1. 웹 기반의 프로젝트에서는 사용자 인증(로그인,로그아웃)을 처리하는 부분은 반드시 필요한 부분이다. 스프링의 경우 이러한 작업을 위해서 Spring Security(스프링 시큐리티)를 적용할 수 있다. 이 장에서는 스프링 시큐리티를 적용해서 다음과 같은 내용들을 학습한다.

1-1. 사용자의 권한에 따른 URI 접근 제어

1-2. 데이터베이스와 연동하는 로그인 처리

1-3. 쿠키를 이용한 자동 로그인(remember-me)

1-4. 패스워드 암호화

2. 시큐리티를 적용하기 위해서는 아주 간단한 용어 정리가 필요하다. 인증과 인가라는 개념인데, 시큐리티를 적용하는 내내 필요한 용어이므로 간단히 알아본다.

2-1. **인증(Authentication)이라는 단어는 '증명하다'는 의미**이다. 이때 증명의 대상은 어떤 경우에는 표시나 자격일 수도 있고, 사람의 신분일 수도 있다. 예를 들어, 만일 우리가 암호가 걸려있는 시스템을 사용한다면, 인증 절차를 거쳐야만 한다.

이러한 의미로 인증이라는 것을 정리해 보면 'Pass(지나가다)'라는 개념과 관련이 있다고 생각할 수 있다. 가장 쉬운 예로 집에 들어가기 전 입력하는 암호 또는 열쇠로 문을 여는 행위가 바로 인증이라고 볼 수 있다. 실제로 인증은 보통 자물쇠와 같은 이미지로 표현된다.

2-2. **인가(Authorization)는 '권한부여'나 '허가'와 같은 의미로 사용**된다. 예를 들어, 아이들이 학교가 끝나고 집에 들어오는 행위는 현관문의 인증 절차를 통하지만, 집 안에 있는 간식을 먹기 위해서는 엄마의 '허락'이라는 것이 필요하다. 이 경우 엄마의 허락을 통해서 아이는 자격을 획득하게 되는 것이다.

보안에서 인가는 어떤 대상이 특정 목적을 실현하도록 허용(Access)하는 것을 의미한다. 보안에서는 어떤 대상이 있다면 인가된 사용자만이 보안된 대상을 사용할 수 있다는 의미로 해석한다.

웹에서 인증이란 해당 URL의 보안절차를 거친 사용자들만이 접근할 수 있다는 의미가 되고, 인가란 URL에 접근한 사용자가 특정한 자격이 있다는 것을 의미한다. 인증, 인가의 의미를 잘 이해하길 바란다.

3. application.properties 파일에 추가적인 설정 파일내용을 기입한다.

```
#Spring Web Log
logging.level.org.springframework.web=debug

#Spring Security Log show print
logging.level.org.springframework.security=debug
```

4. 스프링 시큐리티 기본 설정 추가하기

스프링 프로젝트에서 시큐리티가 추가될 것이므로, 시큐리티에 대한 설정이 필요하다. net.daum.security 패키지를 만들고 SecurityConfig라는 클래스를 만든다. 생성된 이 클래스는 스프링에 빈으로 인식되지 않는다. 그러므로 스프링 웹 시큐리티로 인식되게 @EnableWebSecurity 애노테이션 추가한다. 그리고 스프링 웹 시큐리티 설정을 담당하는 WebSecurityConfigurerAdapter 클래스를 상속받는다. 이 클래스의 여러 메서드 중에서 configure() 메서드를 오버라이딩 한다.

```
package net.daum.security;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigu
rerAdapter;

import lombok.extern.java.Log;

@Log
@EnableWebSecurity //스프링 웹 시큐리티로 인식되게 @EnableWebSecurity 애노테이션
추가
public class SecurityConfig extends WebSecurityConfigurerAdapter {
        //스프링 웹 시큐리티 설정을 담당하는 WebSecurityConfigurerAdapter 클래스를
상속받는다.

        @Override
        protected void configure(HttpSecurity http) throws Exception {//configure()
메서드를 오버라이딩을 함.
                //HttpSecurity는 웹과 관련된 다양한 보안설정을 걸어 줄 수 있다.
```

```
                log.info("security config .....");//간단한 로그 메시지를 출력한다.

        }
}
```

