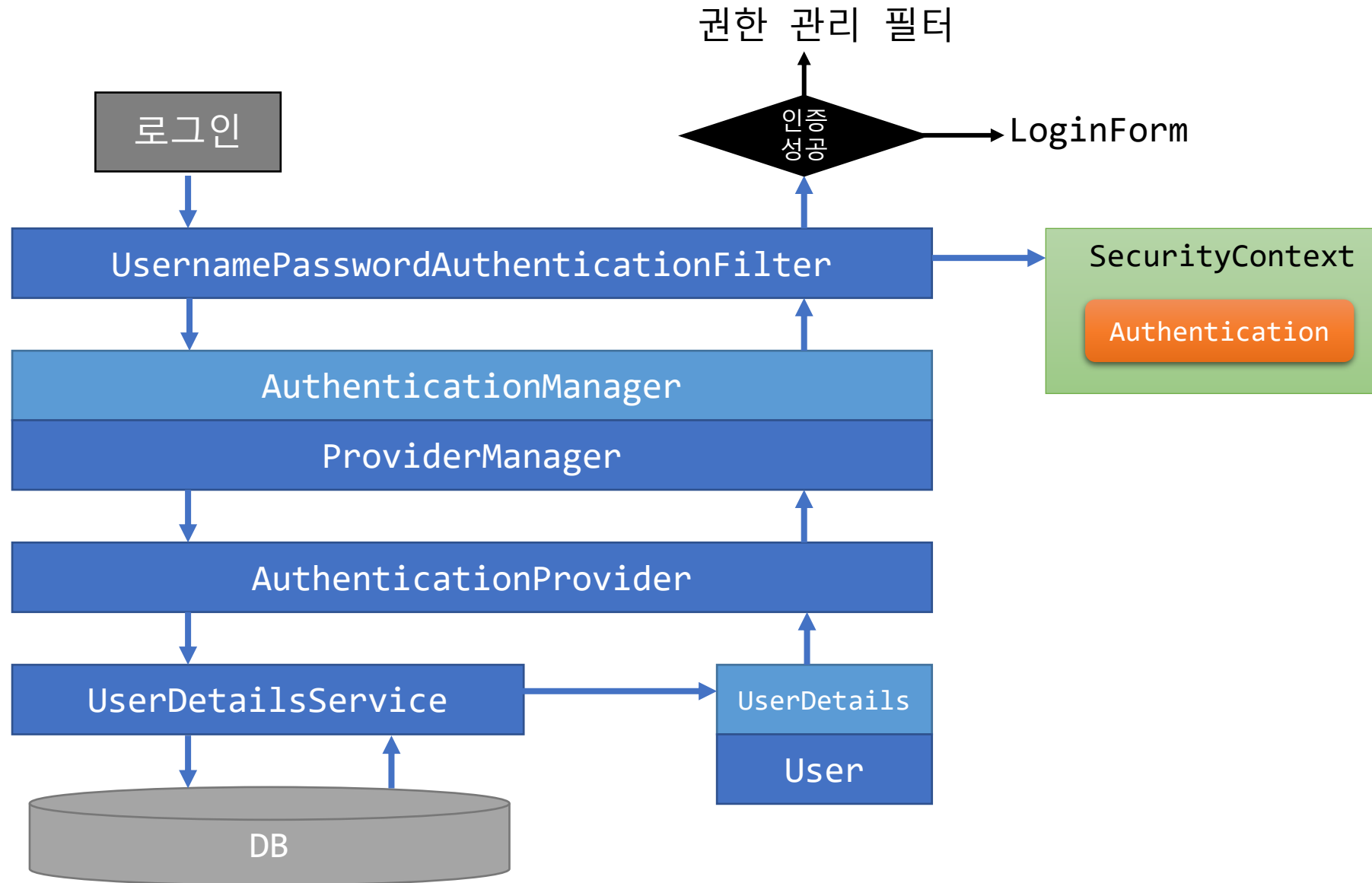
```
    }
```

```
}
```

Spring Security 동작 원리



인증 관리 필터



Authentication

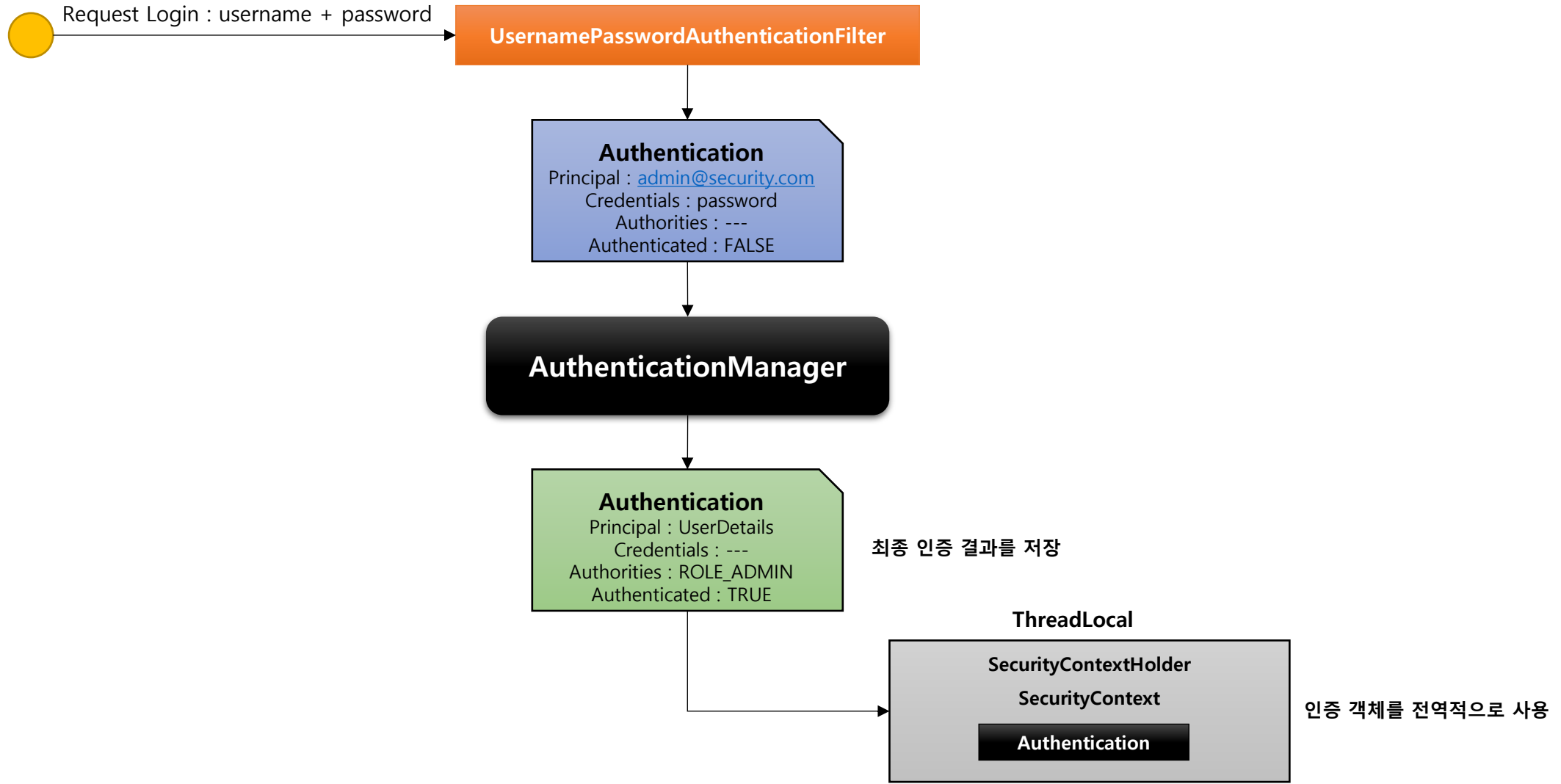
당신이 누구인지 증명하는 것

- 사용자의 인증 정보를 저장하는 토큰 개념
- 인증시 id 와 password 를 담고 인증 검증을 위해 전달되어 사용
- 인증후 최종 인증 결과 (user 객체, 권한정보) 를 담고 SecurityContext 에 저장되어 전역적으로 참조

