

Spring Security



학습목표

1. 스프링 시큐리티의 설정과 적용
2. 일반적인 로그인 처리
3. Google을 통한 소셜 로그인 처리
4. API 서버와 JWT를 이용한 인증 처리

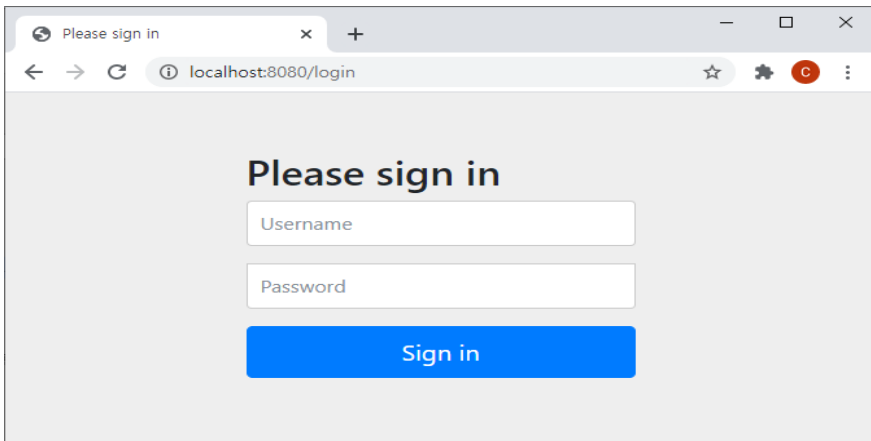
스프링 시큐리티의 설정과 적용

- 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

- 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>

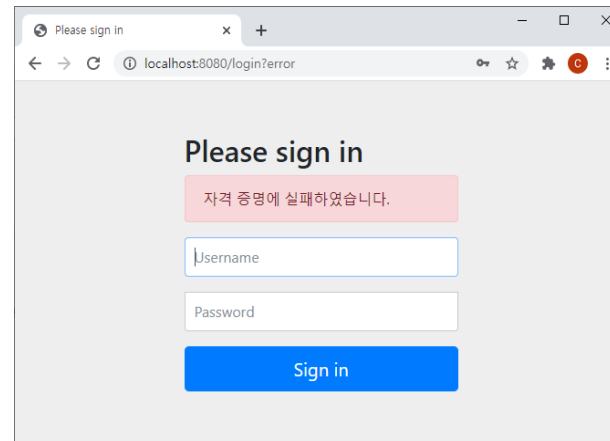


Please sign in

Username

Password

Sign in



Please sign in

자격 증명에 실패하였습니다.

Username

Password

Sign in

■ 시큐리티 설정 클래스 작성

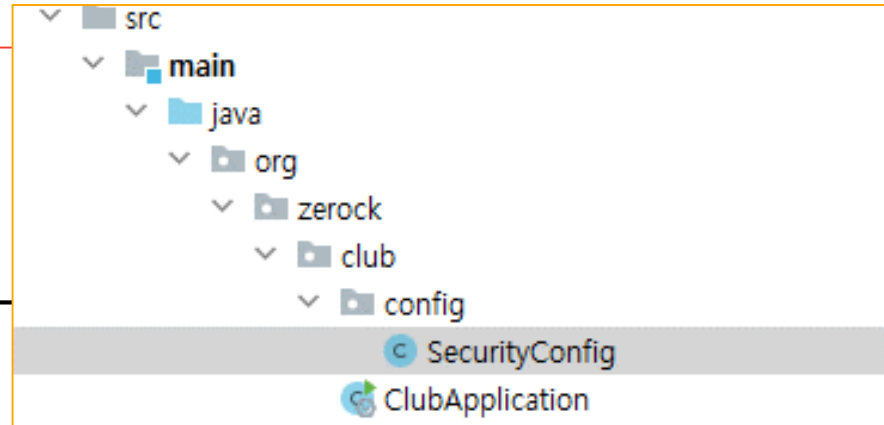
- SecurityConfig클래스 추가

```
package org.zerock.club.config;

import lombok.extern.log4j.Log4j2;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
@EnableWebSecurity //SpringSecurityFilterChain을 자동으로 포함
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Bean
    public BCryptPasswordEncoder encodePwd() { //비밀번호 암호화 저장
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //http요청 보안 설정, 페이지권한설정, 로그인페이지 설정, 로그아웃 페이지 설정, 로그아웃 메소드 설정
    }
}
```



