

학습 목표

- 1. 스프링 시큐리티를 이용해 회원 가입 및 로그인/로그아웃 기능을 구현하는 방법을 학습한다.
- 2. 스프링 시큐리티를 이용해 회원의 역할에 따라서 페이지별 접근 권한을 부여하는 방법을 알아본다.

4.1 스프링 시큐리티 소개

1. 애플리케이션을 만들기 위해서는 보통 인증/인가 등의 보안이 필요
 2. 웹에서 인증이란 해당 리소스에 대해서 작업을 수행할 수 있는 주체인지 확인하는 것
 3. 인가는 인증 과정 이후에 일어나며 리소스에 접근 시 인가된 유저인지 확인 (접근 권한 확인)
- 4장에서는 스프링 시큐리티를 이용하여 인증과 인가 구현
-

4.2 스프링 시큐리티 설정 추가

pom.xml

```
01 <dependency>
02     <groupId>org.springframework.boot</groupId>
03     <artifactId>spring-boot-starter-security</artifactId>
04 </dependency>
```

4.2 스프링 시큐리티 설정 추가

- 스프링 시큐리티 의존성 추가 시 모든 요청은 인증을 필요로 함
- 기존에 진행했던 예제 URL (<http://localhost/thymeleaf/ex07>) 에 접근 시 스프링 시큐리티에서 제공하는 로그인 페이지로 이동



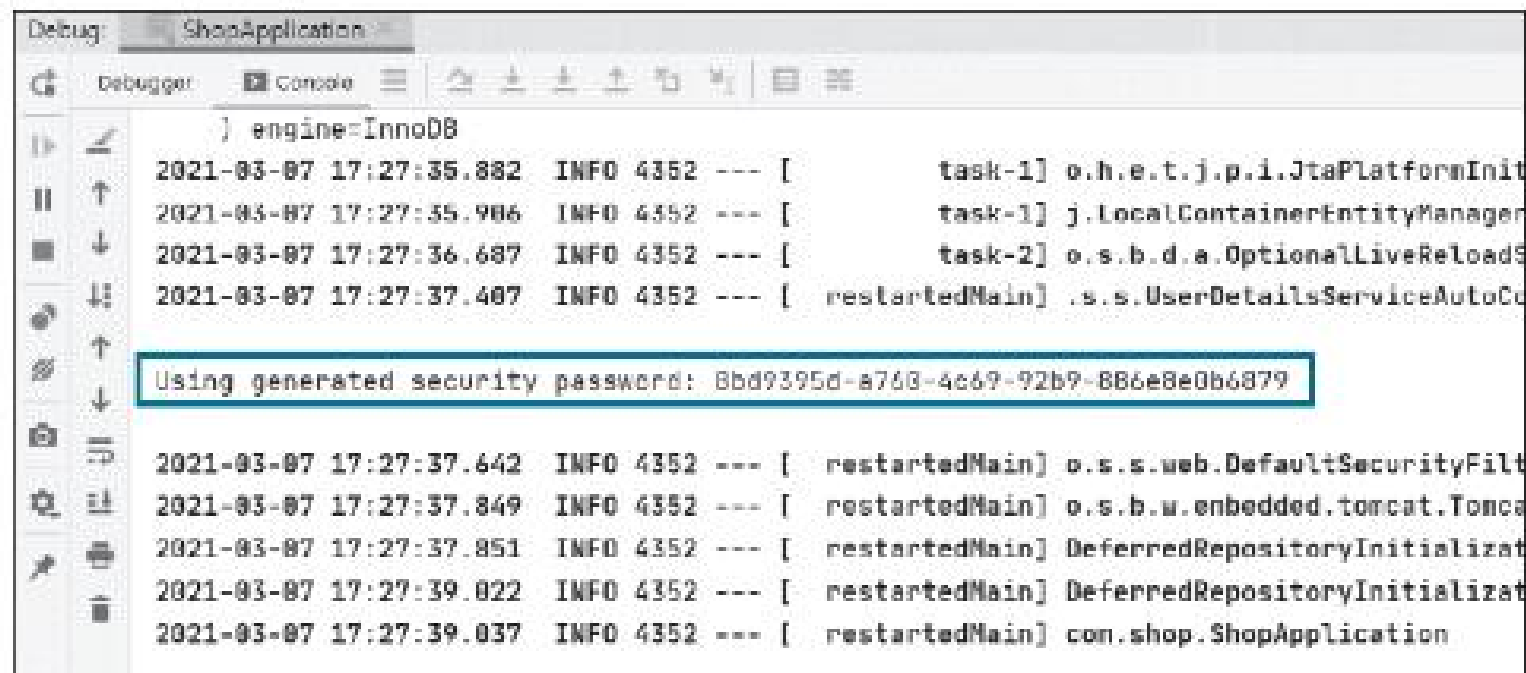
[함께 해봐요 4-1] 스프링 시큐리티 로그인하기



[그림 4-2] 스프링 시큐리티에서 제공하는 기본 로그인 페이지

4.2 스프링 시큐리티 설정 추가

- 기본적으로 제공하는 아이디는 user이고, 비밀번호는 애플리케이션을 실행할 때마다 콘솔창에 출력됨



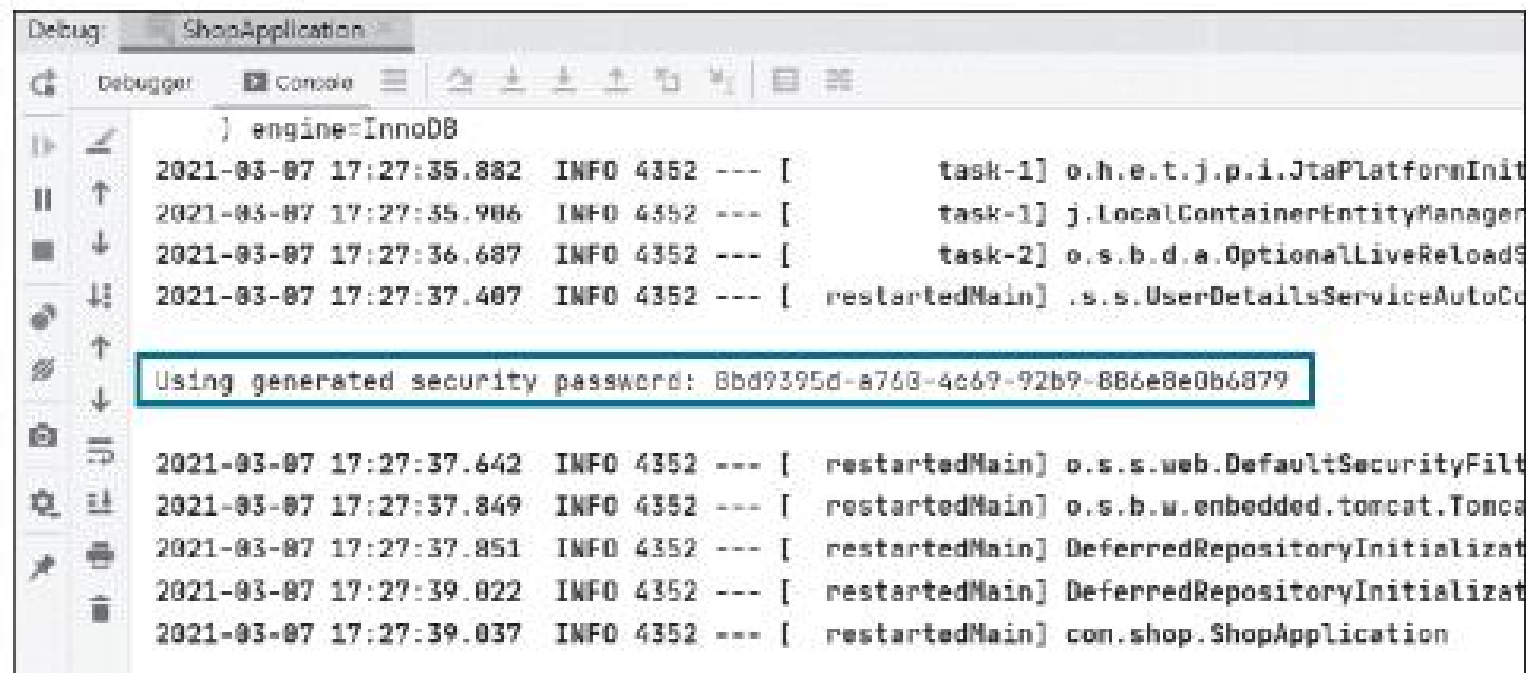
The screenshot shows the Spring IDE's 'Debug' console for a 'ShopApplication'. The 'Console' tab is active, displaying a series of log messages. A specific message, 'Using generated security password: 8bd9395d-a760-4c69-92b9-886e8e0b6879', is highlighted with a blue rectangular box. The log messages include timestamps, log levels (INFO), and thread names (task-1, restartedMain) along with class names and method calls.

```
Debug: ShopApplication
Debugger Console
] engine=InnoDB
2021-03-07 17:27:35.882 INFO 4352 --- [ task-1] o.h.e.t.j.p.i.JtaPlatformInit
2021-03-07 17:27:35.906 INFO 4352 --- [ task-1] j.LocalContainerEntityManager
2021-03-07 17:27:36.687 INFO 4352 --- [ task-2] o.s.b.d.a.OptionalLiveReloadS
2021-03-07 17:27:37.407 INFO 4352 --- [ restartedMain] .s.s.UserDetailsServiceAutoCo
Using generated security password: 8bd9395d-a760-4c69-92b9-886e8e0b6879
2021-03-07 17:27:37.642 INFO 4352 --- [ restartedMain] o.s.s.web.DefaultSecurityFilt
2021-03-07 17:27:37.849 INFO 4352 --- [ restartedMain] o.s.b.w.embedded.tomcat.Tomca
2021-03-07 17:27:37.851 INFO 4352 --- [ restartedMain] DeferredRepositoryInitializat
2021-03-07 17:27:39.022 INFO 4352 --- [ restartedMain] DeferredRepositoryInitializat
2021-03-07 17:27:39.037 INFO 4352 --- [ restartedMain] com.shop.ShopApplication
```

