Spring Boot Application

스프링 부트 응용

스프링 시큐리티와 JPA 연동

한국기술교육대학교
온라인평생교육원

학습내용

- 스프링 시큐리티와 JPA 연동
- 비밀번호 암호화

학습목표

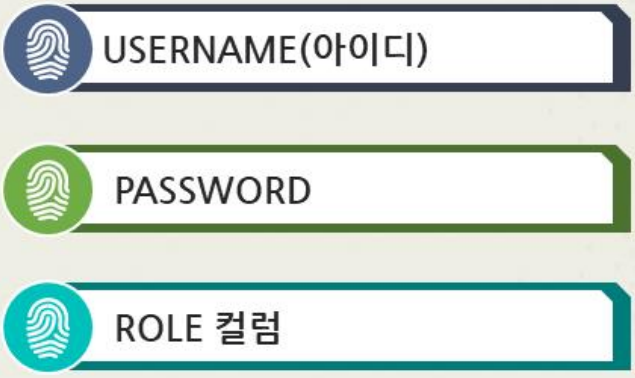
- JPA를 연동하여 시큐리티를 구현할 수 있다.
- 비밀번호를 암호화하여 저장하고 사용할 수 있다.

스프링 시큐리티와 JPA 연동

1 데이터베이스를 연동하는 사용자 인증

1 데이터베이스 설정

- 스프링 시큐리티 회원 테이블에 필요한 것



1 데이터베이스를 연동하는 사용자 인증

1 데이터베이스 설정

- 스프링 시큐리티 회원 테이블에 필요한 것

ENABLED 컬럼

- ENABLED에는 계정의 활성화 여부가 저장됨

스프링 시큐리티와 JPA 연동

1 데이터베이스를 연동하는 사용자 인증

① 데이터베이스 설정

- 데이터베이스에 회원(MEMBER) 테이블을 생성함
- 인증에 필요한 회원 정보를 저장함

```
DROP TABLE MEMBER;

CREATE TABLE MEMBER(
    USERNAME VARCHAR2(100) PRIMARY KEY,
    PASSWORD VARCHAR2(100),
    NAME VARCHAR2(30),
    ROLE VARCHAR2(12),
    ENABLED BOOLEAN
);

INSERT INTO MEMBER VALUES('user', 'user123', '회원', 'ROLE_USER', TRUE);
INSERT INTO MEMBER VALUES('test', 'test123', '매니저', 'ROLE_MANAGER', TRUE);
```

1 데이터베이스를 연동하는 사용자 인증

② 시큐리티 설정 수정

- 데이터베이스 기반으로 인증 처리 시

- 시큐리티 설정 클래스(SecurityConfig)의 authenticate 메소드를 수정함

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private DataSource dataSource;

    @Autowired
    public void authenticate(AuthenticationManagerBuilder auth) throws Exception {
        String query1 = "select username, concat('{noop}', password) password, true enabled "
                      + "from member where username=?";
        String query2 = "select username, role from member where username=?";