■ 확인을 위한 SampleController

SampleController 클래스

```
package org.zerock.club.controller;
```

```
import lombok.extern.log4j.Log4j2;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
@Controller
```

```
@Log
```

```
@RequestMapping("/sample/")
```

```
public class SampleController {
```

```
    @GetMapping("/all")
```

```
    public void exAll(){
```

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

```
    }
```

```
    @GetMapping("/member")
```

```
    public void exMember(){
```

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

```
    }
```

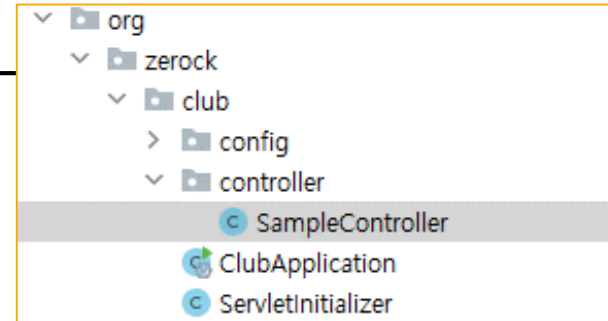
```
    @GetMapping("/admin")
```

```
    public void exAdmin(){
```

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

```
    }
```

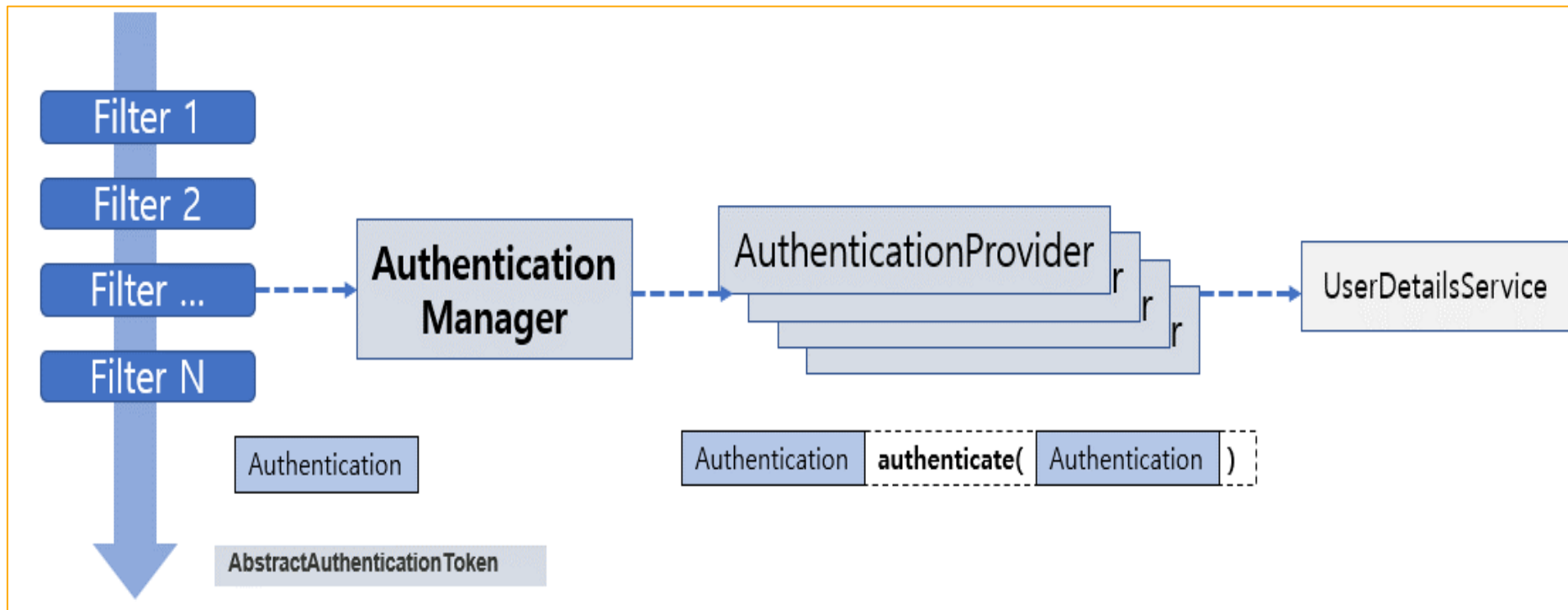
```
}
```



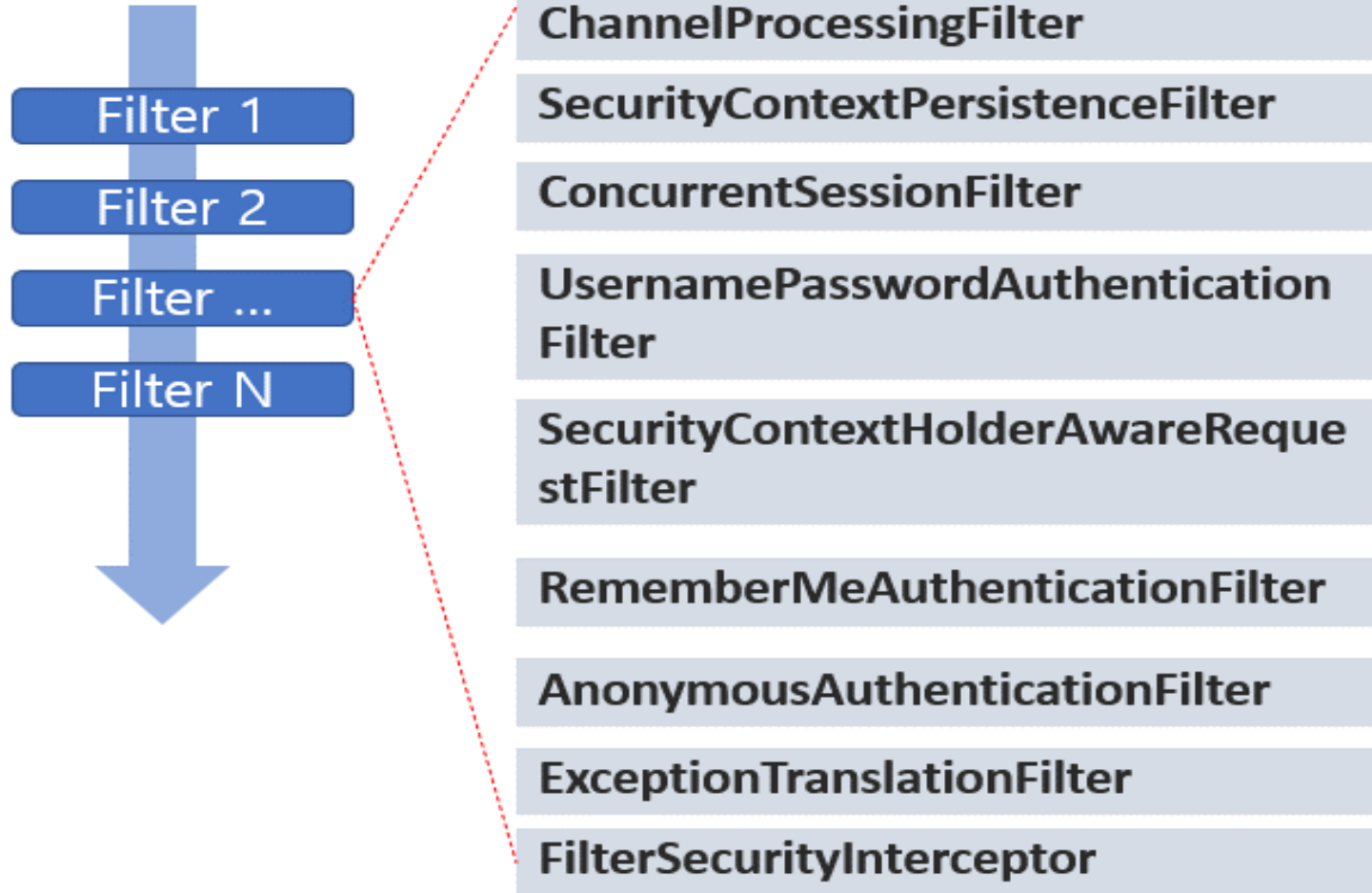
- 로그인하지 않은 사용자도 접근할 수 있는
'/sample/all'
- 로그인한 사용자만이 접근할 수 있는
'/sample/member'
- 관리자(admin) 권한이 있는 사용자만이 접근할 수 있는
'/sample/admin'