[그림 4-3] 스프링 시큐리티에서 제공하는 user 비밀번호

4.2 스프링 시큐리티 설정 추가

- 기본적으로 제공하는 아이디는 user이고, 비밀번호는 애플리케이션을 실행할 때마다 콘솔창에 출력됨



The screenshot shows the Spring IDE's 'Debug' console for a 'ShopApplication'. The 'Console' tab is active, displaying a series of log messages. A specific message, 'Using generated security password: 8bd9395d-a760-4c69-92b9-886e8e0b6879', is highlighted with a blue rectangular box. The log messages include timestamps, log levels (INFO), and thread names (task-1, task-2, restartedMain) along with the corresponding log messages.

```
Debug: ShopApplication
Debugger Console
] engine=InnoDB
2021-03-07 17:27:35.882 INFO 4352 --- [ task-1] o.h.e.t.j.p.i.JtaPlatformInit
2021-03-07 17:27:35.906 INFO 4352 --- [ task-1] j.LocalContainerEntityManager
2021-03-07 17:27:36.687 INFO 4352 --- [ task-2] o.s.b.d.a.OptionalLiveReloadS
2021-03-07 17:27:37.407 INFO 4352 --- [ restartedMain] .s.s.UserDetailsServiceAutoCo
Using generated security password: 8bd9395d-a760-4c69-92b9-886e8e0b6879
2021-03-07 17:27:37.642 INFO 4352 --- [ restartedMain] o.s.s.web.DefaultSecurityFilt
2021-03-07 17:27:37.849 INFO 4352 --- [ restartedMain] o.s.b.w.embedded.tomcat.Tonca
2021-03-07 17:27:37.851 INFO 4352 --- [ restartedMain] DeferredRepositoryInitializat
2021-03-07 17:27:39.022 INFO 4352 --- [ restartedMain] DeferredRepositoryInitializat
2021-03-07 17:27:39.037 INFO 4352 --- [ restartedMain] com.shop.ShopApplication
```

[그림 4-3] 스프링 시큐리티에서 제공하는 user 비밀번호

4.2 스프링 시큐리티 설정 추가



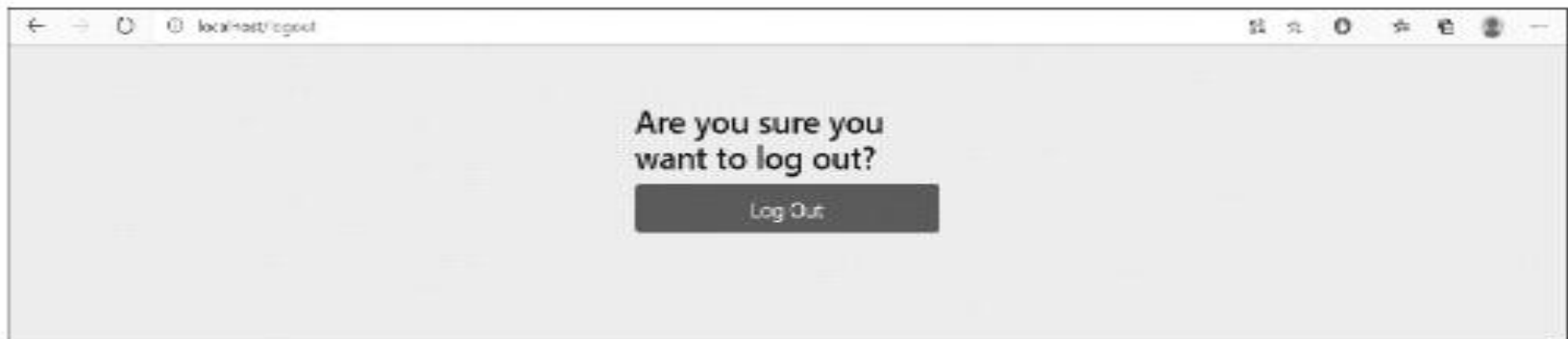
[그림 4-4] user 로그인



[그림 4-5] user 로그인 결과

4.2 스프링 시큐리티 설정 추가

- URL에 localhost/logout을 입력하면 스프링 시큐리티에서 기본으로 제공하는 로그아웃화면 제공



[그림 4-6] user 로그아웃

4.2 스프링 시큐리티 설정 추가

지금은 “user” 계정 밖에 없으며, 애플리케이션 실행할 때마다 비밀번호도 바뀜.
이 상태로는 애플리케이션을 운영할 수 없기 때문에 회원 가입 기능이 필요

- 인증이 필요 없는 경우: 상품 상세 페이지 조회
- 인증이 필요한 경우: 상품 주문
- 관리자 권한이 필요한 경우: 상품 등록

4.2 스프링 시큐리티 설정 추가



[함께 해봐요 4-2] SecurityConfig 클래스 작성하기

com.shop.config.SecurityConfig.java

```
01 package com.shop.config;
02
03 import org.springframework.context.annotation.Bean;
04 import org.springframework.context.annotation.Configuration;
05 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
06 import org.springframework.security.config.annotation.web.configuration.
    EnableWebSecurity;
07 import org.springframework.security.config.annotation.web.configuration.
    WebSecurityConfigurerAdapter;
08 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
09 import org.springframework.security.crypto.password.PasswordEncoder;
10
11 @Configuration
12 @EnableWebSecurity ----- ❶
13 public class SecurityConfig extends WebSecurityConfigurerAdapter { ----- ❷
14
15     @Override
16     protected void configure(HttpSecurity http) throws Exception { ----- ❸
17
18     }
19
20     @Bean
21     public PasswordEncoder passwordEncoder(){ ----- ❹
22         return new BCryptPasswordEncoder();
23     }
24 }
```

- SecurityConfig 클래스 작성 후 configure 메서드에 설정을 추가하지 않으면 더 이상 요청에 인증을 요구하지 않음
- URL에 따른 인증 및 인가 기능은 뒤의 예제에서 진행

4.3 회원 가입 기능 구현



[함께 해봐요 4-3] 회원 가입 기능 구현하기

com.shop.constant.Role.java

```
01 package com.shop.constant;  
02  
03 public enum Role {  
04     USER, ADMIN  
05 }
```

1

4.3 회원 가입 기능 구현

- 회원 가입 화면으로부터 넘어오는 가입 정보를 담은 DTO 생성

com.shop.dto.MemberFormDto.java

```
01 package com.shop.dto;
02
03 import lombok.Getter;
04 import lombok.Setter;
05
06 @Getter @Setter
07 public class MemberFormDto {
08
09     private String name;
10
11     private String email;
12
13     private String password;
14
15     private String address;
16
17 }
```

4.3 회원 가입 기능 구현

com.shop.entity.Member.java

```
01 package com.shop.entity;
02
03 import com.shop.constant.Role;
04 import com.shop.dto.MemberFormDto;
05 import lombok.Getter;
06 import lombok.Setter;
07 import lombok.ToString;
08 import org.springframework.security.crypto.password.PasswordEncoder;
09
10 import javax.persistence.*;
11
12 @Entity
13 @Table(name="member")
14 @Getter @Setter
15 @ToString
16 public class Member {
17
18     @Id
19     @Column(name="member_id")
20     @GeneratedValue(strategy = GenerationType.AUTO)
21     private Long id;
```