4-1. 스프링 시큐리티가 정상적으로 사용 가능하다는 것을 확인했다면, 웹상에 시큐리티를 적용하기 위한 컨트롤러와 화면들을 생성해 두어야 한다. 이를 위해서 net.daum.controller 패키지를 작성하고 SampleController.java 스프링 컨트롤러 클래스를 작성한다.

```
package net.daum.controller;

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

import lombok.extern.java.Log;

@Controller
@Log
public class SampleController {

        @GetMapping("/")
        public String index() {

                log.info("index");
                return "index";
        }//index()

        @RequestMapping("/guest")
        public void forGuest() {

                log.info("guest");
        }//guest

        @RequestMapping("/manager")
        public void forManager() {

                log.info("manager");
```

```
	}//manager

	@RequestMapping("/admin")
	public void forAdmin() {

		log.info("admin");
	}//admin
}
```

5. 회원(MemberVO)과 회원 권한(MemberRole) 엔티티빈 클래스 설계

net.daum.vo 패키지를 생성하고, MemberVO와 MemberRole 엔티티빈 클래스를 만든다.

```
package net.daum.vo;

import java.sql.Timestamp;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;

import org.hibernate.annotations.CreationTimestamp;

import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Setter //setter()메서드 자동제공
@Getter //getter()메서드 자동제공
@ToString //toString() 메서드 자동제공
@Entity
@Table(name="tbl_members7")
@EqualsAndHashCode(of="mem_id")
/* equals(), hashCode(),canEqual() 메서드 자동제공 *
```

```
 */
public class MemberVO {//회원관리 엔티티빈 클래스

	@Id //유일키로 사용될 기본키 컬럼 즉 primary key
	private String mem_id;//회원아이디

	private String mem_pwd;//비번
	private String mem_name;//회원이름

	@CreationTimestamp //@CreationTimestamp 는 하이버네이트의 특별한 기능으로 등록시점 날짜값을 기록,mybatis로 실행할 때는 구동 안됨.
	private Timestamp mem_date;//가입날짜

	@OneToMany(cascade = CascadeType.ALL, fetch=FetchType.EAGER) //일대다 연관관계,cascade = CascadeType.ALL은 JPA에서 영속성 전이중에서
	//모든 변경에 대한 전이로서 모든 엔티티빈 상태 변화에 대해서 같이 처리하는 옵션, fetch=FetchType.EAGER는 tbl_members7과 tbl_member_roles7 두 테이블을 조회해야 하기
	//때문에 트랜잭션을 처리해 주거나, 즉시 로딩을 이용해서 조인하는 방법으로 처리해야 한다. 권한 정보는 회원정보와 마찬가지로 필요한 경우가 많기 때문에 fetch 모드를 즉시 로딩으로
	//설정한다.
	@JoinColumn(name = "member") //이미 존재하는 tbl_member_roles7 테이블에 member컬럼 추가 , foreign key 추가 설정
	private List<MemberRole> roles;
}

package net.daum.vo;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;

import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;
```

```
@Getter
@Setter
@Entity
@SequenceGenerator(//@SequenceGenerator는 시퀀스 생성기를 설정하는 애노테이션
                name="member7_no_seq_gename", //시퀀스 제너레이터 이름
                sequenceName="member7_no_seq", //시퀀스 이름
                initialValue=1, //시작값
                allocationSize=1 //메모리를 통해 할당할 범위 사이즈=>기본값은 50이며,
1로 설정하는 경우 매번 insert시마다 DB의 시퀀스를 호출해서 db시퀀스 번호값을 가져와서
                //1증가한 값이 할당된다. 1씩 증가. 증가값
                )
@Table(name = "tbl_member_roles7")
@EqualsAndHashCode(of = "fno")
@ToString
public class MemberRole {//회원이 가지는 권한

          @Id
          @GeneratedValue(
                                strategy=GenerationType.SEQUENCE, //사용할 전략을
시퀀스로  선택
                                generator="member7_no_seq_gename"  //식별자  생
성기를 설정해놓은 member7_no_seq_gename 시퀀스 제너레이터 이름 으로 설정
                                )
          private int fno;

          private String roleName;//권한 이름
}
```

6. Repository 생성

net.daum.dao 패키지를 생성하고, MemberRepository를 작성한다.

```
package net.daum.dao;

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

```
import org.springframework.data.jpa.repository.Query;

import net.daum.vo.MemberVO;

public interface MemberRepository extends JpaRepository<MemberVO, String> {

	@Query("select m from MemberVO m where m.mem_id=?1 and m.mem_name=?2")
	public MemberVO pwdFind(String id,String name); //아이디와 회원이름을 기준으로 디비로 부터 비번을 검색

