

Spring Security



학습목표

1. Spring Security 개요
2. 스프링 시큐리티의 설정과 적용
3. 일반적인 로그인 처리
4. Google을 통한 소셜 로그인 처리(X)
5. API 서버와 JWT를 이용한 인증 처리

Spring Security 개요

- Spring Security
 - Spring 애플리케이션의 인증 및 권한 부여를 위해 설계된 강력한 프레임워크로, 웹 개발에 필수적인 보안 기능을 제공
- 주요특징
 - 인증 및 권한 부여 : 사용자 인증 및 액세스 제어를 관리하고 세션 고정
 - 공격으로부터 보호 : 사이트 간 요청 위조(CSRF), 클릭재킹과 같은 일반적인 보안 위협으로부터 보호
 - Spring 에코시스템과의 통합 : 다른 Spring 프로젝트와 원활하게 통합되어 광범위한 사용자 정의 코딩 없이 보안 기능을 보다 쉽게 구현 가능

스프링 시큐리티의 설정과 적용

- Spring Boot와 Spring Security 연동
 - 스프링 시큐리티는 기본적으로는 HttpSession 방식
 - 전통적인 id/pw 기반의 로그인 처리
 - JPA를 이용하는 커스텀 로그인 처리
 - jsp에서 로그인 정보 활용하기

Spring Security 개요

■ 핵심 구성 요소

- Authentication(인증)
 - 사용자의 신원을 검증
 - Spring Security에서 이는 일반적으로 로그인 요청을 처리하는 필터를 통해 처리
 - 인증 필터 : 이 필터는 로그인 요청을 가로채서 인증 프로세스를 적절한 핸들러에 위임
 - SecurityContextHolder : 인증된 사용자에 대한 세부 정보를 포함하는 보안 컨텍스트를 저장
- Authorization
 - 인증된 사용자가 특정 리소스에 액세스할 수 있는 권한이 있는지 여부를 결정
 - 역할 및 권한 : 사용자에게는 애플리케이션 내에서의 액세스 권한을 정의하는 역할이 지정
 - 액세스 제어 목록(ACL) : 세분화된 권한을 설정하여 다양한 수준에서 액세스를 제어

Spring Security 개요

■ 핵심 구성 요소

▪ Architecture

- 아키텍처Spring Security의 아키텍처는 들어오는 요청을 처리하는 필터 체인을 중심으로 수행
- 체인의 각 필터는 인증 또는 권한 부여 검사와 같은 특정 보안 작업을 수행
 - 클라이언트 로그인 시도 : 사용자가 로그인 자격 증명을 제출.
 - 인증 처리 : AuthenticationFilter는 이러한 자격 증명을 처리
 - 보안 컨텍스트 관리 : 인증이 성공하면 사용자 세부 정보가 후속 요청을 위해 SecurityContextHolder에 저장

스프링 시큐리티의 설정과 적용

- Spring Boot와 Spring Security 연동
 - 스프링 시큐리티는 기본적으로는 HttpSession 방식
 - 전통적인 id/pw 기반의 로그인 처리
 - JPA를 이용하는 커스텀 로그인 처리
 - jsp에서 로그인 정보 활용하기

스프링 시큐리티의 설정과 적용

- 프로젝트 생성시에 security 항목을 추가
- 프로젝트 실행시 임시 패스워드 확인 (계정은 user)