■ 스프링 시큐리티 용어와 흐름

- 기본적으로 필터를 이용해서 동작
- 필터와 AuthenticationManager 등의 객체를 이용해서 동작

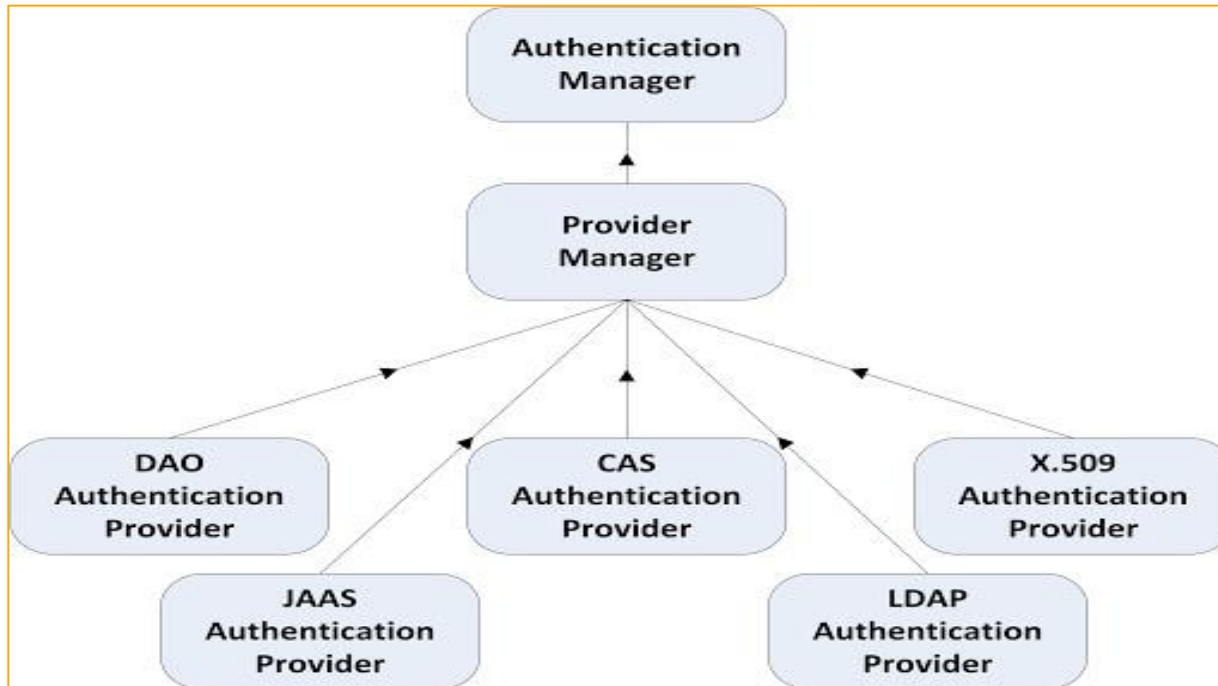


■ 필터와 필터 체이닝



■ 인증을 위한 AuthenticationManager

- 인증 매니저
- 내부적으로 AuthenticationProvider와 연계되어 동작



■ 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:

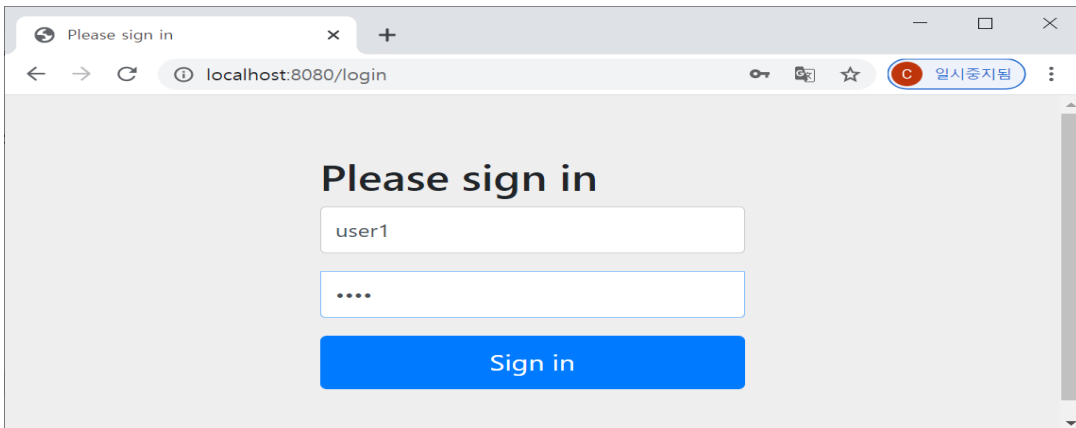
AbstractPasswordEncoder, Argon2PasswordEncoder, BCryptPasswordEncoder, DelegatingPasswordEncoder, LdapShaPasswordEncoder, Md4PasswordEncoder, MessageDigestPasswordEncoder, NoOpPasswordEncoder, Pbkdf2PasswordEncoder, SCryptPasswordEncoder, StandardPasswordEncoder

@Bean

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

■ AuthenticationManager 설정

- 우선은 단순히 로그인만 가능하도록 설정하고 추후에 변경
- '/login'으로 동작 확인



@Override

```
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
```

```
    auth.inMemoryAuthentication().withUser("user1") //사용자 계정은 user1  
    .password("$2a$10$qbTVRGiC8RePIsMz4z/QP.LjBmLOMGXBCkmW2comzfNaoeidd5/aa") //1111 패스워드 인코딩  
    .roles("USER");
```

```
}
```

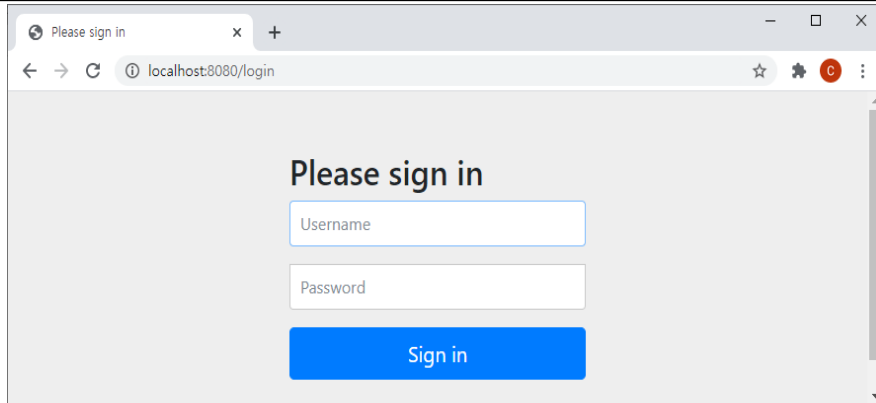
■ 인가가 필요한 리소스 설정

SecurityConfig 클래스 일부

@Override

```
protected void configure(HttpSecurity http) throws Exception {  
    http.authorizeRequests()  
        .antMatchers("/sample/all").permitAll()  
        .antMatchers("/sample/member").hasRole("USER");  
}
```

http.formLogin(); //인가/인증에 문제시 로그인 화면



■ 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

■ 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

```
@Data
@Entity
public class User {
    @Id // primary key
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String username;
    private String password;
    private String email;
    private String role; //ROLE_USER, ROLE_ADMIN, ROLE_MAMAGER
    @CreationTimestamp
    private Timestamp createDate;
}
```

■ 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.