        auth.jdbcAuthentication().dataSource(dataSource)
            .usersByUsernameQuery(query1)
            .authoritiesByUsernameQuery(query2);
    }
```

스프링 시큐리티와 JPA 연동

1 데이터베이스를 연동하는 사용자 인증

〈3〉 JDBC 기반의 인증 처리

- JDBC 기반으로 인증 처리 시

  • AuthenticationManagerBuilder의 jdbcAuthentication 메소드를 사용함

1 데이터베이스를 연동하는 사용자 인증

〈3〉 JDBC 기반의 인증 처리

회원 인증 시 회원 정보 조회에 필요한 두 개의 쿼리	
usersByUsernameQuery	• 특정 아이디(username)에 해당하는 사용자 정보를 조회함 • 사용자 조회에 실패하면 더 이상 다음 단계로 진행되지 않음
authoritiesByUsernameQuery	• 특정 아이디(username)에 해당하는 사용자의 권한 목록을 조회함

스프링 시큐리티와 JPA 연동

2 JPA 적용하기

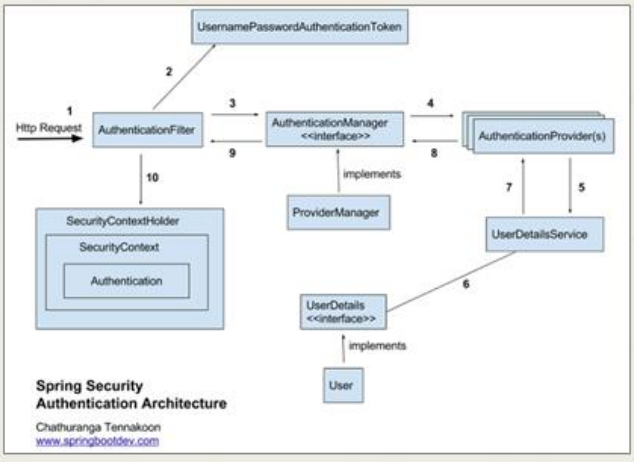
〈1〉 스프링 시큐리티 인증 아키텍처

- 스프링 시큐리티에서 JPA를 연동하기 위해서는 스프링 시큐리티의 인증 아키텍처를 이해해야 함

2 JPA 적용하기

〈1〉 스프링 시큐리티 인증 아키텍처

- springbootdev.com 사이트에서 제공하는 스프링 시큐리티 인증 아키텍처



[출처 : springbootdev.com]

스프링 시큐리티와 JPA 연동

2 JPA 적용하기

② 스프링 시큐리티 인증 과정

- 스프링 시큐리티의 처리 순서

1. 사용자가 아이디와 비밀번호를 입력하고 인증을 요청(Http Request)함
2. AuthenticationFilter는 사용자가 입력한 정보를 추출하여 UsernamePasswordAuthenticationToken 객체를 생성함(username, password 저장)

2 JPA 적용하기

② 스프링 시큐리티 인증 과정

- 스프링 시큐리티의 처리 순서

3. AuthenticationFilter는 AuthenticationManager(ProviderManager)에게 UsernamePasswordAuthenticationToken을 전달하면서 인증을 요청함
4.
   - AuthenticationManager는 AuthenticationProvider에게 실질적인 인증을 위임함
   - 인증에 성공한 경우 검색한 회원 정보를 기반으로 Authentication 객체를 리턴함
   - 실패하면 예외를 발생시킴

스프링 시큐리티와 JPA 연동

2 JPA 적용하기

〈2〉 스프링 시큐리티 인증 과정

- 스프링 시큐리티의 처리 순서

5 다양한 AuthenticationProvider들은 기본적으로 UserDetailsService의 loadUserByUsername 메소드를 통해 검색한 사용자 정보와 AuthenticationManager가 전달한 UsernamePasswordAuthenticationToken을 비교함

6 UserDetailsSerivce는 검색 결과를 기반으로 UserDetails 인터페이스를 구현한 UserDetails(User) 객체를 생성함

2 JPA 적용하기

〈2〉 스프링 시큐리티 인증 과정

- 스프링 시큐리티의 처리 순서

7~10
- 모든 프로세스가 종료되면 Authentication 객체가 SecurityContext에 저장됨
- 따라서 인증된 사용자 정보가 필요한 경우 Authentication 객체를 사용하면 됨

스프링 시큐리티와 JPA 연동

2 JPA 적용하기

〈3〉 Authentication 객체

- 스프링 시큐리티 인증 프로세스에 의해 최종적으로 SecurityContext에 등록되는 Authentication 객체에는 다음과 같은 정보들이 저장됨

2 JPA 적용하기

〈3〉 Authentication 객체

- 스프링 시큐리티 인증 프로세스에 의해 최종적으로 SecurityContext에 등록되는 Authentication 객체에는 다음과 같은 정보들이 저장됨

- Principal(username)
- Credentials(password)
- Authorities(role)

스프링 시큐리티와 JPA 연동

2 JPA 적용하기

④ 엔티티 클래스 작성

- 회원 엔티티는 기존의 Member 엔티티를 재사용함

2 JPA 적용하기

④ 엔티티 클래스 작성

- 권한을 문자열이 아닌 숫자 형태로 저장할 경우는 @Enumerated(EnumType.ORDINAL)을 설정함

```java
package com.mycompany.entity;

import javax.persistence.Entity;

@Data
@Entity
public class Member {
    @Id
    private String username;
    private String password;
    private String name;
    @Enumerated(EnumType.STRING)
    private Role role;
    private boolean enabled;
}
```