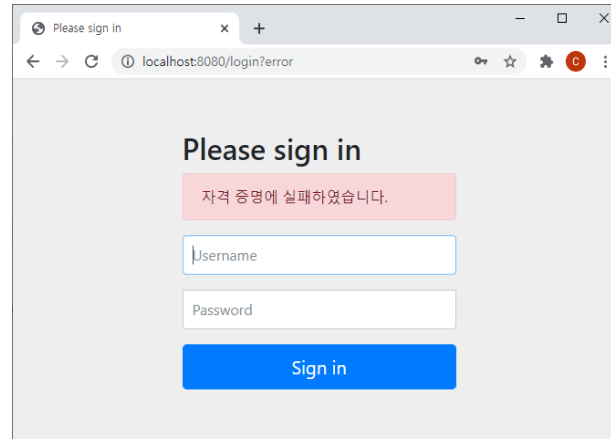
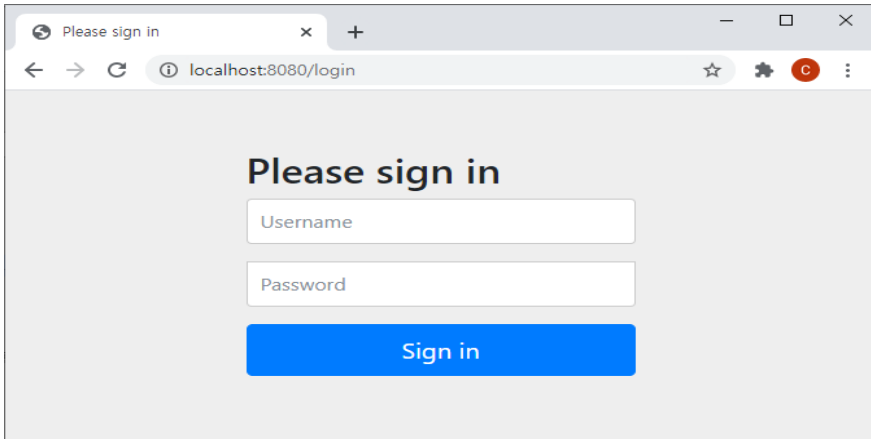
```
2020-10-08 22:21:05.729 INFO 20716 --- [ restartedMain] .s.s.UserDetailsServiceAutoConfiguration :
```

```
Using generated security password: 206f833d-b37e-47fc-94d9-2eaa024d6b4a
```

```
2020-10-08 22:21:05.791 DEBUG 20716 --- [ restartedMain] edFilterInvocationSecurityMetadataSource : Adding web access
```

```
2020-10-08 22:21:05.795 DEBUG 20716 --- [ restartedMain] o.s.s.w.a.i.FilterSecurityInterceptor : Validated configur
```

http://localhost:8080/login



스프링 시큐리티의 설정과 적용

- 시큐리티 설정 클래스 작성
 - CustomSecurityConfig클래스 추가

```
@Configuration
@Log4j2
@EnableMethodSecurity
@RequiredArgsConstructor
public class CustomSecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        log.info("security config.....");

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

- 로그 레벨 조정

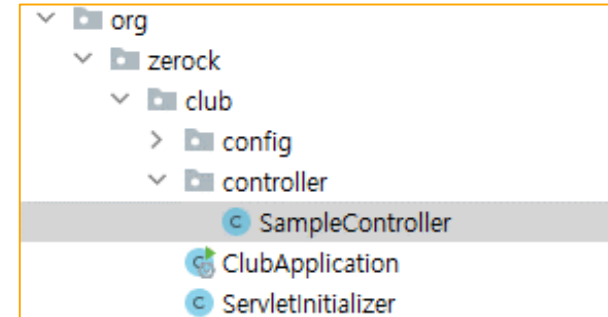
- application.properties 파일