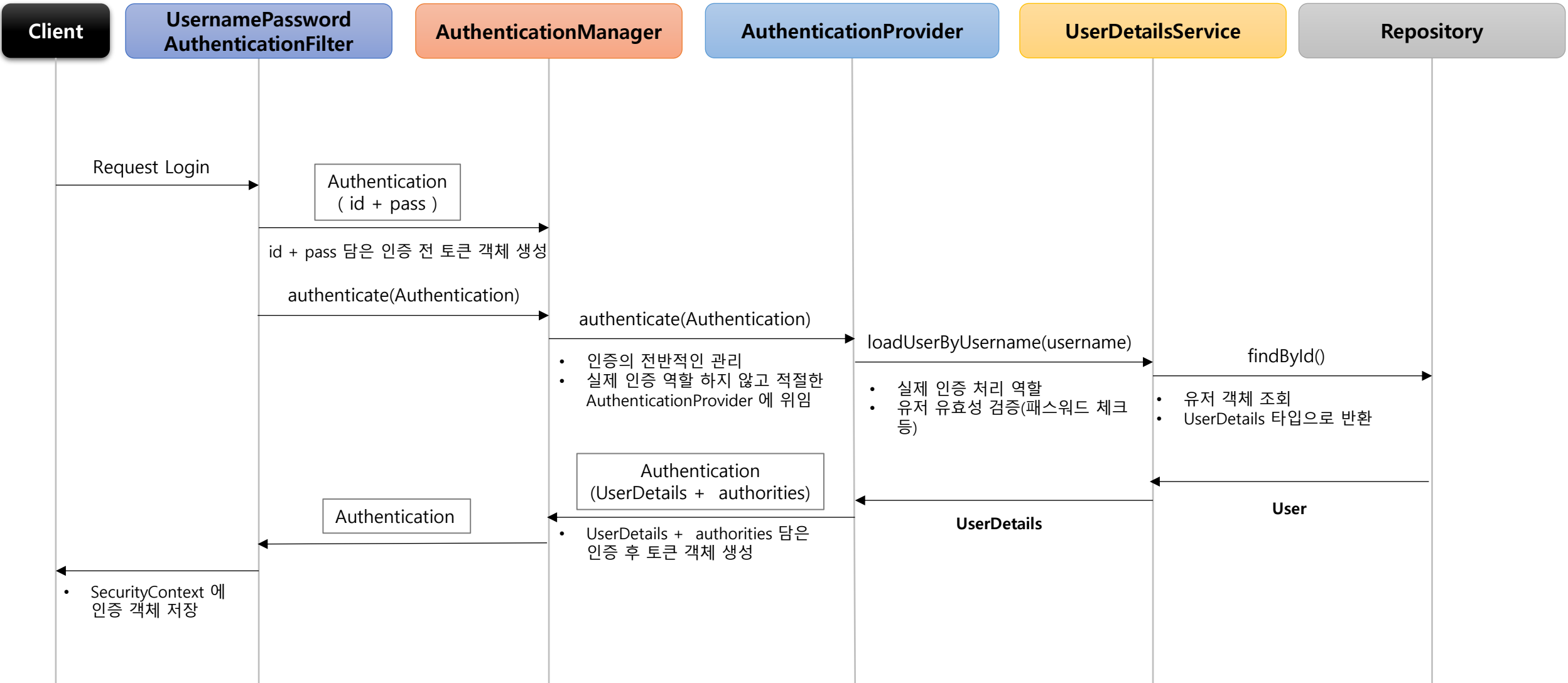
Authentication authentication = SecurityContextHolder.getContext().getAuthentication()

• 구조

- 1.Principal : 사용자 ID 또는 User 객체를 저장
- 2.Credentials : 사용자 비밀번호
- 3.Authorities : 인증된 사용자의 권한 목록
- 4.Details : 인증 부가 정보
- 5.Authenticated : 인증 여부



Authentication Flow



```
package com.shop.service;
```

```
~ 생략 ~
```

```
import org.springframework.security.core.userdetails.User;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;
```

```
@Service
```

```
@Transactional
```

```
@RequiredArgsConstructor
```

```
public class MemberService implements UserDetailsService{
```

```
~ 생략 ~
```

```
@Override
```

```
public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {  
    Member member = memberRepository.findByEmail(email);  
    if(member == null)  
        throw new UsernameNotFoundException(email);  
    return User.builder()  
        .username(member.getEmail())  
        .password(member.getPassword())  
        .roles(member.getRole().toString())  
        .build();  
}
```

```
}
```



```
package com.shop.config;
```

```
@Configuration
```

```
@EnableWebSecurity
```

```
public class SecurityConfig {
```

```
    private final MemberService memberService;
```

```
    public SecurityConfig(MemberService memberService) {
```

```
        this.memberService = memberService;
```

```
    }
```

```
@Bean
```