```java
package com.mycompany.entity;

public enum Role {
    ROLE_USER, ROLE_MANAGER
}
```

스프링 시큐리티와 JPA 연동

2 JPA 적용하기

〈5〉 Repository 인터페이스 작성

- MEMBER 테이블과 CRUD를 처리할 Repository 인터페이스는 MemberRepository를 재사용함

```
package com.mycompany.persistence;

import org.springframework.data.repository.CrudRepository;

@Repository
public interface MemberRepository extends CrudRepository<Member, String> {
}
```

3 스프링 시큐리티 커스터마이징

〈1〉 UserDetails와 UserDetailsService

- 사용자가 시스템의 특정 리소스에 접근하기 위해서는 반드시 AuthenticationFilter 필터의 인증 프로세스를 통과해야 함

스프링 시큐리티와 JPA 연동

3 스프링 시큐리티 커스터마이징

1 UserDetails와 UserDetailsService

- AuthenticationFilter 필터

> • UserDetailsService를 이용하여 회원 정보를 조회하고 이를 바탕으로 UserDetails 객체를 획득함
> • UserDetails 객체로부터 인증과 인가에 필요한 정보들을 추출하여 사용함

3 스프링 시큐리티 커스터마이징

1 UserDetails와 UserDetailsService

- 스프링 시큐리티가 제공하는 기본 인증 처리를 커스터마이징하기 위한 조건

> • UserDetails 클래스와 UserDetailsService 인터페이스를 직접 구현해야 함

스프링 시큐리티와 JPA 연동

3 스프링 시큐리티 커스터마이징

〈2〉 UserDetails 클래스

- UserDetails를 implements함

3 스프링 시큐리티 커스터마이징

〈2〉 UserDetails 클래스

PrincipalDetails 생성자

- 검색한 Member 객체를 멤버 변수에 할당

```
package com.mycompany.security.config;

import java.util.ArrayList;

public class PrincipalDetails implements UserDetails {
    private static final long serialVersionUID = 1L;
    private Member member;

    public PrincipalDetails(Member member) {
        this.member = member;
    }

    @Override
    public String getPassword() {
        return "{noop}" + member.getPassword();
    }

    @Override
    public String getUsername() {
        return member.getUsername();
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }
```

스프링 시큐리티와 JPA 연동

3 스프링 시큐리티 커스터마이징

〈2〉 UserDetails 클래스

getPassword 메소드

- 검색한 비밀번호 리턴

```
package com.mycompany.security.config;

import java.util.ArrayList;

public class PrincipalDetails implements UserDetails {
    private static final long serialVersionUID = 1L;
    private Member member;

    public PrincipalDetails(Member member) {
        this.member = member;
    }

    @Override
    public String getPassword() {
        return "{noop}" + member.getPassword();
    }

    @Override
    public String getUsername() {
        return member.getUsername();
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }
```

3 스프링 시큐리티 커스터마이징

〈2〉 UserDetails 클래스

getUsername 메소드

- 검색한 회원 아이디 리턴

```
package com.mycompany.security.config;

import java.util.ArrayList;

public class PrincipalDetails implements UserDetails {
    private static final long serialVersionUID = 1L;
    private Member member;

    public PrincipalDetails(Member member) {
        this.member = member;
    }

    @Override
    public String getPassword() {
        return "{noop}" + member.getPassword();
    }

    @Override
    public String getUsername() {
        return member.getUsername();
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }
```

스프링 시큐리티와 JPA 연동

3 스프링 시큐리티 커스터마이징

〈2〉 UserDetails 클래스

isAccountNotExpired메소드

- 계정의 만료 정보 리턴

```
package com.mycompany.security.config;

import java.util.ArrayList;

public class PrincipalDetails implements UserDetails {
    private static final long serialVersionUID = 1L;
    private Member member;

    public PrincipalDetails(Member member) {
        this.member = member;
    }

    @Override
    public String getPassword() {
        return "{noop}" + member.getPassword();
    }

    @Override
    public String getUsername() {
        return member.getUsername();
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
```

3 스프링 시큐리티 커스터마이징

〈2〉 UserDetails 클래스

isAccountNonLocked메소드

- 계정의 잠김 상태 리턴

```
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        Collection<GrantedAuthority> roleList = new ArrayList<>();
        roleList.add(()-> {
            return "ROLE_" + member.getRole();
        });
        return roleList;
    }
}
```

스프링 시큐리티와 JPA 연동

3 스프링 시큐리티 커스터마이징

〈2〉 UserDetails 클래스

isCredentialsNonExpired 메소드

- 비밀번호의 만료 정보 리턴