```
logging.level.org.springframework=info  
logging.level.org.zerock=debug  
logging.level.org.springframework.security=trace
```

스프링 시큐리티의 설정과 적용

■ 확인을 위한 SampleController

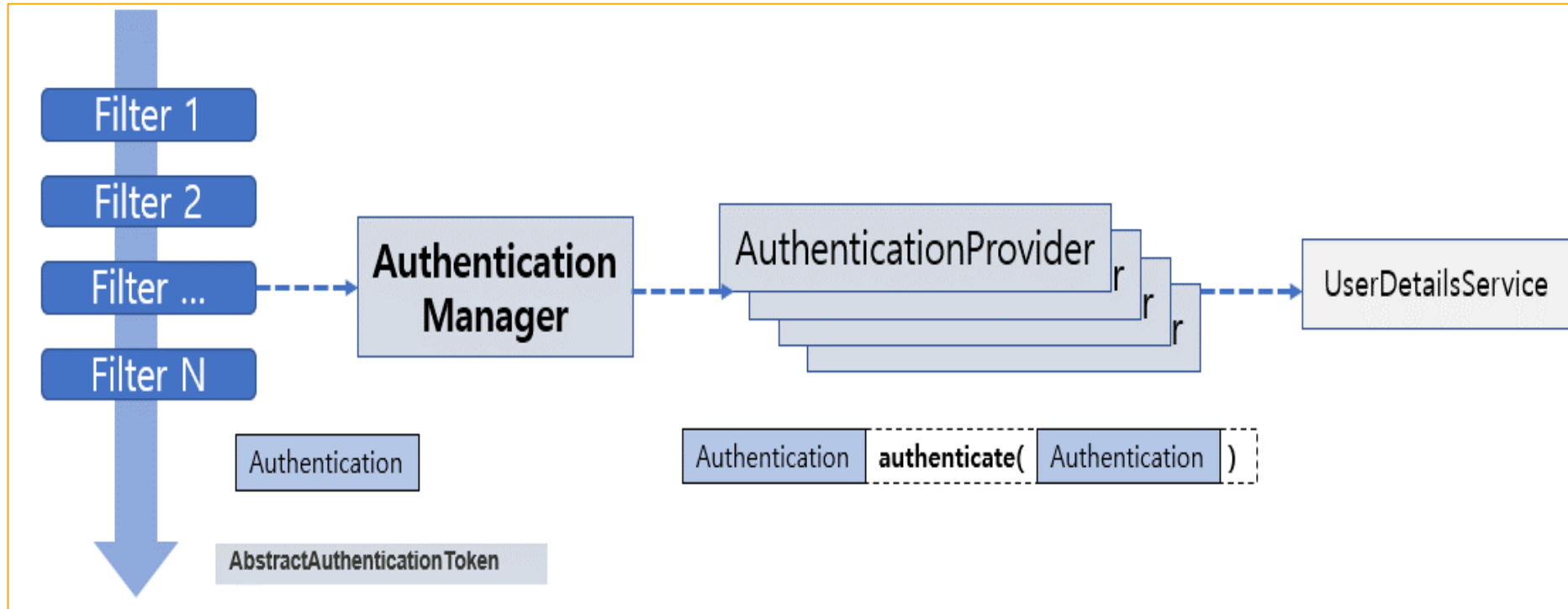
```
@Controller
@Log4j2
@RequiredArgsConstructor
public class SampleController {
    @GetMapping("/")
    public String home(){
        return "home";
    }
    @GetMapping("/user/login")
    public void login(){
    }
    @GetMapping("/all")
    public void exAll(){
        log.info("exAll.....");
    }
    @GetMapping("/member")
    public void exMember(){
        log.info("exMember.....");
    }
    @GetMapping("/admin")
    public void exAdmin(){
        log.info("exAdmin.....");
    }
}
```



- 로그인하지 않은 사용자도 접근할 수 있는
'/all', '/', '/user/login'
- 로그인한 사용자만이 접근할 수 있는
'/member'
- 관리자(admin) 권한이 있는 사용자만이 접근할 수 있는
'/admin'

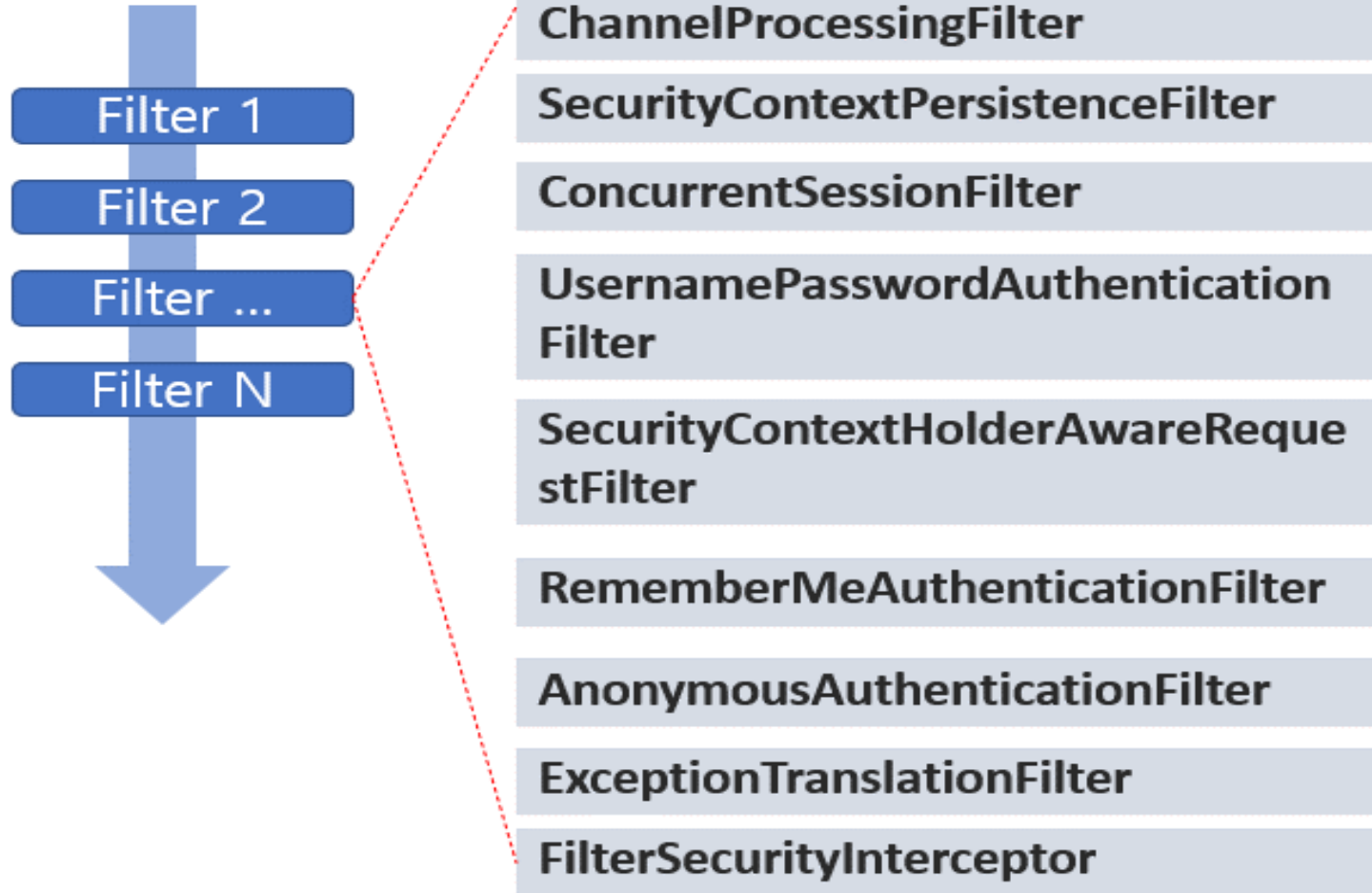