4.3 회원 가입 기능 구현

```
23     private String name;
24
25     @Column(unique = true) ①
26     private String email;
27
28     private String password;
29
30     private String address;
31
32     @Enumerated(EnumType.STRING) ②
33     private Role role;
34
35     public static Member createMember(MemberFormDto memberFormDto,
36                                     PasswordEncoder passwordEncoder){ ③
37         Member member = new Member();
38         member.setName(memberFormDto.getName());
39         member.setEmail(memberFormDto.getEmail());
40         member.setAddress(memberFormDto.getAddress());
41         String password = passwordEncoder.encode(memberFormDto.getPassword()); ④
42         member.setPassword(password);
43         member.setRole(Role.USER);
44         return member;
45     }
46 }
```

4.3 회원 가입 기능 구현

com.shop.repository.MemberRepository.java

```
01 package com.shop.repository;
02
03 import com.shop.entity.Member;
04 import org.springframework.data.jpa.repository.JpaRepository;
05
06 public interface MemberRepository extends JpaRepository<Member, Long> {
07
08     Member findByEmail(String email);
09
10 }
```

1

4.3 회원 가입 기능 구현

```
com.shop.service.MemberService.java

01 package com.shop.service;
02
03 import com.shop.entity.Member;
04 import com.shop.repository.MemberRepository;
05 import lombok.RequiredArgsConstructor;
06 import org.springframework.stereotype.Service;
07 import org.springframework.transaction.annotation.Transactional;
08
09 @Service
10 @Transactional ..... ❶
11 @RequiredArgsConstructor ..... ❷
12 public class MemberService {
13
14     private final MemberRepository memberRepository; ..... ❸
15
16     public Member saveMember(Member member){
17         validateDuplicateMember(member);
18         return memberRepository.save(member);
19     }
20
21     private void validateDuplicateMember(Member member){ ..... ❹
22         Member findMember = memberRepository.findByEmail(member.getEmail());
23         if(findMember != null){
24             throw new IllegalStateException("이미 가입된 회원입니다.");
25         }
26     }
27
28 }
```


4.3 회원 가입 기능 구현



[함께 해봐요 4-4] 회원 가입 기능 테스트하기

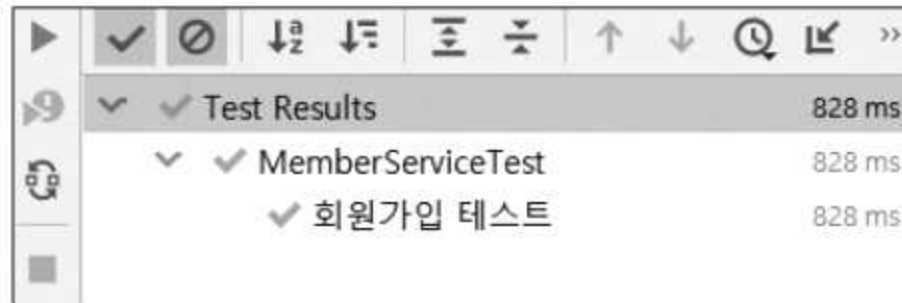
com.shop.service.MemberServiceTest.java

```
01 package com.shop.service;
02
03 import com.shop.dto.MemberFormDto;
04 import com.shop.entity.Member;
05 import org.junit.jupiter.api.DisplayName;
06 import org.junit.jupiter.api.Test;
07 import org.springframework.beans.factory.annotation.Autowired;
08 import org.springframework.boot.test.context.SpringBootTest;
09 import org.springframework.security.crypto.password.PasswordEncoder;
10 import org.springframework.test.context.TestPropertySource;
11 import org.springframework.transaction.annotation.Transactional;
12
13 import static org.junit.jupiter.api.Assertions.assertEquals;
14
15 @SpringBootTest
16 @Transactional
17 @TestPropertySource(locations="classpath:application-test.properties")
18 class MemberServiceTest {
19
20     @Autowired
21     MemberService memberService;
22
23     @Autowired
24     PasswordEncoder passwordEncoder;
```

4.3 회원 가입 기능 구현

```
26 public Member createMember(){ ..... ②
27     MemberFormDto memberFormDto = new MemberFormDto();
28     memberFormDto.setEmail("test@email.com");
29     memberFormDto.setName("홍길동");
30     memberFormDto.setAddress("서울시 마포구 합정동");
31     memberFormDto.setPassword("1234");
32     return Member.createMember(memberFormDto, passwordEncoder);
33 }
34
35 @Test
36 @DisplayName("회원가입 테스트")
37 public void saveMemberTest(){ ..... ③
38     Member member = createMember();
39     Member savedMember = memberService.saveMember(member);
40
41     assertEquals(member.getEmail(), savedMember.getEmail());
42     assertEquals(member.getName(), savedMember.getName());
43     assertEquals(member.getAddress(), savedMember.getAddress());
44     assertEquals(member.getPassword(), savedMember.getPassword());
45     assertEquals(member.getRole(), savedMember.getRole());
46 }
47
48 }
```

4.3 회원 가입 기능 구현



The screenshot shows a test results window with a toolbar at the top containing icons for pass, fail, and various sorting options. The results are listed in a tree view:

Test Item	Duration
Test Results	828 ms
MemberServiceTest	828 ms
회원가입 테스트	828 ms

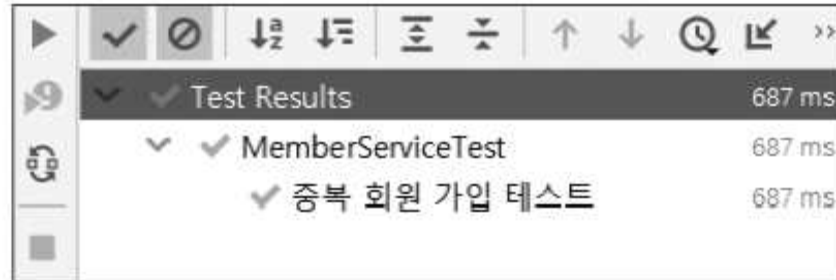
[그림 4-7] 회원 가입 테스트 코드 실행 결과

4.3 회원 가입 기능 구현

com.shop.service.MemberServiceTest.java

```
01 package com.shop.service;
02
03 .....기존 임포트 생략.....
04
05 import static org.junit.jupiter.api.Assertions.assertThrows;
06
07 @SpringBootTest
08 @Transactional
09 @TestPropertySource(locations="classpath:application-test.properties")
10 class MemberServiceTest {
11
12     .....코드 생략.....
13
14     @Test
15     @DisplayName("중복 회원 가입 테스트")
16     public void saveDuplicateMemberTest(){
17         Member member1 = createMember();
18         Member member2 = createMember();
19         memberService.saveMember(member1);
20
21         Throwable e = assertThrows(IllegalStateException.class, () -> { ❶
22             memberService.saveMember(member2);});
23
24         assertEquals("이미 가입된 회원입니다.", e.getMessage()); ❷
25     }
26
27 }
```

4.3 회원 가입 기능 구현



The screenshot shows a test runner window with a toolbar at the top containing icons for play, check, cancel, and various sorting options. The main area displays a tree of test results, all of which are marked with green checkmarks, indicating successful execution. The tests and their durations are as follows:

Test Name	Duration
Test Results	687 ms
MemberServiceTest	687 ms
중복 회원 가입 테스트	687 ms

[그림 4-8] 중복 회원 가입 테스트 코드 실행 결과

4.3 회원 가입 기능 구현



[함께 해봐요 4-5] 회원 가입 페이지 작성하기

com.shop.controller.MemberController.java

```
01 package com.shop.controller;
02
03 import com.shop.dto.MemberFormDto;
04 import com.shop.service.MemberService;
05 import lombok.RequiredArgsConstructor;
06 import org.springframework.stereotype.Controller;
07 import org.springframework.ui.Model;
08 import org.springframework.web.bind.annotation.GetMapping;
09 import org.springframework.web.bind.annotation.RequestMapping;
10
11 @RequestMapping("/members")
12 @Controller
13 @RequiredArgsConstructor
14 public class MemberController {
15
16     private final MemberService memberService;
17
18     @GetMapping(value = "/new")
19     public String memberForm(Model model){ ----- ❶
20         model.addAttribute("memberFormDto", new MemberFormDto());
21         return "member/memberForm";
22     }
23
24 }
```