```
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        Collection<GrantedAuthority> roleList = new ArrayList<>();
        roleList.add(()-> {
            return "ROLE_" + member.getRole();
        });
        return roleList;
    }
}
```

3 스프링 시큐리티 커스터마이징

〈2〉 UserDetails 클래스

isEnabled 메소드

- 계정의 활성화 정보 리턴

```
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        Collection<GrantedAuthority> roleList = new ArrayList<>();
        roleList.add(()-> {
            return "ROLE_" + member.getRole();
        });
        return roleList;
    }
}
```

스프링 시큐리티와 JPA 연동

3 스프링 시큐리티 커스터마이징

〈2〉 UserDetails 클래스

getAuthorities 메소드

- 권한 목록 리턴

```
@Override
public boolean isAccountNonLocked() {
    return true;
}

@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return true;
}

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    Collection<GrantedAuthority> roleList = new ArrayList<>();
    roleList.add(()-> {
        return "ROLE_" + member.getRole();
    });
    return roleList;
}
```

3 스프링 시큐리티 커스터마이징

〈3〉 UserDetailsService 클래스

- 스프링에서 제공하는 UserDetailsService를 구현하는 클래스를 작성함

스프링 시큐리티와 JPA 연동

3 스프링 시큐리티 커스터마이징

〈3〉 UserDetailsService 클래스

- UserDetailsService의 loadUserByUsername 메소드

  - 사용자가 입력한 아이디(username)를 이용하여 회원 정보를 조회함
  - 조회 결과를 이용하여 UserDetails 타입의 객체를 생성하여 리턴함

3 스프링 시큐리티 커스터마이징

〈3〉 UserDetailsService 클래스

- UserDetailsService의 loadUserByUsername 메소드

```java
package com.mycompany.security.config;

import org.springframework.beans.factory.annotation.Autowired;

@Service
public class PrincipalDetailsService implements UserDetailsService {
    @Autowired
    private MemberRepository memberRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        Member principal = memberRepository.findById(username).get();
        return new PrincipalDetails(principal);
    }
}
```

스프링 시큐리티와 JPA 연동

3 스프링 시큐리티 커스터마이징

④ 사용자 정의 UserDetailsService 적용

- 스프링 시큐리티에서 기본 UserDetailsService가 아닌 사용자가 정의한 UserDetailsService를 사용하도록 SecurityConfig 파일을 수정함

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private UserDetailsService userDetailsService;

    @Autowired
    public void authenticate(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService);
    }
```

비밀번호 암호화

1 비밀번호 암호화

⟨1⟩ 비밀번호와 암호화

- 사용자 정보를 이용하여 SecurityUser 객체 생성 시

> • 비밀번호에 대한 암호화를 사용하지 않기 위해 비밀번호 앞에 "{noop}"이라는 접두사를 붙임

1 비밀번호 암호화

⟨1⟩ 비밀번호와 암호화

> • 사용자가 입력한 비밀번호는 반드시 암호화하여 저장해야 함
> • 그렇지 않으면 보안 사고 발생시 심각한 문제를 가져올 수 있음

비밀번호 암호화

1 비밀번호 암호화

〈2〉 PasswordEncoder

- 스프링 시큐리티

> • 패스워드 암호화를 처리할 수 있도록 PasswordEncoder 인터페이스를 구현한 다양한 클래스들을 제공함

1 비밀번호 암호화

〈2〉 PasswordEncoder

- PasswordEncoder 객체를 생성할 때 사용하는 PasswordEncoderFactories 클래스의 createDelegatingPasswordEncoder 메소드

```
@SuppressWarnings("deprecation")
public static PasswordEncoder createDelegatingPasswordEncoder() {
    String encodingId = "bcrypt";
    Map<String, PasswordEncoder> encoders = new HashMap<>();
    encoders.put(encodingId, new BCryptPasswordEncoder());
    encoders.put("ldap", new org.springframework.security.crypto.password.LdapShaPasswordEncoder());
    encoders.put("MD4", new org.springframework.security.crypto.password.Md4PasswordEncoder());
    encoders.put("MD5", new org.springframework.security.crypto.password.MessageDigestPasswordEncoder("MD5"));
    encoders.put("noop", org.springframework.security.crypto.password.NoOpPasswordEncoder.getInstance());
    encoders.put("pbkdf2", new Pbkdf2PasswordEncoder());
    encoders.put("scrypt", new SCryptPasswordEncoder());
    encoders.put("SHA-1", new org.springframework.security.crypto.password.MessageDigestPasswordEncoder("SHA-1"));
    encoders.put("SHA-256",
            new org.springframework.security.crypto.password.MessageDigestPasswordEncoder("SHA-256"));
    encoders.put("sha256", new org.springframework.security.crypto.password.StandardPasswordEncoder());
    encoders.put("argon2", new Argon2PasswordEncoder());
    return new DelegatingPasswordEncoder(encodingId, encoders);
}
```