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

```
    http
```

```
        .authorizeHttpRequests(authz -> authz
```

```
            .requestMatchers("/", "/member/new", "/member/login").permitAll()
```

```
            .anyRequest().authenticated()
```

```
        )
```

```
        .formLogin(form -> form
```

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

```
            .defaultSuccessUrl("/")
```

```
            .usernameParameter("email")
```

```
            .failureUrl("/member/login/error")
```

```
        )
```

```
        .logout(logout -> logout
```

```
            .logoutUrl("/member/logout")
```

```
            .logoutSuccessUrl("/")
```

```
        )
```

```
        .userDetailsService(memberService);
```

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

```
}
```

```
}
```

1. http://localhost:8080/member/login 접속

로그인

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

2. 이메일과 비밀번호 입력

3. [로그인] 버튼 클릭

4. 유효한 이메일과 비밀번호 입력시
Main Page 이동

Main Page

[회원 가입](#)

[로그인](#)

Spring Security Test

```
<dependency>  
    <groupId>org.springframework.security</groupId>  
    <artifactId>spring-security-test</artifactId>  
    <scope>test</scope>  
</dependency>
```

```

package com.shop.controller;

@SpringBootTest
@AutoConfigureMockMvc
public class MemberControllerTest {
    @Autowired
    private MemberService memberService;
    @Autowired
    private MockMvc mockMvc;

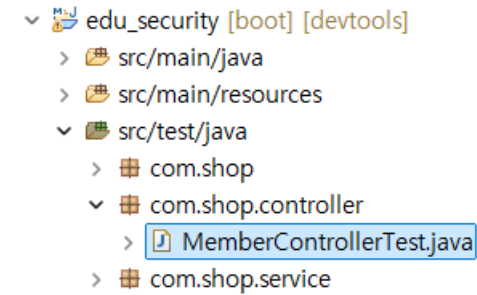
    public Member createMember(String email, String password) {
        MemberFormDto memberFormDto = new MemberFormDto();
        memberFormDto.setEmail(email);
        memberFormDto.setName("홍길동");
        memberFormDto.setAddress("서울 강남구");
        memberFormDto.setPassword(password);
        return memberService.saveMember(memberFormDto);
    }

    @Test
    @DisplayName("로그인 성공 테스트")
    public void loginSuccessTest() throws Exception {
        String email = "test5@email.com";
        String password = "12345678";
        this.createMember(email, password);

        mockMvc.perform(
            formLogin().userParameter("email").loginProcessingUrl("/member/login").user(email).password(password)
                .andExpect(SecurityMockMvcResultMatchers.authenticated());
        );
    }

    @Test
    @DisplayName("로그인 실패 테스트")
    public void loginFailTest() throws Exception {
        String email = "test6@email.com";
        String password = "12345678";
        this.createMember(email, password);
        mockMvc.perform(
            formLogin().userParameter("email").loginProcessingUrl("/member/login").user(email).password("12345")
                .andExpect(SecurityMockMvcResultMatchers.unauthenticated());
        );
    }
}

```



Thymeleaf 인증 처리

```
<dependency>  
  <groupId>org.thymeleaf.extras</groupId>  
  <artifactId>thymeleaf-extras-springsecurity6</artifactId>  
</dependency>
```