스프링 시큐리티의 설정과 적용

- 스프링 시큐리티 용어와 흐름
 - 기본적으로 필터를 이용해서 동작
 - 필터와 AuthenticationManager 등의 객체를 이용해서 동작



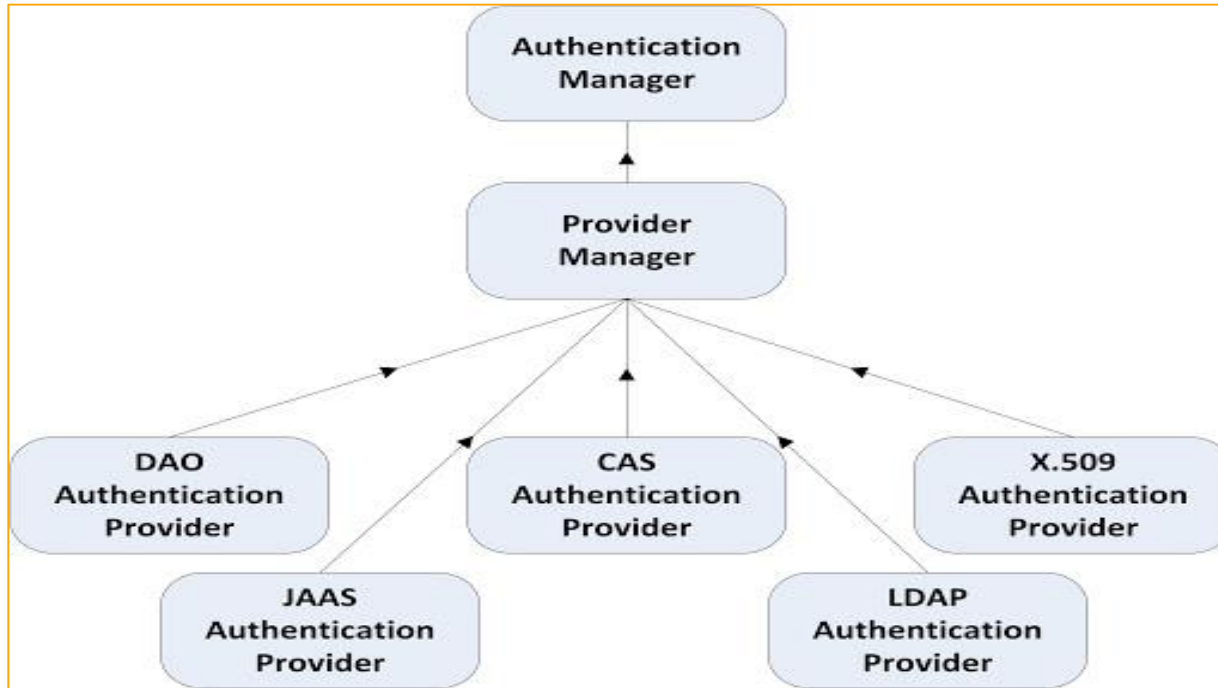
스프링 시큐리티의 설정과 적용

필터와 필터 체이닝



스프링 시큐리티의 설정과 적용

- 인증을 위한 AuthenticationManager
 - 인증 매니저
 - 내부적으로 AuthenticationProvider와 연계되어 동작



스프링 시큐리티의 설정과 적용

■ PasswordEncoder

- 스프링 부트 2.0 부터는 반드시 필요
- 인터페이스이므로 구현하거나 구현된 클래스 이용
- BCryptPasswordEncoder
 - 패스워드 암호화 전용
 - 동일한 메시지도 매번 다르게 암호화 생성
 - 복호화 불가
 - 올바르게 암호화 된 것인지만 확인

org.springframework.security.crypto.password

Interface PasswordEncoder

All Known Implementing Classes:

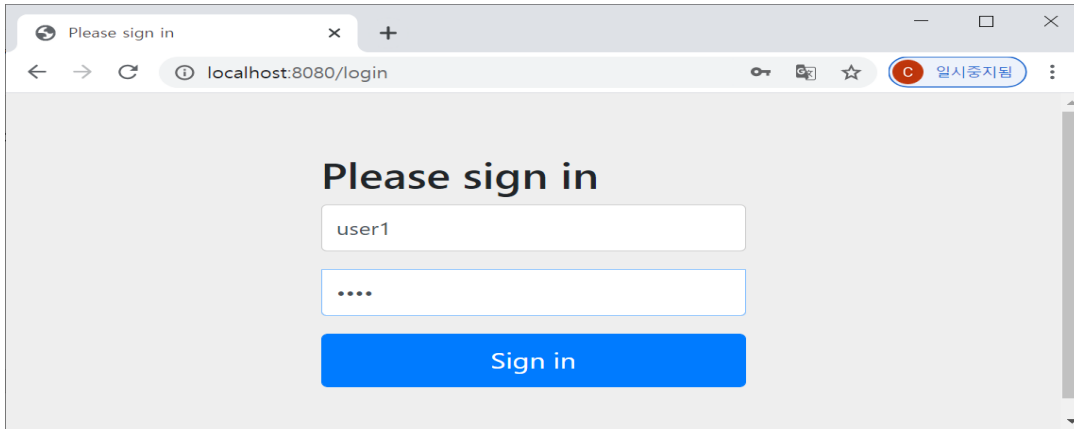
AbstractPasswordEncoder, Argon2PasswordEncoder, BCryptPasswordEncoder, DelegatingPasswordEncoder, LdapShaPasswordEncoder, Md4PasswordEncoder, MessageDigestPasswordEncoder, NoOpPasswordEncoder, Pbkdf2PasswordEncoder, SCryptPasswordEncoder, StandardPasswordEncoder

@Bean