비밀번호 암호화

1 비밀번호 암호화

〈2〉 PasswordEncoder

- 별도의 설정이 없으면 createDelegatingPasswordEncoder 메소드는 BCryptPasswordEncoder 객체를 리턴함

1 비밀번호 암호화

〈3〉 PasswordEncoder 등록

- 시큐리티 설정 클래스(SecurityConfig)에 PasswordEncoder를 생성하는 메소드를 추가함

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private UserDetailsService userDetailsService;

    @Bean
    public PasswordEncoder passwordEncoder() {
        return PasswordEncoderFactories.createDelegatingPasswordEncoder();
    }
```

비밀번호 암호화

2 비밀번호 암호화 적용

⟨1⟩ 테스트 케이스 작성

- 회원 가입 기능을 테스트하는 테스트 케이스를 작성함

```
package com.mycompany;

import org.junit.jupiter.api.Test;

@SpringBootTest
public class MemberServiceTest {
    @Autowired
    private MemberRepository memberRepository;

    @Autowired
    private PasswordEncoder encoder;

    @Test
    public void testInsert() {
        Member member = new Member();
        member.setUsername("guest");
        member.setPassword(encoder.encode("guest123"));
        member.setName("손님");
        member.setRole(Role.ROLE_USER);
        member.setEnabled(true);
        memberRepository.save(member);
    }
}
```

2 비밀번호 암호화 적용

⟨2⟩ 비밀번호 암호화 확인

- MEMBER 테이블을 조회하여 비밀번호가 암호화 되어 저장된 것을 확인함