4.3 회원 가입 기능 구현

resources/templates/member/memberForm.html

```
01 <!DOCTYPE html>
02 <html xmlns:th="http://www.thymeleaf.org"
03       xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
04       layout:decorate="~{layouts/layout1}">
05
06 <!-- 사용자 CSS 추가 -->
07 <th:block layout:fragment="css">
08     <style>
09         .fieldError {
10             color: #bd2130;
11         }
12     </style>
13 </th:block>
14
15 <!-- 사용자 스크립트 추가 -->
16 <th:block layout:fragment="script">
17
18     <script th:inline="javascript">
19         $(document).ready(function(){
20             var errorMessage = [[${errorMessage}]];
21             if(errorMessage != null){
22                 alert(errorMessage);
23             }
24         }
25     </script>
26 </th:block>
```

1

4.3 회원 가입 기능 구현

```
24     });
25     </script>
26
27 </th:block>
28
29 <div layout:fragment="content">
30
31     <form action="/members/new" role="form" method="post"
32           th:object="${memberFormDto}">
33         <div class="form-group">
34             <label th:for="name">이름</label>
35             <input type="text" th:field="*{name}" class="form-control"
36                   placeholder="이름을 입력해주세요">
37             <p th:if="${#fields.hasErrors('name')}"
38                 th:errors="*{name}" class="fieldError">Incorrect data</p>
39         </div>
40         <div class="form-group">
41             <label th:for="email">이메일주소</label>
42             <input type="email" th:field="*{email}" class="form-control"
43                   placeholder="이메일을 입력해주세요">
44             <p th:if="${#fields.hasErrors('email')}"
45                 th:errors="*{email}" class="fieldError">Incorrect data</p>
46         </div>
47     </form>
48 </div>
```


4.3 회원 가입 기능 구현

```
42     <div class="form-group">
43         <label th:for="password">비밀번호</label>
44         <input type="password" th:field="*{password}"
45             class="form-control" placeholder="비밀번호 입력">
46     <p th:if="${#fields.hasErrors('password')}}" th:errors="*{password}"
47         class="fieldError">Incorrect data</p>
48 </div>
49 <div class="form-group">
50     <label th:for="address">주소</label>
51     <input type="text" th:field="*{address}" class="form-control"
52         placeholder="주소를 입력해주세요">
53     <p th:if="${#fields.hasErrors('address')}}" th:errors="*{address}"
54         class="fieldError">Incorrect data</p>
55 </div>
56 <div style="text-align: center;">
57     <button type="submit" class="btn btn-primary" style="">Submit
58     </button>
59 </div>
60 <input type="hidden" th:name="${_csrf.parameterName}"
    th:value="${_csrf.token}"> ..... ②
61 </form>
62 </div>
63 </html>
```

4.3 회원 가입 기능 구현



여기서 잠깐

CSRF

CSRF(Cross Site Request Forgery)란 사이트간 위조 요청으로 사용자가 자신의 의지와 상관없이 해커가 의도한 대로 수정, 등록, 삭제 등의 행위를 웹사이트 요청하게 하는 공격을 말합니다.

4.3 회원 가입 기능 구현



[함께 해봐요 4-6] 회원 가입 컨트롤러 소스코드 작성하기

com.shop.controller.MemberController.java

```
01 package com.shop.controller;
02
03 .....기존 임포트 생략.....
04
05 import com.shop.entity.Member;
06 import org.springframework.security.crypto.password.PasswordEncoder;
07 import org.springframework.web.bind.annotation.PostMapping;
08
09 @RequestMapping("/members")
10 @Controller
11 @RequiredArgsConstructor
12 public class MemberController {
13
14     private final MemberService memberService;
15     private final PasswordEncoder passwordEncoder;
16 }
```

4.3 회원 가입 기능 구현

```
17     @GetMapping(value = "/new")
18     public String memberForm(Model model){
19         model.addAttribute("memberFormDto", new MemberFormDto());
20         return "member/memberForm";
21     }
22
23     @PostMapping(value = "/new")
24     public String memberForm(MemberFormDto memberFormDto){
25
26         Member member = Member.createMember(memberFormDto, passwordEncoder);
27         memberService.saveMember(member);
28
29         return "redirect:/";
30     }
31
32 }
```

4.3 회원 가입 기능 구현

com.shop.controller.MainController.java

```
01 package com.shop.controller;
02
03 import org.springframework.stereotype.Controller;
04 import org.springframework.web.bind.annotation.GetMapping;
05
06 @Controller
07 public class MainController {
08
09     @GetMapping(value="/")
10     public String main(){
11         return "main";
12     }
13
14 }
```

4.3 회원 가입 기능 구현

resources/templates/main.html

```
01 <!DOCTYPE html>
02 <html xmlns:th="http://www.thymeleaf.org"
03       xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
04       layout:decorate="~{layouts/layout1}">
05
06 <div layout:fragment="content">
07
08     <h1>메인페이지입니다.</h1>
09
10 </div>
```

4.3 회원 가입 기능 구현

localhost/members/new

Shop | 상품 목록 | 상품 판매 | 판매 관리 | 관리자 페이지 | 로그인 | 로그아웃

Search

이름
이름을 입력해주세요

이메일주소
이메일을 입력해주세요

비밀번호
비밀번호를 입력

주소
주소를 입력해주세요

Submit

© 2020 Shopping Mall Example WebSite

[그림 4-10] 회원 가입 페이지

4.3 회원 가입 기능 구현

- 회원 가입 페이지에서 서버로 넘어오는 값을 검증하기 위해서 pom.xml에 “spring-boot-starter-validation” 추가



[함께 해봐요 4-7] 회원 가입 처리하기

pom.xml

```
01 <dependency>
02     <groupId>org.springframework.boot</groupId>
03     <artifactId>spring-boot-starter-validation</artifactId>
04 </dependency>
```


4.3 회원 가입 기능 구현

- 유효한 값인지 판단하는 소스가 여러 군데 흩어지면 관리하기가 힘들.
- 자바 빈 밸리데이션을 이용하면 객체의 값을 효율적 검증 가능

[표 3-1] javax.validation 어노테이션 예시

어노테이션	설명
@NotEmpty	NULL 체크 및 문자열의 경우 길이 0인지 검사
@NotBlank	NULL 체크 및 문자열의 경우 길이 0 및 빈 문자열(" ") 검사
@Length(min=, max=)	최소, 최대 길이 검사
@Email	이메일 형식인지 검사
@Max(숫자)	지정한 값보다 작은지 검사
@Min(숫자)	지정한 값보다 큰지 검사
@Null	값이 NULL인지 검사
@NotNull	값이 NULL이 아닌지 검사

4.3 회원 가입 기능 구현

com.shop.dto.MemberFormDto.java

```
01 package com.shop.dto;
02
03 import lombok.Getter;
04 import lombok.Setter;
05 import org.hibernate.validator.constraints.Length;
06
07 import javax.validation.constraints.Email;
08 import javax.validation.constraints.NotBlank;
09 import javax.validation.constraints.NotEmpty;
10
11 @Getter @Setter
12 public class MemberFormDto {
13
14     @NotBlank(message = "이름은 필수 입력 값입니다.")
15     private String name;
16
17     @NotEmpty(message = "이메일은 필수 입력 값입니다.")
18     @Email(message = "이메일 형식으로 입력해주세요.")
19     private String email;
20
21     @NotEmpty(message = "비밀번호는 필수 입력 값입니다.")
22     @Length(min=8, max=16, message = "비밀번호는 8자 이상, 16자 이하로 입력해주세요")
23     private String password;
24
25     @NotEmpty(message = "주소는 필수 입력 값입니다.")
26     private String address;
27
28 }
```

4.3 회원 가입 기능 구현

com.shop.controller.MemberController.java

```
01 package com.shop.controller;
02
03 .....기존 импорт 생략.....
04
05 import org.springframework.validation.BindingResult;
06 import javax.validation.Valid;
07
08 @RequestMapping("/members")
09 @Controller
10 @RequiredArgsConstructor
11 public class MemberController {
12
13     .....코드 생략.....
14
15     @PostMapping(value = "/new")
16     public String newMember(@Valid MemberFormDto memberFormDto,
                             BindingResult bindingResult, Model model){ — ❶
```

4.3 회원 가입 기능 구현

```
17
18     if(bindingResult.hasErrors()){ ..... ❷
19         return "member/memberForm";
20     }
21
22     try {
23         Member member = Member.createMember(memberFormDto, passwordEncoder);
24         memberService.saveMember(member);
25     } catch (IllegalStateException e){
26         model.addAttribute("errorMessage", e.getMessage()); ..... ❸
27         return "member/memberForm";
28     }
29
30     return "redirect:/";
31 }
32
33 }
```

4.3 회원 가입 기능 구현

Shop | 상품 목록 | 상품 소개 | 장바구니 | 구매하기 | 로그인 | 회원가입

Search

이름

이름을 입력해주세요.