```
PasswordEncoder passwordEncoder(){  
    return new BCryptPasswordEncoder();  
}
```

스프링 시큐리티의 설정과 적용

- AuthenticationManager 설정
 - 우선은 단순히 로그인 가능하도록 설정하고 추후에 변경
 - '/login'으로 동작 확인



스프링 시큐리티의 설정과 적용

■ 인가가 필요한 리소스 설정

@Bean

```
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
    log.info("security config.....");  
  
    return http  
        .authorizeHttpRequests(authorizeHttpRequestsConfigurer -> authorizeHttpRequestsConfigurer  
            .dispatcherTypeMatchers(DispatcherType.FORWARD).permitAll()  
            .requestMatchers("/login", "/sinup", "/user/**", "/", "/all").permitAll()  
            .requestMatchers("/admin/**").hasAuthority("ADMIN")  
            .anyRequest().authenticated())  
  
        .formLogin(formLoginConfigurer -> formLoginConfigurer  
            .loginPage("/user/login")  
            .loginProcessingUrl("/loginProcess")  
            .usernameParameter("username")  
            .passwordParameter("password")  
            .defaultSuccessUrl("/")  
            .permitAll())  
  
        .logout(logoutConfigurer -> logoutConfigurer  
            .logoutUrl("/logout")  
            .logoutSuccessUrl("/")  
            .invalidateHttpSession(true)  
            .clearAuthentication(true))  
        .build();  
}
```

스프링 시큐리티의 설정과 적용

■ CSRF 설정

- Cross Site Request Forgery – 크로스 사이트 요청 위조
- 스프링 시큐리티를 적용하면 기본적으로 CSRF 방지를 위한 토큰(CSRF토큰)이 사용됨
- 세션마다 다른 CSRF토큰 값이 생성
- GET방식을 제외한 모든 요청에 대해서 CSRF토큰이 필수적으로 필요
- `csrf().disable()`을 통해서 비활성화 가능
- CSRF토큰이 비활성화 되면 보안상 위험할 수 있으므로 신중하게 결정

- CSRF 방지를 비활성화해야 하는 경우
 - REST API 서버와 같이 stateless한 서비스를 제공할 때
 - 토큰 기반 인증(예: JWT)을 사용하는 경우
 - 테스트 환경에서 임시로 비활성화할 때

스프링 시큐리티의 설정과 적용

@Bean