```
SELECT * FROM MEMBER;
```

USERNAME	PASSWORD	NAME	ROLE	ENABLED
user	user123	회원	ROLE_USER	TRUE
test	test123	매니저	ROLE_MANAGER	TRUE
guest	{bcrypt}$2a$10$tRy8Gw6CFaX1OfeIPFUaoOHm9NyYMAKmedBuNYQHZKS3N534TaNBG	손님	ROLE_USER	TRUE

(3 행, 2 ms)

비밀번호 암호화

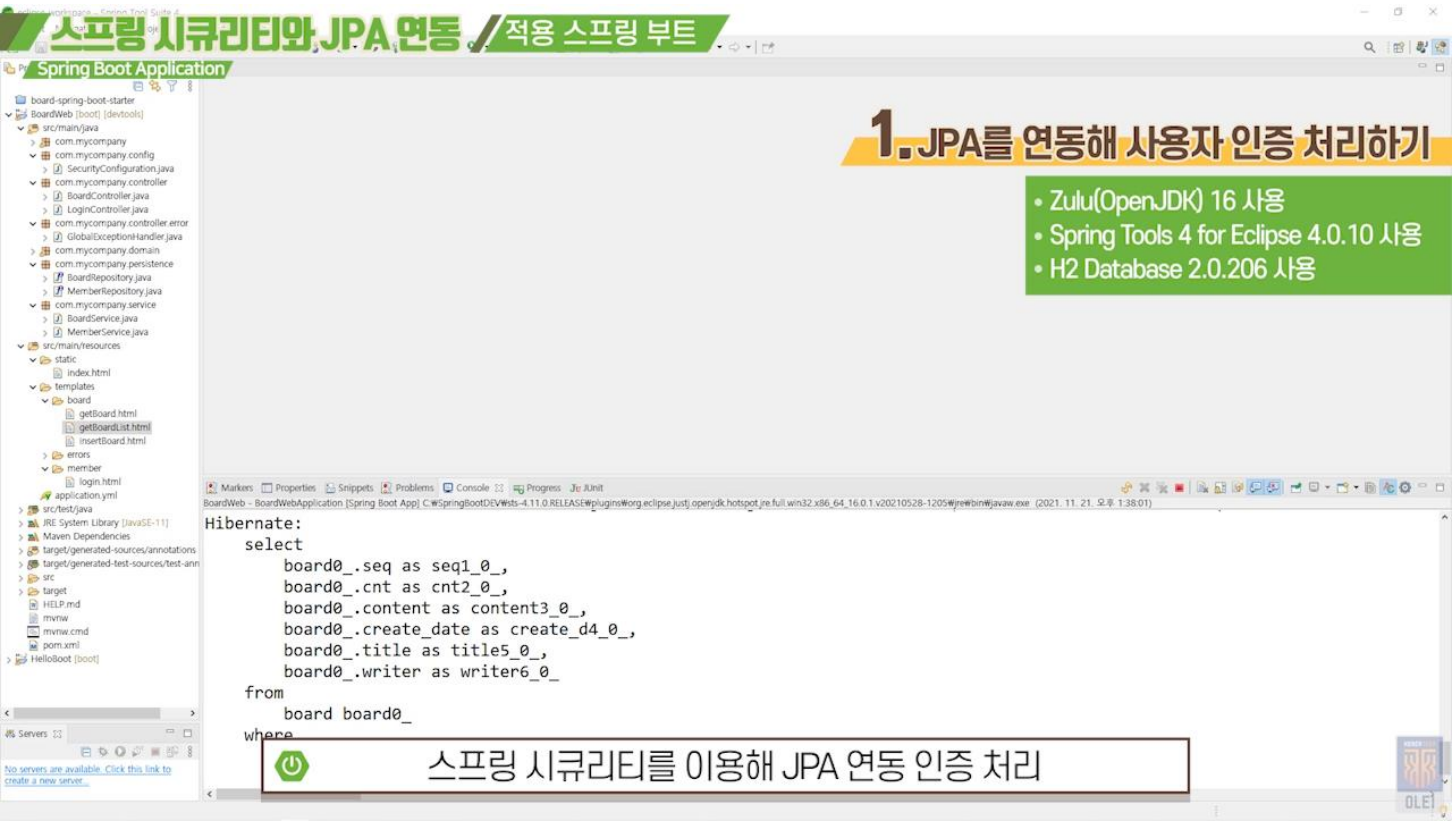
2 비밀번호 암호화 적용

〈3〉 {noop} 제거

- 더 이상 비밀번호를 암호화하지 않기 위해서 PrincipalDetails 클래스에서 사용했던 '{noop}' 접두사를 제거함

```
package com.mycompany.security.config;

import java.util.ArrayList;

public class PrincipalDetails implements UserDetails {
    private static final long serialVersionUID = 1L;
    private Member member;