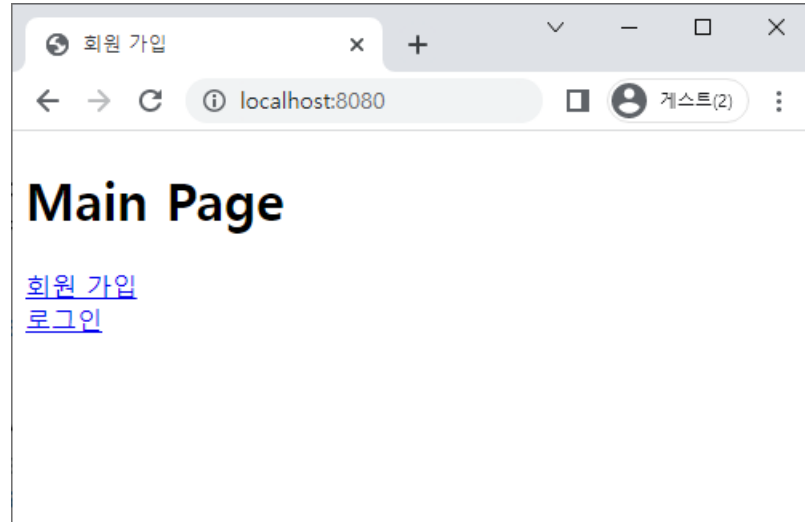
main.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<head>
    <meta charset="UTF-8">
    <title>main</title>
</head>
<body>
    <h1>Main Page</h1>

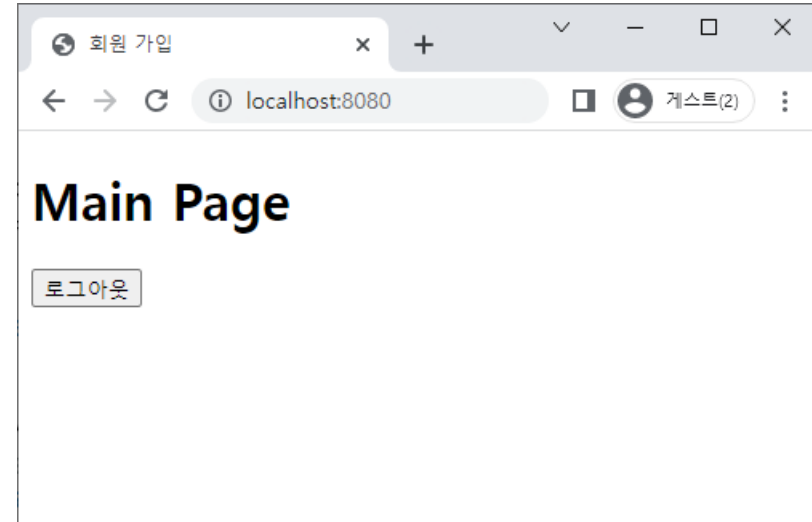
    <!-- 사용자가 인증되지 않은 경우에만 보이는 링크들 -->
    <div sec:authorize="isAnonymous()">
        <a href="/member/new">회원 가입</a><br>
        <a href="/member/login">로그인</a><br>
    </div>

    <!-- 사용자가 인증된 경우에만 보이는 로그아웃 폼 -->
    <div sec:authorize="isAuthenticated()">
        <form action="/member/logout" method="post">
            <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}" />
            <input type="submit" value="로그아웃" />
        </form>
    </div>
</body>
</html>
```

Thymeleaf 인증 처리



<로그인 전 화면>



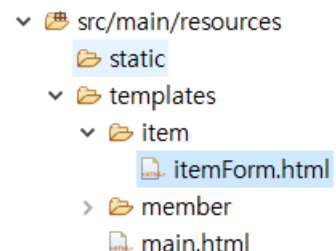
<로그인 후 화면>

권한 설정

Spring Security

templates/item/itemForm.html

<h1> 상품 등록 페이지 입니다. </h1>



```
package com.shop.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class ItemController {

    @GetMapping("/admin/item/new")
    public String itemForm() {
        return "/item/itemForm";
    }
}
```

```
package com.shop.config;
```

~ 생략 ~

```
@Configuration
```

```
@EnableWebSecurity
```

```
public class SecurityConfig {
```

```
    @Autowired
```

```
    private MemberService memberService;
```

```
    @Bean
```

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

```
        http
```

```
            .authorizeRequests()
```

```
            .requestMatchers("/", "/member/new", "/member/login").permitAll()
```

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

```
            .anyRequest().authenticated();
```

```
        http.formLogin()
```

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