```
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception{
    return http
        .csrf(httpSecurityCsrfConfigurer -> httpSecurityCsrfConfigurer.disable()) //CSRF 설정
        .cors(httpSecurityCorsConfigurer -> httpSecurityCorsConfigurer.disable())
        .authorizeHttpRequests(authorizeHttpRequestsConfigurer -> authorizeHttpRequestsConfigurer
            .dispatcherTypeMatchers(DispatcherType.FORWARD).permitAll()
            .requestMatchers("/login", "/sinup", "/user/**", "/").permitAll()
            .requestMatchers("/admin/**").hasAuthority("ADMIN")
            .anyRequest().authenticated())
        .formLogin(formLoginConfigurer -> formLoginConfigurer
            .loginPage("/user/login")
            .loginProcessingUrl("/loginProcess")
            .usernameParameter("username")
            .passwordParameter("password")
            .defaultSuccessUrl("/")
            .permitAll())
        // .logout(Customizer.withDefaults())
        .logout(logoutConfigurer -> logoutConfigurer
            .logoutUrl("/logout")
            .logoutSuccessUrl("/")
            .invalidateHttpSession(true)
            .clearAuthentication(true))
        .build();
}
```

스프링 시큐리티의 설정과 적용

■ 프로젝트를 위한 JPA처리

회원(User) 정보

- id(PK)
- username(아이디 역할)
- 패스워드
- email
- role
- createDate(등록일)

권한(ClubMemberRole)

- USER: 일반 회원
- MANGER: 중간 관리 회원
- ADMIN: 총괄 관리자

@NoArgsConstructor

@AllArgsConstructor

@Data

@Entity

public class User {

 @Id

 @GeneratedValue(strategy = GenerationType.IDENTITY)

 private Long id;

 @Column(nullable = false)

 private String username;

 private String password;

 private String email;

 private String role;

}

스프링 시큐리티의 설정과 적용(Board, Comment Entity)

```
@Getter
@Setter
@Entity(name="tbl_board3")
public class Board {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long num;
    private String title;
    private String writer;
    private String content;
    @CreationTimestamp
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name="regdate")
    private Date regdate;
    private Long hitcount;
    private Long replycnt;
    @OneToMany(mappedBy = "board",
        fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JsonIgnoreProperties("board")
    private List<Comment> comments;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name="user_id")
    private User user; // 유저 정보

    @PrePersist
    public void prePersist() {
        this.hitcount= this.hitcount==null? 0 : this.hitcount;
        this.replycnt= this.replycnt==null? 0 : this.replycnt;
    }
}
```

```
@Getter
@Setter
@Entity(name="tbl_comment3")
public class Comment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long cnum;
    private String content;
    @CreationTimestamp
    @Temporal(TemporalType.TIMESTAMP)
    @JsonFormat(pattern="yyyy-MM-dd")
    private Date regdate;

    @ManyToOne
    @JoinColumn(name = "bnum")
    private Board board;

    @ManyToOne
    @JoinColumn(name="user_id")
    private User user;
}
```

스프링 시큐리티의 설정과 적용

- Repository 설정

```
public interface CommentRepository extends JpaRepository<Comment, Long> {  
}
```

```
public interface BoardRepository extends JpaRepository<Board, Long> {  
}
```

```
public interface UserRepository extends JpaRepository<User, Long> {  
    User findByUsername(String username);  
}
```

스프링 시큐리티의 설정과 적용

■ service 설정

```
public interface BoardService {  
    void insert(Board board, User  
user);  
    public List<Board> list();  
    public Board findById(Long num);  
    public void update(Board board);  
    public void delete(Long num);  
}
```

```
@Service  
@RequiredArgsConstructor  
public class BoardServiceImpl implements BoardService{  
  
    private final BoardRepository boardRepository;  
    @Override  
    public void insert(Board board, User user) {  
        board.setUser(user);  
        boardRepository.save(board);  
    }  
  
    @Override  
    public List<Board> list() {  
        return boardRepository.findAll();  
    }  
  
    @Override  
    public Board findById(Long num) {  
        Board board = boardRepository.findById(num).get();  
        board.setHitcount(board.getHitcount()+1);  
        return board;  
    }  
  
    @Override  
    public void update(Board board) {  
        Board b = boardRepository.findById(board.getNum()).get();  
        b.setContent(board.getContent());  
        b.setTitle(board.getTitle());  
    }  
  