이름은 필수 입력 값입니다.

이메일주소

이메일을 입력해주세요.

이메일은 필수 입력 값입니다.

비밀번호

비밀번호 입력

비밀번호는 필수 입력 값입니다.
비밀번호는 8자 이상, 16자 이하로 입력해주세요.

주소

주소를 입력해주세요.

주소는 필수 입력 값입니다.

Submit

©2021 Shopping Mall Example Website

[그림 4-11] 회원 가입 시 데이터 검증 결과

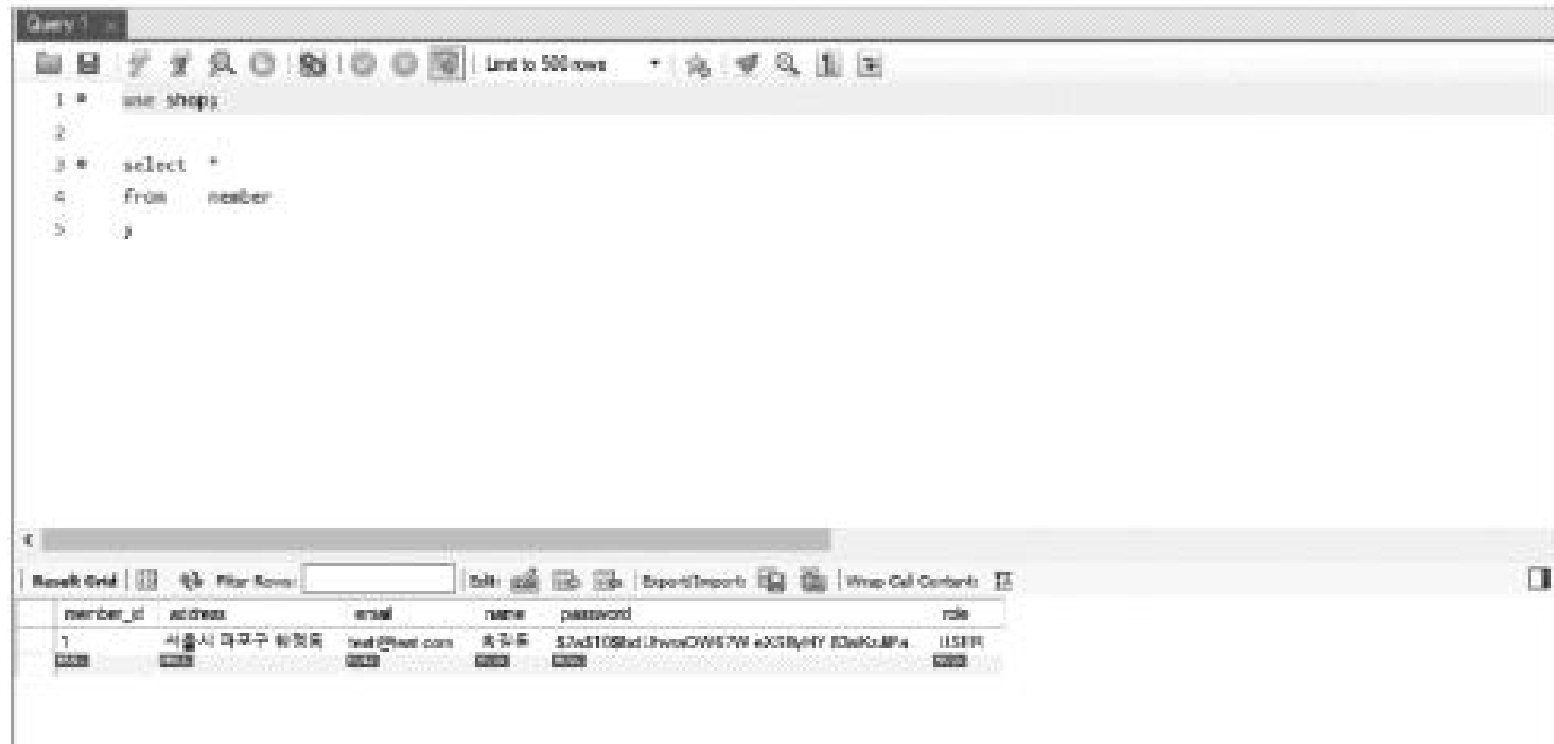
4.3 회원 가입 기능 구현

- 회원 가입이 정상적으로 이루어졌다면 메인 페이지로 이동



[그림 4-12] 회원 가입 성공

4.3 회원 가입 기능 구현



[그림 4-13] 회원 가입 정보 mysql 데이터 베이스 조회

4.4 로그인/로그아웃 구현

- 스프링 시큐리티에서 UserDetailsService를 구현하고 있는 클래스를 통해 로그인 기능을 구현
- UserDetailsService 인터페이스는 데이터베이스에서 회원 정보를 가져오는 역할을 담당
- loadUserByUsername() 메소드가 존재하며, 회원 정보를 조회하여 사용자의 정보와 권한을 갖는 UserDetails 인터페이스를 반환
- UserDetails : 스프링 시큐리티에서 회원의 정보를 담기 위해서 사용하는 인터페이스. 인터페이스를 직접 구현하거나 스프링 시큐리티에서 제공하는 User 클래스 (UserDetails 인터페이스를 구현하고 있는 클래스)를 사용

4.4 로그인/로그아웃 구현



[함께 해봐요 4-8] 로그인/로그아웃 기능 구현하기

com.shop.service.MemberService.java

```
01 package com.shop.service;
02
03 .....기존 임포트 생략.....
04
05 import org.springframework.security.core.userdetails.User;
06 import org.springframework.security.core.userdetails.UserDetails;
07 import org.springframework.security.core.userdetails.UserDetailsService;
08 import org.springframework.security.core.userdetails.UsernameNotFoundException;
09
10 @Service
11 @Transactional
12 @RequiredArgsConstructor
13 public class MemberService implements UserDetailsService { .....❶
14
15     .....코드 생략.....
```

4.4 로그인/로그아웃 구현

```
17  @Override
18  public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException { ..... ❷
19      Member member = memberRepository.findByEmail(email);
20
21      if(member == null){
22          throw new UsernameNotFoundException(email);
23      }
24
25      return User.builder() ..... ❸
26          .username(member.getEmail())
27          .password(member.getPassword())
28          .roles(member.getRole().toString())
29          .build();
30  }
31 }
```

4.4 로그인/로그아웃 구현

com.shop.config.SecurityConfig.java

```
01 package com.shop.config;
02
03 .....기존 임포트 생략.....
04
05 import com.shop.service.MemberService;
06 import org.springframework.beans.factory.annotation.Autowired;
07 import org.springframework.security.config.annotation.authentication.
    builders.AuthenticationManagerBuilder;
08 import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
09
10 @Configuration
11 @EnableWebSecurity
12 public class SecurityConfig extends WebSecurityConfigurerAdapter {
13
14     @Autowired
15     MemberService memberService;
16
17     @Override
18     protected void configure(HttpSecurity http) throws Exception {
19         http.formLogin()
20             .loginPage("/members/login") ----- ❶
21             .defaultSuccessUrl("/") ----- ❷
22             .usernameParameter("email") ----- ❸
23             .failureUrl("/members/login/error") ----- ❹
24             .and()
25             .logout()
26             .logoutRequestMatcher(new AntPathRequestMatcher
                ("/members/logout")) ----- ❺
27             .logoutSuccessUrl("/") ----- ❻
28         ;
29     }
```

4.4 로그인/로그아웃 구현

```
31  @Bean
32  public PasswordEncoder passwordEncoder(){
33      return new BCryptPasswordEncoder();
34  }
35
36  @Override
37  protected void configure(AuthenticationManagerBuilder auth)
throws Exception { ..... 7
38      auth.userDetailsService(memberService)
        .passwordEncoder(passwordEncoder()); ..... 8
39  }
40 }
```

4.4 로그인/로그아웃 구현

resources/templates/member/memberLoginForm.html

```
01 <!DOCTYPE html>
02 <html xmlns:th="http://www.thymeleaf.org"
03       xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
04       layout:decorate="~{layouts/layout1}">
05
06 <!-- 사용자 CSS 추가 -->
07 <th:block layout:fragment="css">
08     <style>
09         .error {
10             color: #bd2130;
11         }
12     </style>
13 </th:block>
14
15 <div layout:fragment="content">
16
```