	@Modifying //@Query 애노테이션은 select문만 가능하지만 @Modifying을 이용해서 DML(insert,update,delete)문 작업 처리가 가능하다.
	@Query("update MemberVO m set m.mem_pwd=?1 where m.mem_id=?2") //?1은 첫번째로 전달되는 피라미터 값 ,?2은 두번째로 전달된  피라미터 값
	//JPQL(JPA에서 사용하는 Query Language => Java Persistence Query Language의 약어)이다.
	//JPQL은 테이블 대신 엔티빈 클래스를 이용하고,테이블 컬럼대신 엔티티빈 클래스의 변수 즉 속성을 이용한다.
	public void updatePwd(String pwd,String id); //아이디를 기준으로 암호화 된 임시 비번을 수정

}
```

7. 특정 권한을 가진 사람만이 특정 URI에 접근할 수 있다.

net.daum.security 패키지의 SecurityConfig 클래스에 진하게 한 부분만 새롭게 추가한다. 변경된 설정이 제대로 적용되는지를 확인하기 위해서는 프로젝트를 실행하고 '/guest','/manager' 경로로 접근해 본다. '/manager'로 접근하는 경우 403 접근금지 에러가 나는 것을 확인 할수 있다. 인가 받은 권한이 없기 때문에 해당 경로에 접근할 수 없도록 처리가 된 것이다.

```
.. 중략...
@Log
@EnableWebSecurity //스프링 웹 시큐리티로 인식되게 @EnableWebSecurity 애노테이션 추가
public class SecurityConfig extends WebSecurityConfigurerAdapter {
	//스프링 웹 시큐리티 설정을 담당하는 WebSecurityConfigurerAdapter 클래스를 상속받는다.

	@Override
```

```
        protected void configure(HttpSecurity http) throws Exception {//configure()
메서드를 오버라이딩을 함.
                //HttpSecurity는 웹과 관련된 다양한 보안설정을 걸어 줄 수 있다.

                log.info("security config .....");//간단한 로그 메시지를 출력한다.


http.authorizeRequests().antMatchers("/guest/**").permitAll();//authorizeRequests()는
시큐리티 처리에서 HttpServletRequest에 해당한다.
        /* antMatchers()에서는 특정한 경로를 지정한다. permitAll()은 모든 사용자가 접
근할 수 있다는 것을 의미한다.
         */

http.authorizeRequests().antMatchers("/manager/**").hasRole("MANAGER");
//hasRole()은 특정권한을 가진 사람만이 접근할 수 있다는 것을 의미한다.
        }
}
```

7-1. 인가받은 권한이 없는 관계로 접근이 막혔다면 '/login' 이라는 매핑주소를 호출해서 스프링 시큐리티 내부에서 제공하는 기본 로그인 페이지로 이동하게 만들 수 있다. 그렇게 하기 위해서 마지막에 http.formLogin() 이라는 경로를 작성해 준다. 이를 이용하면 별도의 로그인 페이지를 작성하지 않아도 스프링 시큐리티에서 제공하는 기본 로그인 페이지가 띄워진다.

```
       @Override
        protected void configure(HttpSecurity http) throws Exception {//configure()
메서드를 오버라이딩을 함.
                //HttpSecurity는 웹과 관련된 다양한 보안설정을 걸어 줄 수 있다.

                log.info("security config .....");//간단한 로그 메시지를 출력한다.


http.authorizeRequests().antMatchers("/guest/**").permitAll();//authorizeRequests()는
시큐리티 처리에서 HttpServletRequest에 해당한다.
        /* antMatchers()에서는 특정한 경로를 지정한다. permitAll()은 모든 사용자가 접
근할 수 있다는 것을 의미한다.
         */

http.authorizeRequests().antMatchers("/manager/**").hasRole("MANAGER");
//hasRole()은 특정권한을 가진 사람만이 접근할 수 있다는 것을 의미한다.
```

```
http.formLogin();

        }
```

7-2 . '/manager' 매핑주소로 접근했을 때 스프링 시큐리티에서 기본으로 제공하는 로그인 페이지에서 'manager' 아이디와 비번 '1111'로 로그인해서 정상적인 manager페이지로 이동하기 위해서 다음의 코드를 net.daum.security패키지의 SecurityConfig 클래스에 추가한다.