    @Override  
    public void delete(Long num) {  
        boardRepository.deleteById(num);  
    }  
}
```

■ 시큐리티를 위한 UserDetailsService

- 개발자가 원하는 방식으로 로그인을 처리하기 위해서 구현하는 인터페이스
- User라는 용어는 키워드처럼 사용됨
- username이 실제로는 id에 해당
- username/password가 동시에 사용되는 방식이 아니므로 주의
- 인증이 끝나면 인가 처리

Method Summary

All Methods

Instance Methods

Abstract Methods

Modifier and Type

Method and Description

`UserDetails`

`loadUserByUsername(java.lang.String username)`

Locates the user based on the username.

@Data

```
public class PrincipalDetails implements UserDetails {
```

```
    private User user;
```

```
    public PrincipalDetails(User user){
```

```
        this.user= user;
```

```
    }
```

@Override

```
    public Collection<? extends GrantedAuthority> getAuthorities() {
```

```
        Collection<GrantedAuthority> collection=new ArrayList<GrantedAuthority>();
```

```
        collection.add(()->{return user.getRole();});
```

```
        return collection;
```

```
    }
```

@Override

```
    public String getPassword() {
```

```
        return user.getPassword();
```

```
    }
```

@Override

```
    public String getUsername() {
```

```
        return user.getUsername();
```

```
    }
```

@Override

```
    public boolean isAccountNonExpired() {
```

```
        return true;
```

```
    }
```

@Override

```
    public boolean isAccountNonLocked() {
```

```
        return true;
```

```
    }
```

@Override

```
    public boolean isCredentialsNonExpired()
```

```
    {
```

```
        return true;
```

```
    }
```

@Override

```
    public boolean isEnabled() {
```

```
        return true;
```

```
    }
```

```
}
```

- loadByUsername() – username이라는 회원 아이디로 UserDetails 타입의 객체를 반환
- UserDetails인터페이스를 이용해서 구할 수 있는 데이터

- getAuthorities() - 사용자가 가지는 권한에 대한 정보
 - getPassword() - 인증을 마무리하기 위한 패스워드 정보
 - getUsername() - 인증에 필요한 아이디와 같은 정보
 - 계정 만료 여부 - 더이상 사용이 불가능한 계정인지 알 수 있는 정보
- 계정 잠김 여부 – 현재 계정의 잠김 여부

- 기존 구조에서 UserDetails를 처리하는 방식

- 기존의 DTO클래스에 UserDetails 인터페이스를 구현하는 방법
- DTO와 같은 개념으로 별도의 클래스를 구성하고 이를 활용하는 방법

- org.springframework.security.core.userdetails.User 클래스를 상속하는 DTO

■ UserDetailsService 구현

```
@Service
@Log4j2
@RequiredArgsConstructor
public class PrincipalDetailsService implements UserDetailsService {

    private final UserRepository userRepository;