4.4 로그인/로그아웃 구현

```
17 <form role="form" method="post" action="/members/login">
18   <div class="form-group">
19     <label th:for="email">이메일주소</label>
20     <input type="email" name="email" class="form-control"
21           placeholder="이메일을 입력해주세요">
22   </div>
23   <div class="form-group">
24     <label th:for="password">비밀번호</label>
25     <input type="password" name="password" id="password"
26           class="form-control" placeholder="비밀번호 입력">
27   </div>
28   <p th:if="${loginErrorMsg}" class="error" th:text="${loginErrorMsg}"></p>
29   <button class="btn btn-primary">로그인</button>
30   <button type="button" class="btn btn-primary"
31         onClick="location.href='/members/new'">회원가입</button>
32   <input type="hidden" th:name="${_csrf.parameterName}"
33         th:value="${_csrf.token}">
34 </form>
</div>
</html>
```

4.4 로그인/로그아웃 구현

com.shop.controller.MemberController.java

```
01 package com.shop.controller;
02
03 .....기존 임포트 생략.....
04
05 @RequestMapping("/members")
06 @Controller
07 @RequiredArgsConstructor
08 public class MemberController {
09
10     .....코드 생략.....
11
12     @GetMapping(value = "/login")
13     public String loginMember(){
14         return "/member/memberLoginForm";
15     }
16
17     @GetMapping(value = "/login/error")
18     public String loginError(Model model){
19         model.addAttribute("loginErrorMsg", "아이디 또는 비밀번호를 확인해주세요");
20         return "/member/memberLoginForm";
21     }
22 }
```

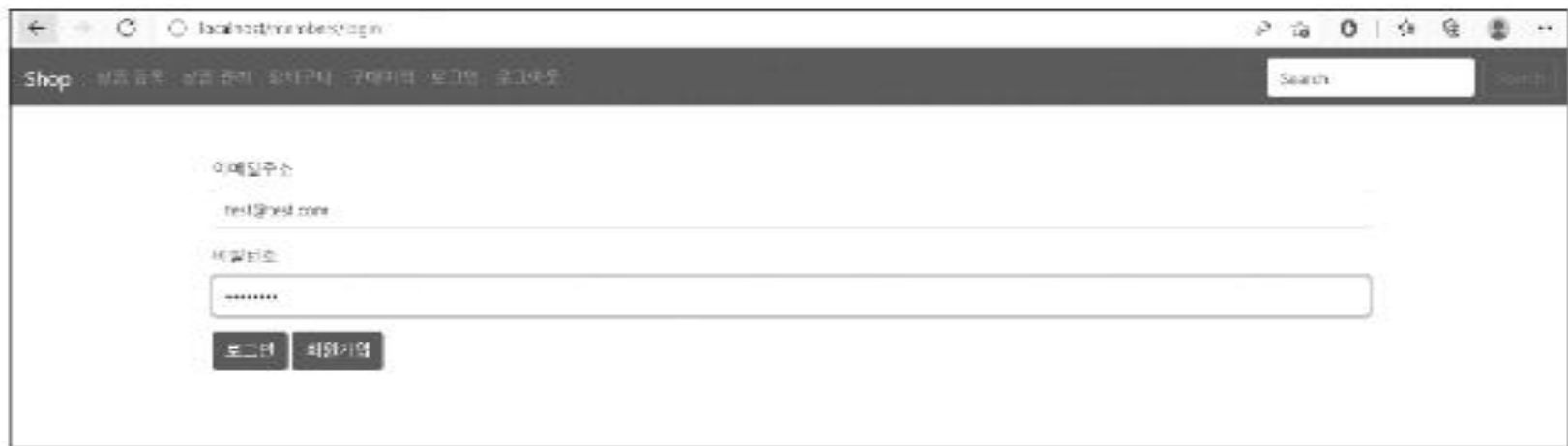
4.4 로그인/로그아웃 구현

com.shop.controller.MemberController.java

```
01 package com.shop.controller;
02
03 .....기존 임포트 생략.....
04
05 @RequestMapping("/members")
06 @Controller
07 @RequiredArgsConstructor
08 public class MemberController {
09
10     .....코드 생략.....
11
12     @GetMapping(value = "/login")
13     public String loginMember(){
14         return "/member/memberLoginForm";
15     }
16
17     @GetMapping(value = "/login/error")
18     public String loginError(Model model){
19         model.addAttribute("loginErrorMsg", "아이디 또는 비밀번호를 확인해주세요");
20         return "/member/memberLoginForm";
21     }
22 }
```


4.4 로그인/로그아웃 구현

- 로그인 페이지(localhost/member/login) url로 이동



The screenshot shows a web browser window with the address bar displaying 'localhost/member/login'. The page features a dark header bar with navigation links: 'Shop', '상품 등록', '상품 관리', '회원관리', '구매관리', '로그인', and '로그아웃'. A search bar is located on the right side of the header. The main content area is white and contains a login form. The form has two input fields: the first is labeled '이메일주소' (Email Address) and contains the text 'test@test.com'; the second is labeled '비밀번호' (Password) and is masked with asterisks. Below the input fields are two buttons: '로그인' (Login) and '회원가입' (Sign Up).

[그림 4-14] 로그인 화면

4.4 로그인/로그아웃 구현



The screenshot shows a web browser window with the address bar displaying "localhost:members/login.html". The page has a dark header with the text "Shop" and navigation links: "상품 등록", "상품 관리", "판매구입", "구매자의", "로그인", and "로그아웃". A search bar is on the right. The main content area contains a login form with the following elements:

- A label "이메일주소" above an input field with the placeholder text "이메일을 입력해주세요".
- A label "비밀번호" above an input field with the placeholder text "비밀번호 입력".
- A text label "아이디 또는 비밀번호를 확인해주세요".
- Two buttons at the bottom: "로그인" and "회원가입".

[그림 4-15] 로그인 실패



[그림 4-16] 로그인 성공

4.4 로그인/로그아웃 구현



[함께 해봐요 4-9] 로그인 테스트하기

pom.xml

```
01 <dependency>
02   <groupId>org.springframework.security</groupId>
03   <artifactId>spring-security-test</artifactId>
04   <scope>test</scope>
05   <version>${spring-security.version}</version>
06 </dependency>
```

4.4 로그인/로그아웃 구현

com.shop.controller.MemberControllerTest.java

```
01 package com.shop.controller;
02
03 import com.shop.dto.MemberFormDto;
04 import com.shop.entity.Member;
05 import com.shop.service.MemberService;
06 import org.junit.jupiter.api.DisplayName;
07 import org.junit.jupiter.api.Test;
08 import org.springframework.beans.factory.annotation.Autowired;
09 import org.springframework.boot.test.autoconfigure.web.servlet
    .AutoConfigureMockMvc;
10 import org.springframework.boot.test.context.SpringBootTest;
11 import org.springframework.security.crypto.password.PasswordEncoder;
12 import org.springframework.security.test.web.servlet.response
    .SecurityMockMvcResultMatchers;
13 import org.springframework.test.context.TestPropertySource;
14 import org.springframework.test.web.servlet.MockMvc;
15 import org.springframework.transaction.annotation.Transactional;
16
17 import static org.springframework.security.test.web.servlet.request
    .SecurityMockMvcRequestBuilders.formLogin;
18
19 @SpringBootTest
20 @AutoConfigureMockMvc
21 @Transactional
22 @TestPropertySource(locations="classpath:application-test.properties")
```

4.4 로그인/로그아웃 구현

```
23 class MemberControllerTest {
24
25     @Autowired
26     private MemberService memberService;
27
28     @Autowired
29     private MockMvc mockMvc; ②
30
31     @Autowired
32     PasswordEncoder passwordEncoder;
33
34     public Member createMember(String email, String password){ ③
35         MemberFormDto memberFormDto = new MemberFormDto();
36         memberFormDto.setEmail(email);
37         memberFormDto.setName("홍길동");
38         memberFormDto.setAddress("서울시 마포구 합정동");
39         memberFormDto.setPassword(password);
40         Member member = Member.createMember(memberFormDto, passwordEncoder);
41         return memberService.saveMember(member);
42     }
```

4.4 로그인/로그아웃 구현

```
44     @Test
45     @DisplayName("로그인 성공 테스트")
46     public void loginSuccessTest() throws Exception{
47         String email = "test@email.com";
48         String password = "1234";
49         this.createMember(email, password);
50         mockMvc.perform(formLogin().userParameter("email")
51                     .loginProcessingUrl("/members/login") ..... 4
52                     .user(email).password(password))
53                     .andExpect(SecurityMockMvcResultMatchers.authenticated()); 5
54     }
55 }
```

4.4 로그인/로그아웃 구현



[그림 4-17] 로그인 성공 케이스 테스트 코드 실행 결과

4.4 로그인/로그아웃 구현

- 로그인 실패 테스트 코드 작성

```
01 package com.shop.controller;
02
03 .....기존 임포트 생략.....
04
05 @SpringBootTest
06 @AutoConfigureMockMvc
07 @Transactional
08 @TestPropertySource(locations="classpath:application-test.properties")
09 class MemberControllerTest {
10
11     .....코드 생략.....
12
13     @Test
14     @DisplayName("로그인 실패 테스트")
15     public void loginFailTest() throws Exception{
16         String email = "test@email.com";
17         String password = "1234";
18         this.createMember(email, password);
19         mockMvc.perform(formLogin().userParameter("email")
20                     .loginProcessingUrl("/members/login")
21                     .user(email).password("12345"))
22                 .andExpect(MockMvcResultMatchers.unauthenticated()); ❶
23     }
```


4.4 로그인/로그아웃 구현



The screenshot shows the Test Results window of an IDE. The window has a toolbar with icons for running, passing, failing, and other test-related actions. The test results are listed in a table-like structure with expandable nodes. The '로그인 실패 테스트' (Login Failure Test) is highlighted, indicating it failed. The execution time for this test is 956 ms.