```
   @Autowired//화면에 로그인 페이지가 띄워져도 어떤 아이디나 비번을 입력해도 로그인 실패가 된다. 이런 경우는 로그인 되게 만들기 위해서
            //AuthenticationManagerBuilder을 주입해서 인증에 대한 처리를 해야 한다.
                public void configureGlobal(AuthenticationManagerBuilder auth) throws
            Exception {//AuthenticationManagerBuilder는 인증에 대한 다양한 설정을 할 수 있다. 예를 들어 메모리상에 정보만을 이용한다든지,
            //jdbc등을 이용해서 인증 처리가 가능하다. 여기서는 메모리상의 인증 정보를 활용한다.

                log.info("build Auth global.....");


auth.inMemoryAuthentication().withUser("manager").password("{noop}1111").roles("MANAGER");
                //사용자 manager,비번 1111,권한 MANAGER 지정
                //Spring Security 4에서는 메모리 내 인증을 사용하여 암호를 인코딩 즉 암호화 하지않고 일반 텍스트

로 저장할 수 있었다.
                //Spring Security 5부터는 비번을 인코딩 즉 암호화 해서 저장한다. 그러므로  There is no PasswordEncoder mapped for the id "null"
                //오류를 내지 않기 위해서는 {noop}을 사용해서 비번을 인코딩 즉 암호화해서 처리해야 한다.
            }
```

8. 커스텀 로그인 페이지 만들기

사용자 로그인 페이지를 따로 만들기 위해서는 net.daum.security 패키지의 SecurityConfig.java 코드의 formLoign()이후에 loginPage()메서드를 이용해서 이동할 매핑주소를 지정해 주면 된다.

..중략
 @Override
	protected void configure(HttpSecurity http) throws Exception {//configure() 메서드를 오버라이딩을 함.
		//HttpSecurity는 웹과 관련된 다양한 보안설정을 걸어 줄 수 있다.

		log.info("security config .....");//간단한 로그 메시지를 출력한다.

http.authorizeRequests().antMatchers("/guest/**").permitAll();//authorizeRequests()는 시큐리티 처리에서 HttpServletRequest에 해당한다.
	/* antMatchers()에서는 특정한 경로를 지정한다. permitAll()은 모든 사용자가 접근할 수 있다는 것을 의미한다.
	 */

http.authorizeRequests().antMatchers("/manager/**").hasRole("MANAGER");//hasRole()은 특정권한을 가진 사람만이 접근할 수 있다는 것을 의미한다.
		http.formLogin().loginPage("/login");
		/*http.formLogin()은 form 태그기반의 로그인을 지원하겠다는 설정이다. 이를 이용하면 별도의 로그인 페이지를 제작하지 않아도 스프링 시큐리티에서 제공하는
		 * /login 매핑주소로 인식되는 기본 로그인 페이지가 띄워진다.
			loginPage("/member_login");을 사용하면 매핑주소가 member_login인 사용자 즉 커스텀 로그인 페이지를 만들 수 있다.
		*/;
	}
	 ..중략

8-1. net.daum.controller 패키지에 LoginController.java를 만든다.

```
package net.daum.controller;

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

@Controller
public class LoginController {
```

```
        @GetMapping("/login") //사용자 로그인 페이지
        public void login() {

        }

        @RequestMapping("/accessDenied") //403 접근금지 에러가 났을때
        public void accessDenied() {

        }

        @GetMapping("/logout") //로그아웃 페이지
        public void logout() {

        }
}
```

8-2. '/admin'으로 접근할 때에는 'ADMIN'이라는 권한이 있어야만 하는 설정을 추가해 본다. net.daum.security.SecurityConfig.java에 다음과 같은 코드를 추가한다.

```
@Override
        protected void configure(HttpSecurity http) throws Exception {//configure() 메서드를 오버라이딩을 함.
                //HttpSecurity는 웹과 관련된 다양한 보안설정을 걸어 줄 수 있다.

                log.info("security config .....");//간단한 로그 메시지를 출력한다.


http.authorizeRequests().antMatchers("/guest/**").permitAll();//authorizeRequests()는 시큐리티 처리에서 HttpServletRequest에 해당한다.
        /* antMatchers()에서는 특정한 경로를 지정한다. permitAll()은 모든 사용자가 접근할 수 있다는 것을 의미한다.
         */

http.authorizeRequests().antMatchers("/manager/**").hasRole("MANAGER");
//hasRole()은 특정권한을 가진 사람만이 접근할 수 있다는 것을 의미한다.