    public PrincipalDetails(Member member) {
        this.member = member;
    }

    @Override
    public String getPassword() {
        return member.getPassword();
    }
```

적용 스프링 부트

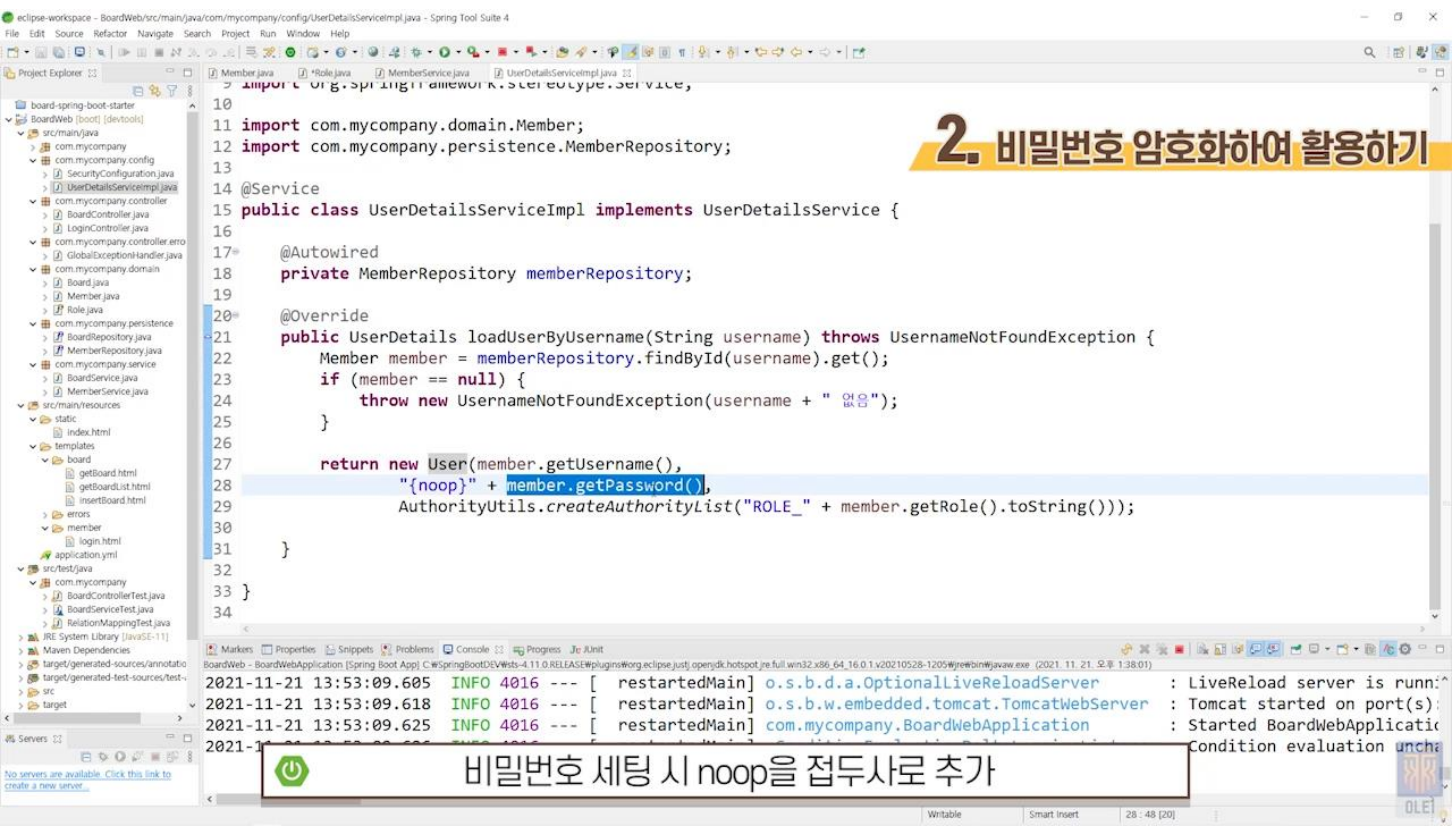


실습단계
스프링 시큐리티를 이용해 JPA 연동 인증 처리
Member 엔티티 이용
문자열로 id, password, name, role 관리
com.mycompany.domain → New → Enum
권한 설정
중요! 인증에 사용하는 엔티티는 ID를 쓰면 안 되며 username을 써야 함
@Enumerated 어노테이션 추가, EnumType 설정
ORDINAL : 숫자 형태로 값을 세팅
STRING : 문자 형태로 값을 세팅
계정 활성화/비활성화 가능
중요! UserDetailsService 인터페이스 구현 클래스 작성
메소드 수정
에러가 발생한 코드를 주석으로 처리
com.mycompany.config → New → Class

적용 스프링 부트

실습단계
Name : UserDetailsServiceImpl
UserDetailsService 작성
인터페이스이기 때문에 추상 메소드를 오버라이딩 해야 함
스프링 시큐리티가 제공하는 객체는 더 이상 뜨지 않음
MemberRepository 타입의 객체 의존성 주입
loadUserByUsername 메소드 오버라이딩
중요! UserType 클래스, 스프링 프레임워크 시큐리티가 제공하는 User, Member 클래스는 Member 엔티티
중요! User 객체를 생성할 때 username을 이용해서 아이디 세팅

적용 스프링 부트



실습단계
비밀번호 세팅 시 noop을 접두사로 추가
권한 목록 세팅
권한 앞에는 ROLE_ 접두사를 추가해야 함
SecurityConfiguration.java 클래스에서 UserDetailsServiceImpl 객체를 @Autowired로 의존성 주입해야 함
커스터마이징한 DetailsService를 사용
전체 저장 후 애플리케이션 다시 실행
확인! H2 console에서 기존 사용하던 member table을 drop 해야 함
새롭게 생성한 table에 맞는 데이터 insert
작성한 insert로 계정 하나를 등록함
브라우저에서 확인