▶	✓	⊘	↕	↕	⏏	⏏	↑	↓	🔍	📄	»
▶	✓	Test Results									956 ms
▶	✓	MemberControllerTest									956 ms
	✓	로그인 실패 테스트									956 ms

[그림 4-18] 로그인 실패 케이스 테스트 코드 실행 결과

4.4 로그인/로그아웃 구현

- 현재 상태로는 로그인을 해도 메뉴바에는 로그인이라는 메뉴가 나타나고, 로그인 상태라면 로그아웃이라는 메뉴가 화면에 나타나야함
- 로그인 또는 권한에 따라서 화면 출력 결과를 만드는걸 도와주는 라이브러리로 “thymeleaf-extras-springsecurity5” 사용



[함께 해봐요 4-10] 로그인/로그아웃 화면 연동하기

pom.xml

```
01 <dependency>
02     <groupId>org.thymeleaf.extras</groupId>
03     <artifactId>thymeleaf-extras-springsecurity5</artifactId>
04 </dependency>
```

4.4 로그인/로그아웃 구현

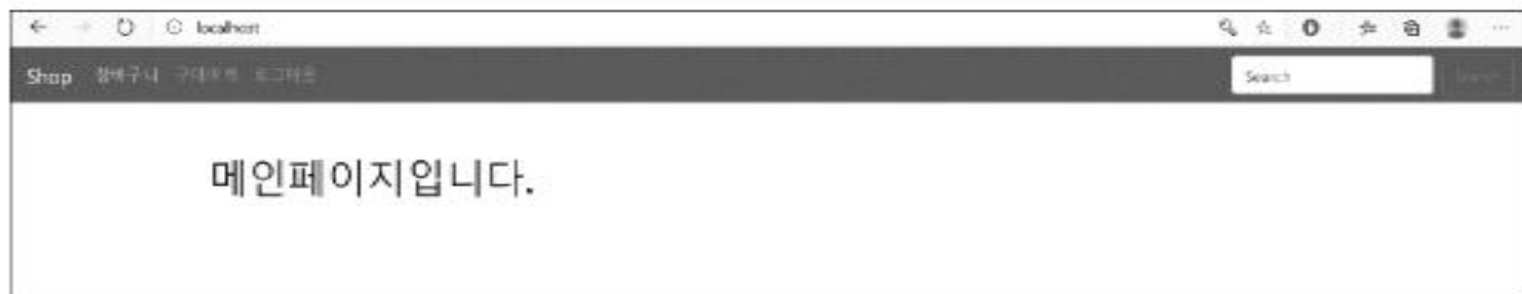
resources/templates/fragments/header.html

```
01 <!DOCTYPE html>
02 <html xmlns:th="http://www.thymeleaf.org"
03       xmlns:sec="http://www.thymeleaf.org/extras/spring-security"> ①
04
05 <div th:fragment="header">
06     <nav class="navbar navbar-expand-sm bg-primary navbar-dark">
07         <button class="navbar-toggler" type="button" data-toggle="collapse"
08             data-target="#navbarTogglerDemo03"
09             aria-controls="navbarTogglerDemo03"
10             aria-expanded="false" aria-label="Toggle navigation">
11             <span class="navbar-toggler-icon"></span>
12         </button>
13         <a class="navbar-brand" href="/">Shop</a>
14
15         <div class="collapse navbar-collapse" id="navbarTogglerDemo03">
16             <ul class="navbar-nav mr-auto mt-2 mt-lg-0">
17                 <li class="nav-item">
18                     sec:authorize="hasAnyAuthority('ROLE_ADMIN')"> ②
19                     <a class="nav-link" href="/admin/item/new">상품 등록</a>
20                 </li>
21                 <li class="nav-item">
22                     sec:authorize="hasAnyAuthority('ROLE_ADMIN')"> ③
23                     <a class="nav-link" href="/admin/items">상품 관리</a>
24                 </li>
25             </ul>
26         </div>
27     </nav>
28 </div>
```

4.4 로그인/로그아웃 구현

```
23         <li class="nav-item" sec:authorize="isAuthenticated()"> ❹
24             <a class="nav-link" href="/cart">장바구니</a>
25         </li>
26         <li class="nav-item" sec:authorize="isAuthenticated()"> ❺
27             <a class="nav-link" href="/orders">구매이력</a>
28         </li>
29         <li class="nav-item" sec:authorize="isAnonymous()"> ❻
30             <a class="nav-link" href="/members/login">로그인</a>
31         </li>
32         <li class="nav-item" sec:authorize="isAuthenticated()"> ❼
33             <a class="nav-link" href="/members/logout">로그아웃</a>
34         </li>
35     </ul>
36     <form class="form-inline my-2 my-lg-0"
43         th:action="@{/}" method="get">
37         <input name="searchQuery" class="form-control mr-sm-2"
38             type="search" placeholder="Search" aria-label="Search">
39         <button class="btn btn-outline-success my-2 my-sm-0"
40             type="submit">Search</button>
41     </form>
42 </div>
43 </nav>
44 </div>
45 </html>
```

4.4 로그인/로그아웃 구현



[그림 4-19] 회원 역할에 따른 네비게이션 노출



[그림 4-20] 회원 역할에 따른 네비게이션 노출

4.5 페이지 권한 설정



[함께 해봐요 4-11] 페이지 권한 설정하기

resources/templates/item/itemForm.html

```
01 <!DOCTYPE html>
02 <html xmlns:th="http://www.thymeleaf.org"
03       xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
04       layout:decorate="~~{layouts/layout1}">
05
06 <div layout:fragment="content">
07
08     <h1>상품등록 페이지입니다.</h1>
09
10 </div>
11
12 </html>
```

4.5 페이지 권한 설정

com.shop.controller.ItemController.java

```
01 package com.shop.controller;
02
03 import org.springframework.stereotype.Controller;
04 import org.springframework.web.bind.annotation.GetMapping;
05
06 @Controller
07 public class ItemController {
08
09     @GetMapping(value = "/admin/item/new")
10     public String itemForm(){
11         return "/item/itemForm";
12     }
13
14 }
```

4.5 페이지 권한 설정

com.shop.config.CustomAuthenticationEntryPoint.java

```
01 package com.shop.config;
02
03 import org.springframework.security.core.AuthenticationException;
04 import org.springframework.security.web.AuthenticationEntryPoint;
05
06 import javax.servlet.ServletException;
07 import javax.servlet.http.HttpServletRequest;
08 import javax.servlet.http.HttpServletResponse;
09 import java.io.IOException;
10
11 public class CustomAuthenticationEntryPoint implements AuthenticationEntryPoint {
12
13     @Override public void commence
14     (HttpServletRequest request, HttpServletResponse response,
15     AuthenticationException authException) throws IOException, ServletException {
16         response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Unauthorized");
17     }
18 }
```

인증되지 않은 사용
자가 리소스를 요청
하는 경우
"Unauthorized" 에러
발생 하도록
Authentication
EntryPoint 인터페이
스 구현

4.5 페이지 권한 설정

com.shop.config.SecurityConfig.java

```
01 package com.shop.config;
02
03 .....기존 임포트 생략.....
04
05 import org.springframework.security.config.annotation.web.builders.WebSecurity;
06
07 @Configuration
08 @EnableWebSecurity
09 public class SecurityConfig extends WebSecurityConfigurerAdapter {
10
11     .....코드 생략.....
12
13     @Override
14     protected void configure(HttpSecurity http) throws Exception {
15         http.formLogin()
16             .loginPage("/members/login")
17             .defaultSuccessUrl("/")
18             .usernameParameter("email")
19             .failureUrl("/members/login/error")
20             .and()
21             .logout()
22             .logoutRequestMatcher(new AntPathRequestMatcher
23                                     ("/members/logout"))
24             .logoutSuccessUrl("/");
25     }
```