http.authorizeRequests().antMatchers("/admin/**").hasRole("ADMIN");//     /admin으로 접근할 때에는 'ADMIN'이라는 권한이 있어야만 하는 설정 추가
                http.formLogin().loginPage("/login");
                /*http.formLogin()은 form 태그기반의 로그인을 지원하겠다는 설정이다.
```

이를 이용하면 별도의 로그인 페이지를 제작하지 않아도 스프링 시큐리티에서 제공하는
* /login 매핑주소로 인식되는 기본 로그인 페이지가 띄워진다.
loginPage("/member_login");을 사용하면 매핑주소가 member_login 인 사용자 즉 커스텀 로그인 페이지를 만들 수 있다.
*/;
}

브라우저에서 '/admin' 경로로 접근하면 브라우저는 자동으로 '/login' 경로로 이동하게 된다. 이때, 지정한 아이디와 패스워드인 'manager/1111'을 입력하고 로그인 하면 403 'Forbidden' 접근 금지 에러가 발생한다.
이러한 경우에 사용자에게 권한이 없음을 알려주고, 로그인 화면으로 이동할 수 있도록 안내 페이지를 작성할 필요가 있다. 이 설정은 HttpSecurity에서 exceptionHandling() 을 이용해서 지정한다.

net.daum.security.SecurityConfig.java 의 configure() 메서드 내부 일부 코드 수정

```
http.formLogin().loginPage("/login");
/*http.formLogin()은 form 태그기반의 로그인을 지원하겠다는 설정이다. 이를 이용하면 별도의 로그인 페이지를 제작하지 않아도 스프링 시큐리티에서 제공하는
* /login 매핑주소로 인식되는 기본 로그인 페이지가 띄워진다.
loginPage("/member_login");을 사용하면 매핑주소가 member_login인 사용자 즉 커스텀 로그인 페이지를 만들 수 있다.
*/

http.exceptionHandling().accessDeniedPage("/accessDenied");//403 접근금지 에러가 났을 때 실행
```

8-3. 로그아웃 처리를 하기 위해서는 net.daum.security.SecurityConfig.java 의 configure() 메서드 내부 일부 코드 수정한다.

```
http.exceptionHandling().accessDeniedPage("/accessDenied");//403 접근금지 에러가 났을 때 실행
http.logout().logoutUrl("/logout").invalidateHttpSession(true);//세션무효화
```

9. JDBC를 이용한 인증 처리

9-1. net.daum.security 패키지에 ZerockUsersService 클래스를 만든다.

```
package net.daum.security;
```

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

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

import lombok.extern.java.Log;
import net.daum.dao.MemberRepository;

@Service //스프링에서 빈으로 처리
@Log
public class ZerockUsersService implements UserDetailsService {
        /* CustomUserDetailsService 별도의 인증/권한 체크를 하는 이유는 jsp등에서 단
순히 사용자 아이디
         * (스프링 시큐리티에서는 username) 정도가 아닌 사용자의 이름이나 이메일
         * 같은 추가적인 정보를 이용하기 위해서 이다.
         */
    @Autowired
    private MemberRepository memberRepo;

   @Autowired
   private HttpServletRequest request;

        @Override
        public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {//회원아이디를 이용해서 UserDetails객체를 반환

                System.out.println(" \n ==============> UserDetailsService 로 접
근함");
        //System.out.println("아이디:"+username);
                return
                        this.memberRepo.findById(username)
                    .filter(member -> member != null) //검색된 회원정보(권한정보)가
있다면
                                                .map(member -> new
ZerockSecurityUser(member,request)).get();//검색된 회원정보(권한정보)를
ZerockSecurityUser 생성자 인자값으로 전달하고
```

```
                        //ZerockSecurityUser 객체 타입을 get()메서드로 구함
        }
}
```