```
            .defaultSuccessUrl("/")
```

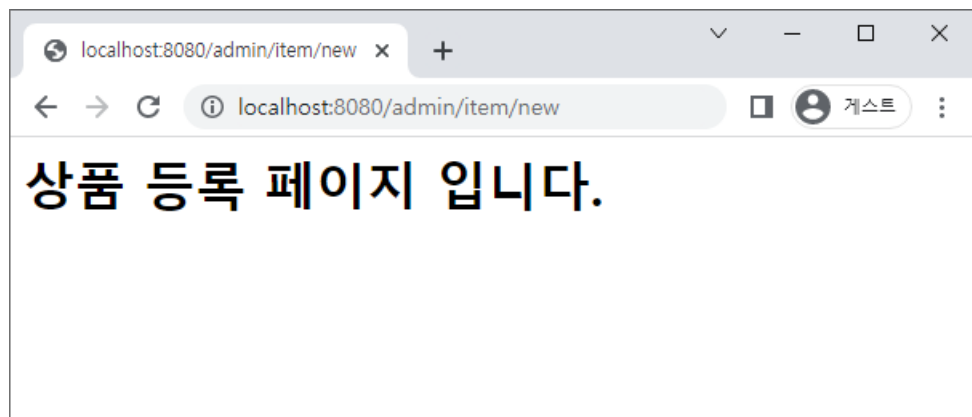
```
            .usernameParameter("email")
```

```
            .failureUrl("/member/login/error");
```

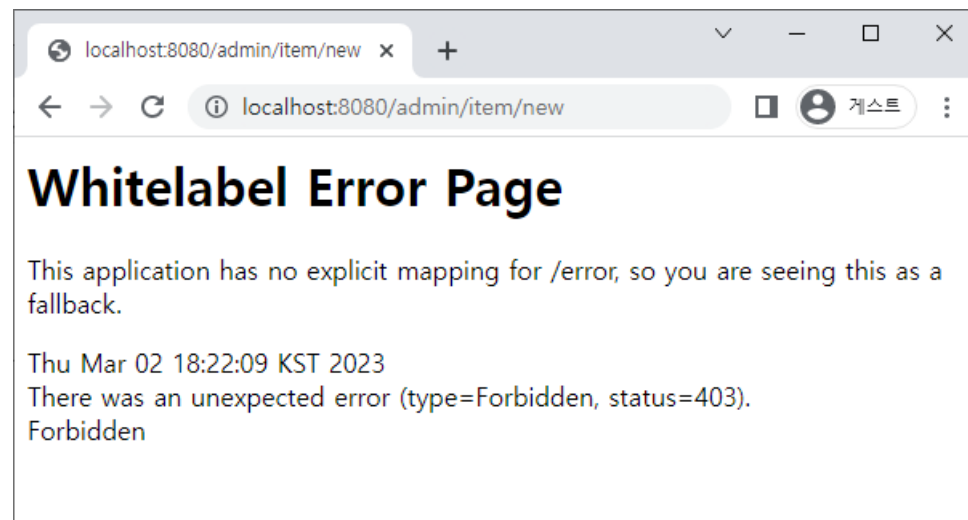
```
package com.shop.service;  
~ 생략 ~
```

```
public class MemberService implements UserDetailsService{  
    ~ 생략 ~  
    public Member saveMember(MemberFormDto memberFormDto){  
        Member member = modelMapper.map(memberFormDto, Member.class);  
  
        //비밀번호 인코딩  
        String password = passwordEncoder.encode(memberFormDto.getPassword());  
        member.setPassword(password);  
  
        //역할(Role) 설정  
        member.setRole(Role.ADMIN);  
  
        validateDuplicateMember(member);  
        return memberRepository.save(member);  
    }  
}
```

http://localhost:8080/admin/item/new



<ADMIN Role 회원 로그인 후 접속 화면>



<USER Role 회원 로그인 후 접속 화면>

```
package com.shop.controller;

import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.security.test.context.support.WithMockUser;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
~ 생략 ~
```

```
@SpringBootTest
@AutoConfigureMockMvc
public class ItemControllerTest {
```

```
    @Autowired
    private MockMvc mockMvc;
```

```
    @Test
    @DisplayName("상품 등록 페이지 권한 테스트")
    @WithMockUser(username = "admin", roles = "ADMIN")
    public void itemFormTest() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get("/admin/item/new"))
            .andDo(print())
            .andExpect(status().isOk());
    }
```

```
    @Test
    @DisplayName("상품 등록 페이지 일반 회원 접근 테스트")
    @WithMockUser(username = "user", roles = "USER")
    public void itemFormNotAdminTest() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get("/admin/item/new"))
            .andDo(print())
            .andExpect(status().isForbidden());
    }
```

```
}
```