- 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

ClubAuthMemberDTO 클래스 선언

@Log4j2

@Getter

@Setter

@ToString

```
public class ClubAuthMemberDTO extends User {
```

```
    private String email;
```

```
    private String name;
```

```
    private boolean fromSocial;
```

```
    public ClubAuthMemberDTO(String username, String password, boolean fromSocial,  
                             Collection<? extends GrantedAuthority> authorities) {
```

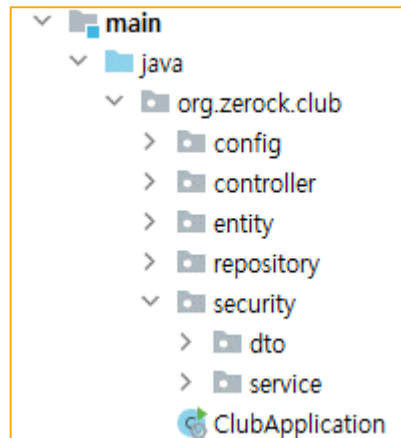
```
        super(username, password, authorities);
```

```
        this.email = username;
```

```
        this.fromSocial = fromSocial;
```

```
    }
```

```
}
```



■ UserDetailsService 구현

ClubUserDetailsService 클래스

```
package org.zerock.club.security.service;

import lombok.extern.log4j.Log4j2;
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;

@Log4j2
@Service
public class ClubUserDetailsService implements UserDetailsService {

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        Log.info("ClubUserDetailsService loadUserByUsername " + username);
        return null;
    }
}
```

■ SecurityConfig 수정

- ClubUserDetailsService가 빈으로 등록되므로 별도의 설정없이 기존의 AuthenticationManagerBuilder를 이용하는 메서드는 필요없음

```
@Configuration
@Log4j2
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Bean
    PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/sample/all").permitAll()
            .antMatchers("/sample/member").hasRole("USER");
        http.formLogin();
    }

    // @Override 더이상 사용하지 않는다.
    // protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    // }
}
```