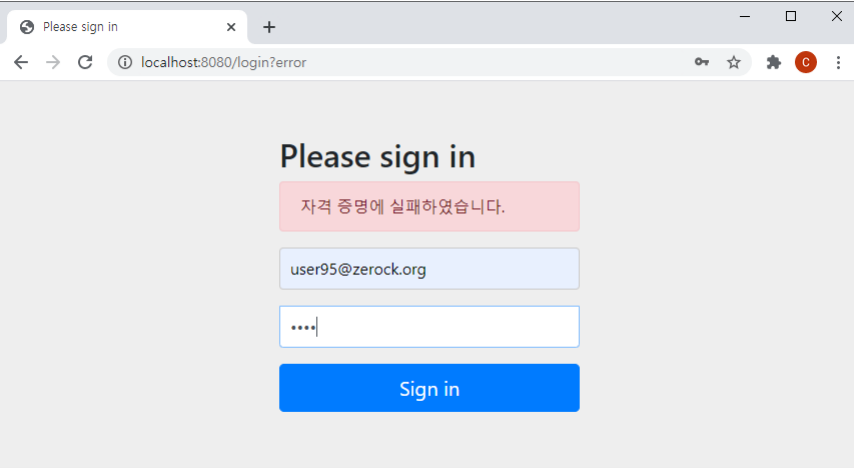
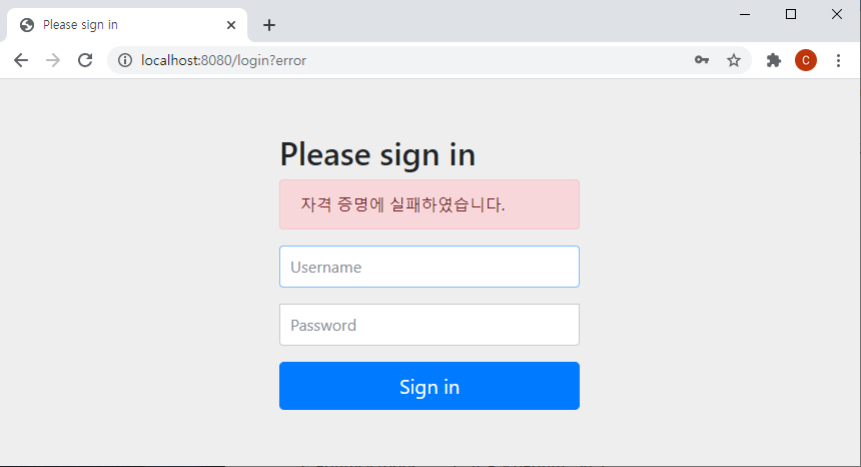
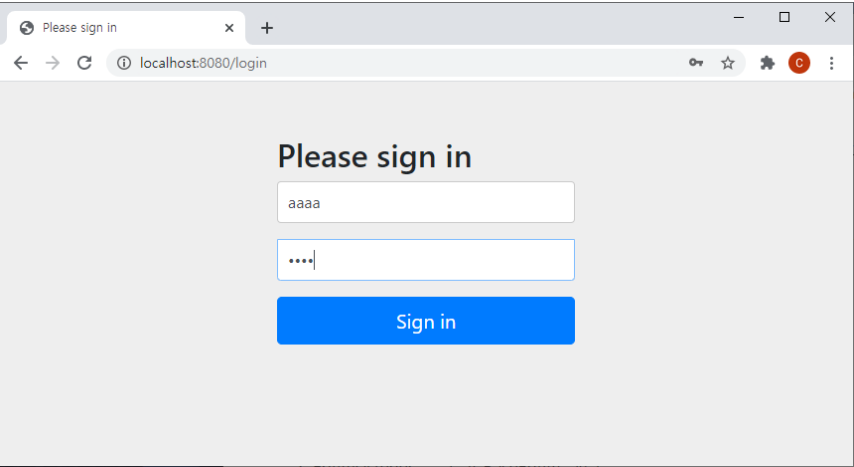
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        log.info("loadUserByUsername");
        User user=userRepository.findByUsername(username);
        if(user==null) return null;
        PrincipalDetails puser=new PrincipalDetails(user);
        log.info(puser);
        return puser;
    }
}
```

■ SecurityConfig 수정

- 인증관리자 Bean 추가

@Bean

```
public AuthenticationManager authenticationManager(HttpSecurity http, BCryptPasswordEncoder  
bCryptPasswordEncoder, UserDetailsService userDetailsService)  
    throws Exception {  
  
    AuthenticationManagerBuilder authenticationManagerBuilder =  
        http.getSharedObject(AuthenticationManagerBuilder.class);  
    authenticationManagerBuilder.userDetailsService(userDetailsService).passwordEncoder(bCryptPasswordEncoder);  
    return authenticationManagerBuilder.build();  
}
```



■ 컨트롤러에서 출력

```
@Controller
@RequestMapping("/admin")
public class AdminController {
    @GetMapping("/list")
    public void List(){

    }
}
```

```
@Controller
@RequestMapping("/board")
public class BoardController {
    @Autowired
    private BoardService boardService;

    @GetMapping("insert")
    public String insert() {
        return "/board/register";
    }
    @PostMapping("/insert")
    public String insert(Board board,
        @AuthenticationPrincipal PrincipalDetails principal) {
        boardService.insert(board,principal.getUser() );
        return "redirect:/board/list";
    }
    @GetMapping("/{"/view“, “/modify”})
    public void view(@RequestParam Long num, Model model) {
        model.addAttribute("board", boardService.findById(num));
        return "/board/view";
    }
    @GetMapping("/list")
    public String list(Model model) {
        model.addAttribute("lists", boardService.list());
        return "/board/list";
    }
    @PutMapping("/update")
    public String update(Board board) {
        boardService.update(board);
        return "redirect:/board/view?num="+board.getNum();
    }
    @GetMapping("/delete")
    public String delete(@RequestParam Long num) {
        boardService.delete(num);
        return "redirect:/board/list";
    }
}
```

■ UserController

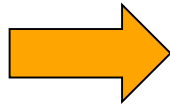
@RequiredArgsConstructor

```
public class UserController {  
    private final UserRepository userRepository;  
    private final BCryptPasswordEncoder bCryptPasswordEncoder;
```

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

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

```
    @GetMapping("login")  
    public void login(){  
  
    }
```



```
    @PostMapping("register")  
    public String register(User user) {  
        System.out.println("회원가입 진행 : " + user);  
        String rawPassword = user.getPassword();  
        String encPassword = bCryptPasswordEncoder.encode(rawPassword);  
        user.setPassword(encPassword);  
        user.setRole("USER");  
        userRepository.save(user);  
        return "redirect:/";  
    }  
}
```