4.5 페이지 권한 설정

```
26     http.authorizeRequests() ----- ❶
27         .mvcMatchers("/", "/members/**",
28             "/item/**", "/images/**").permitAll() ----- ❷
29         .mvcMatchers("/admin/**").hasRole("ADMIN") ----- ❸
30         .anyRequest().authenticated() ----- ❹
31     ;
32     http.exceptionHandling()
33         .authenticationEntryPoint
34         (new CustomAuthenticationEntryPoint()) ----- ❺
35     ;
36 }
37
38 @Override
39 public void configure(WebSecurity web) throws Exception {
40     web.ignoring().antMatchers("/css/**", "/js/**", "/img/**"); ----- ❻
41 }
```

4.5 페이지 권한 설정

- 현재 회원 가입 시 권한을 USER로 생성하므로, 로그인 후 'http://localhost/admin/item/new'라는 상품 등록 ADMIN 페이지에 접근하려고 하면 403 Forbidden에러 발생



[그림 4-21] USER Role 회원 ADMIN 페이지 접근 실패

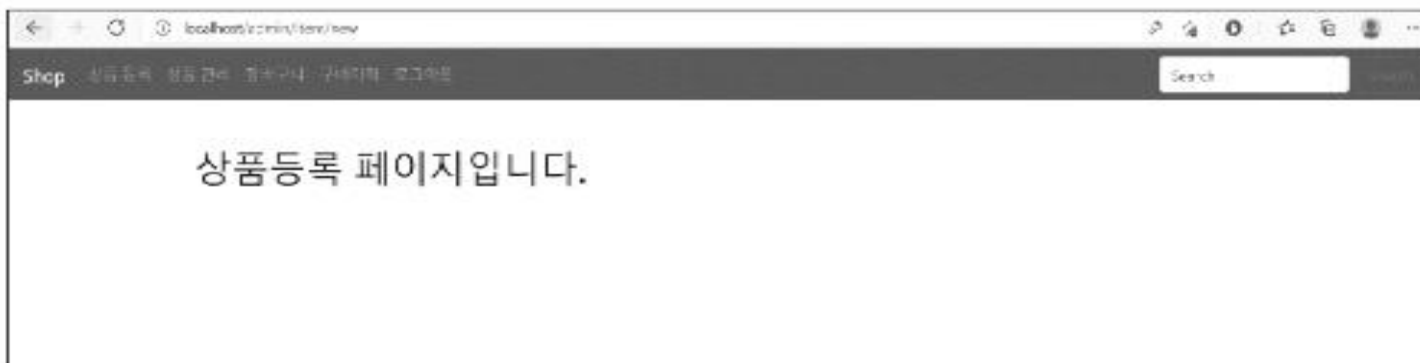
4.5 페이지 권한 설정

com.shop.entity.Member.java

```
01 package com.shop.entity;
02
03 @Entity
04 @Table(name="member")
05 @Getter @Setter
06 @ToString
07 public class Member {
08
09     .....코드 생략.....
10
11     public static Member createMember(MemberFormDto memberFormDto,
12                                     PasswordEncoder passwordEncoder){
13         Member member = new Member();
14         member.setName(memberFormDto.getName());
15         member.setEmail(memberFormDto.getEmail());
16         member.setAddress(memberFormDto.getAddress());
17         String password = passwordEncoder.encode(memberFormDto.getPassword());
18         member.setPassword(password);
19         member.setRole(Role.ADMIN);
20     }
21
22 }
```

4.5 페이지 권한 설정

- 다시 회원 가입 진행 및 로그인 후 상품 등록 페이지 접근 시 정상적으로 화면이 나오는 것을 볼 수 있음



[그림 4-22] ADMIN Role 회원 ADMIN 페이지 접근 성공

4.5 페이지 권한 설정



[함께 해봐요 4-12] 유저 접근 권한 테스트하기

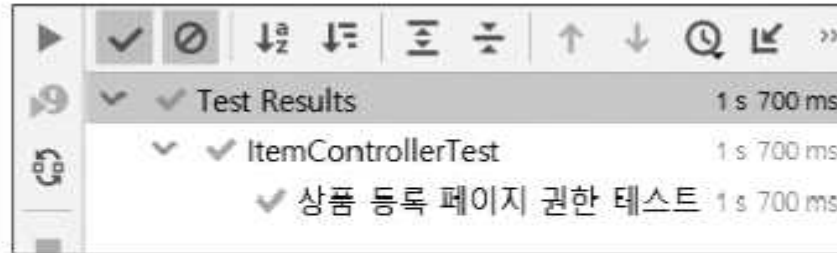
com.shop.controller.ItemControllerTest.java

```
01 package com.shop.controller;
02
03 import org.junit.jupiter.api.DisplayName;
04 import org.junit.jupiter.api.Test;
05 import org.springframework.beans.factory.annotation.Autowired;
06 import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
07 import org.springframework.boot.test.context.SpringBootTest;
08 import org.springframework.security.test.context.support.WithMockUser;
09 import org.springframework.test.context.TestPropertySource;
10 import org.springframework.test.web.servlet.MockMvc;
11 import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
12
13 import static org.springframework.test.web.servlet.result
    .MockMvcResultHandlers.print;
14 import static org.springframework.test.web.servlet.result
    .MockMvcResultMatchers.status;
15
16 @SpringBootTest
17 @AutoConfigureMockMvc
18 @TestPropertySource(locations="classpath:application-test.properties")
```

4.5 페이지 권한 설정

```
19 class ItemControllerTest {
20
21     @Autowired
22     MockMvc mockMvc;
23
24     @Test
25     @DisplayName("상품 등록 페이지 권한 테스트")
26     @WithMockUser(username = "admin", roles = "ADMIN") ❶
27     public void itemFormTest() throws Exception{
28         mockMvc.perform(MockMvcRequestBuilders.get("/admin/item/new")) ❷
29             .andDo(print()) ❸
30             .andExpect(status().isOk()); ❹
31     }
32
33 }
```

4.5 페이지 권한 설정



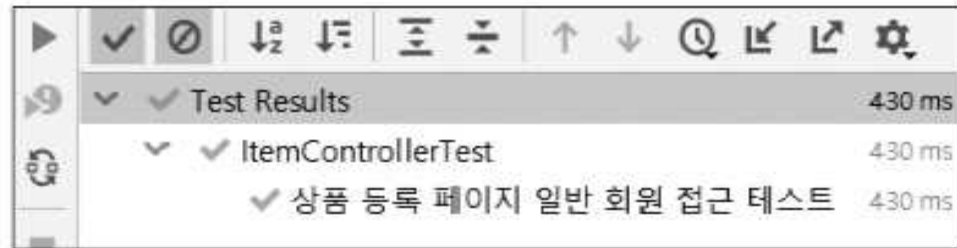
[그림 4-23] ADMIN Role 회원 ADMIN 페이지 접근 성공

4.5 페이지 권한 설정

com.shop.controller.ItemControllerTest.java

```
01 package com.shop.controller;
02
03 .....기존 임포트 생략.....
04
05 @SpringBootTest
06 @AutoConfigureMockMvc
07 @TestPropertySource(locations="classpath:application-test.properties")
08 class ItemControllerTest {
09
10     .....코드 생략.....
11
12     @Test
13     @DisplayName("상품 등록 페이지 일반 회원 접근 테스트")
14     @WithMockUser(username = "user", roles = "USER") ..... ❶
15     public void itemFormNotAdminTest() throws Exception{
16         mockMvc.perform(MockMvcRequestBuilders.get("/admin/item/new"))
17             .andDo(print())
18             .andExpect(status().isForbidden()); ..... ❷
19     }
20
21 }
```

4.5 페이지 권한 설정



▶	✓	⊘	↓	↓	≡	÷	↑	↓	🔍	↶	↷	⚙️
▼	✓	Test Results										430 ms
▼	✓	ItemControllerTest										430 ms
	✓	상품 등록 페이지 일반 회원 접근 테스트										430 ms

[그림 4-24] 상품 등록 페이지 USER Role 접근 테스트 결과

Thank you for your attention

© 2021. 변구훈 & 로드북 all rights reserved.

이 콘텐츠의 저작권은 조휘용과 로드북에 있습니다.
재배포가 가능하지만 저작권자 표시 및 콘텐츠 시작 부분에 나오는 표지를 반드시 실어야 합니다.
수정하여 재배포할 시에는 수정한 부분을 반드시 명시해야 합니다.