9-2. net.daum.security 패키지의 SecurityConfig에서 ZerockUsersService 자동의존성 주입을 하고 configure() 메서드 일부를 수정한다.

```
@Log
@EnableWebSecurity //스프링 웹 시큐리티로 인식되게 @EnableWebSecurity 애노테이션 추가
public class SecurityConfig extends WebSecurityConfigurerAdapter {

..중략

@Autowired
ZerockUsersService zerockUsersService;

        @Override
        protected void configure(HttpSecurity http) throws Exception {

        http.rememberMe().key("zerock").userDetailsService(zerockUsersService)
                //rememberMe()에서 쿠키값을 암호화 해서 전달하므로 암호의 '키(key)'를 지정하여 사용
                .tokenRepository(getJDBCRepository())
            .tokenValiditySeconds(60*60*24);//쿠키 유효 시간을 초단위로 설정 => 60초*60분*24시간 즉 24시간 쿠키 유효시간 설정

        }

        private PersistentTokenRepository getJDBCRepository() {
                /*      SecurityConfig 에서 rememberMe()를 처리할 때 JdbcTokenRepositoryImpl을 지정해 주어야 하는대 기본적으로
                 *   DataSource가 필요하므로 의존성을 주입한다.
                 */

                JdbcTokenRepositoryImpl repo = new JdbcTokenRepositoryImpl();
                repo.setDataSource(dataSource);
                return repo;
        }
```

```
            //@Autowired//화면에 로그인 페이지가 띄워져도 어떤 아이디나 비번을 입력해도 로그인 실패가 된다. 이런 경우는 로그인 되게 만들기 위해서
            //AuthenticationManagerBuilder을 주입해서 인증에 대한 처리를 해야 한다.
                // public void configureGlobal(AuthenticationManagerBuilder auth) throws
                //Exception {//AuthenticationManagerBuilder는 인증에 대한 다양한 설정을 할 수 있다. 예를 들어 메모리상에 정보만을 이용한다든지,
                //jdbc등을 이용해서 인증 처리가 가능하다. 여기서는 메모리상의 인증 정보를 활용한다.

                // log.info("build Auth global.....");

//auth.inMemoryAuthentication().withUser("manager").password("{noop}1111").roles("MANAGER");
                //사용자 manager,비번 1111,권한 MANAGER 지정
                //Spring Security 4에서는 메모리 내 인증을 사용하여 암호를 인코딩 즉 암호화 하지않고 일반 텍스트로 저장할 수 있었다.
                //Spring Security 5부터는 비번을 인코딩 즉 암호화 해서 저장한다. 그러므로  There is no PasswordEncoder mapped for the id “null”
                //오류를 내지 않기 위해서는 {noop}을 사용해서 비번을 인코딩 즉 암호화해서 처리해야 한다.
                //}
}
```

9-3. net.daum.security 패키지에서 ZerockSecurityUser 클래스를 생성한다.

```
package net.daum.security;

import java.util.ArrayList;
import java.util.List;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;

import lombok.Getter;
import lombok.Setter;
```

```
import net.daum.vo.MemberRole;
import net.daum.vo.MemberVO;

@Setter
@Getter
public class ZerockSecurityUser extends User {

    private static final String ROLE_PREFIX = "ROLE_";

        private MemberVO member;

        public ZerockSecurityUser(MemberVO member,HttpServletRequest request) {
                super(member.getMem_id(), member.getMem_pwd(),
makeGrantedAuthority(member.getRoles()));
                //부모클래스 생성자 인자값으로 아이디,비번,권한 목록을 넘겨줌

                System.out.println("권한이름:" +
makeGrantedAuthority(member.getRoles()).toString());
                HttpSession session=request.getSession();
                session.setAttribute("id", member.getMem_id());
                session.setAttribute("name", member.getMem_name());

                List<GrantedAuthority> list =
makeGrantedAuthority(member.getRoles());//권한 목록
                String total_Auth="";//누적 권한

                for(int i=0;i<list.size();i++) {
                  System.out.println(list.get(i));
                  total_Auth += list.get(i);
                }
                System.out.println("누적권한:"+total_Auth);
                session.setAttribute("auth",total_Auth);
        }

        private static List<GrantedAuthority>
makeGrantedAuthority(List<MemberRole> roles) {//메서드 인자값으로 권한 목록이 넘겨
짐

                List<GrantedAuthority> list = new ArrayList<>();
```

```
                roles.forEach(role -> list.add(new
SimpleGrantedAuthority(ROLE_PREFIX + role.getRoleName())));//권한 목록만큼 반복해
서
                //접두어 "ROLE_"를 붙인 권한이름을 SimpleGrantedAuthority 생성자
인자값으로 전달한 제네릭 타입 객체로 컬렉션에 추가

                return list;
        }
}
```

9-4. 자동 로그인 remember-me 정보를 저장할 DB를 설계한다.

```
--스프링 시큐리티 자동로그인 정보를 유지하는 테이블
create table persistent_logins(
  username varchar2(64) not null --회원아이디
  ,series varchar2(64) primary key --비번
  ,token varchar2(64) not null --토큰정보
  ,last_used timestamp not null --로그인한 날짜 시간
);
```