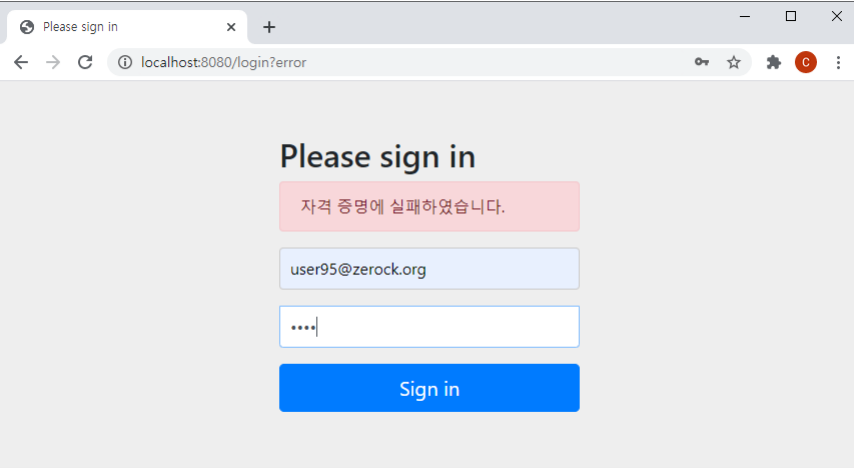
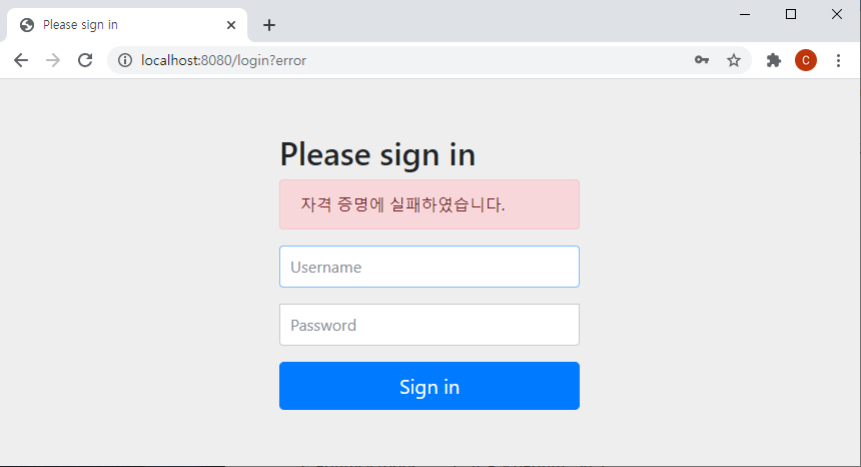
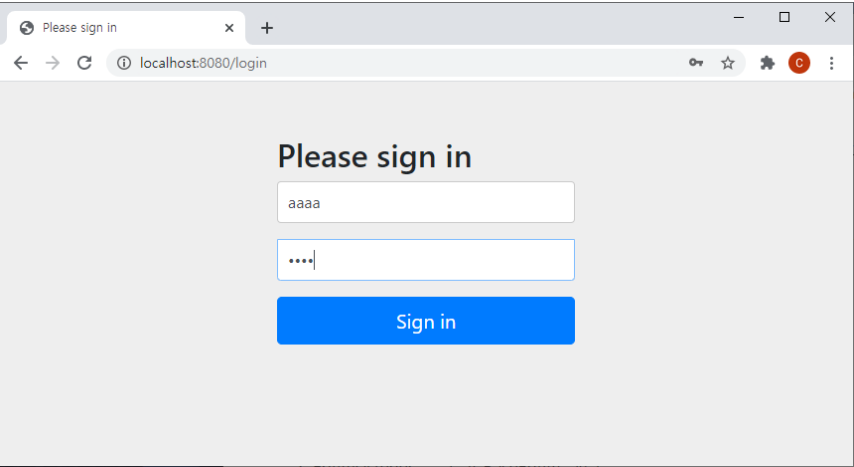
■ ClubMemberRepository 연동

```
@Log4j2
@Service
@RequiredArgsConstructor
public class ClubUserDetailsService implements UserDetailsService {
    private final ClubMemberRepository clubMemberRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        Log.info("ClubUserDetailsService loadUserByUsername " + username);
        Optional<ClubMember> result = clubMemberRepository.findByEmail(username, false);

        if(result.isEmpty()){
            throw new UsernameNotFoundException("Check User Email or from Social ");
        }
        ClubMember clubMember = result.get();
        Log.info("-----"+clubMember);

        ClubAuthMemberDTO clubAuthMember = new ClubAuthMemberDTO(
            clubMember.getEmail(),
            clubMember.getPassword(),
            clubMember.isFromSocial(),
            clubMember.getRoleSet().stream()
                .map(role -> new SimpleGrantedAuthority("ROLE_"+role.name()))
                .collect(Collectors.toSet())
        );
        clubAuthMember.setName(clubMember.getName());
        clubAuthMember.setFromSocial(clubMember.isFromSocial());
        return clubAuthMember;
    }
}
```



■ 컨트롤러에서 출력

SampleController 일부

```
@GetMapping("/member")
public void exMember(@AuthenticationPrincipal ClubAuthMemberDTO clubAuthMember){
    Log.info("exMember.....");
    Log.info("-----");
    Log.info(clubAuthMember);
}
```

```
exMember.....
-----
ClubAuthMemberDTO(email=user95@zerock.org, name=사용자95, fromSocial=false)
```