Spring security 사용하기!

1. DB에 계정 관련 테이블 설정하기

```
# db사용을 명시적으로 설정한다.
use testdb;

# 계정 테이블은 계정정보인 user과 권한정보인 authority 테이블로 나눴다.

# user 테이블은 보통 이러한 컬럼을 넣는다고 한다.
CREATE TABLE `user` (
    `username` VARCHAR(20) NULL DEFAULT NULL,
    `password` VARCHAR(500) NULL DEFAULT NULL,
    `name` VARCHAR(20) NULL DEFAULT NULL,
    `isAccountNonExpired` TINYINT(1) NULL DEFAULT NULL,
    `isAccountNonLocked` TINYINT(1) NULL DEFAULT NULL,
    `isCredentialsNonExpired` TINYINT(1) NULL DEFAULT NULL,
    `isEnabled` TINYINT(1) NULL DEFAULT NULL
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB
;

# authority 테이블은 username과 권한정보만을 담고있다.
# 여기서 username은 ID입니다.

CREATE TABLE `authority` (
    `username` VARCHAR(20) NULL DEFAULT NULL,
    `authority_name` VARCHAR(20) NULL DEFAULT NULL
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB
;


# abc라는 user의 권한을 ADMIN과 USER로 추가해보았다.
INSERT INTO `authority` (`username`, `authority_name`) VALUES
    ('abc', 'ADMIN'),
    ('abc', 'USER');

# 계정 정보를 넣는다.
# password가 왜 저렇게 긴지 궁금하다!
# 그 이유는 Spring security가 hash값으로 변환해서 password를 확인하기 때문이다.
# 저 긴 password는 abcd를 hash값으로 바꾼 값이다.

INSERT INTO `user` (`username`, `password`, `name`, `isAccountNonExpired`,
`isAccountNonLocked`, `isCredentialsNonExpired`, `isEnabled`) VALUES
    ('abc', '$2a$10$zNM1N.WnfC1Sq.vkqieCnuEfE3sZ3Hwo6.ytaSBtFTyg33qr2oI2G',
'JKS', 1, 1, 1, 1);

# 잘들어갔는지 확인해보자.
SELECT * from authority;
SELECT * from user;
```

2. DB설정에 맞게 Spring Boot 소스 작성하기

2-1. Domain 생성

```java
public class Member {

    private String username;

    private String password;

    private String name;

    private boolean isAccountNonExpired;

    private boolean isAccountNonLocked;

    private boolean isCredentialsNonExpired;

    private boolean isEnabled;
```

DB의 테이블에 맞는 Domain(java에선 VO객체)을 생성한다.
user 테이블에 맞는 객체를 생성한다. (Authority는 생성하지 않는다.)
Authority는 username을 key값으로 접근만한다.

DB 명과 혼동을 방지하기 위해 Member라는 클래스로 정의했다.
메소드는 getter와 setter만 정의했다. (소스참고)

2-2. Mapper 구현

```java
import java.util.List;

public interface UserMapper {

    public Member readUser(String username);

    public List<String> readAuthority(String username);
}
```

DB접근은 앞서 배운 Mybatis의 방식을 사용한다.
따라서 interface로 Mapper의 내용물을 선언만 해준다.

2-3. mapper 메소드 정의 xml파일 정의

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.demo.security.Mapper.UserMapper">
    <select id="readUser" parameterType="String" resultType="com.example.demo.security.domain.Member">
        SELECT * FROM user WHERE username = #{username}
    </select>
    <select id="readAuthority" parameterType="String" resultType="String">
        SELECT authority_name FROM authority WHERE username = #{username}
    </select>
</mapper>
```

/src/main/resource/mybatis/mapper/ 경로에 새로운 xml파일을 생성한다.

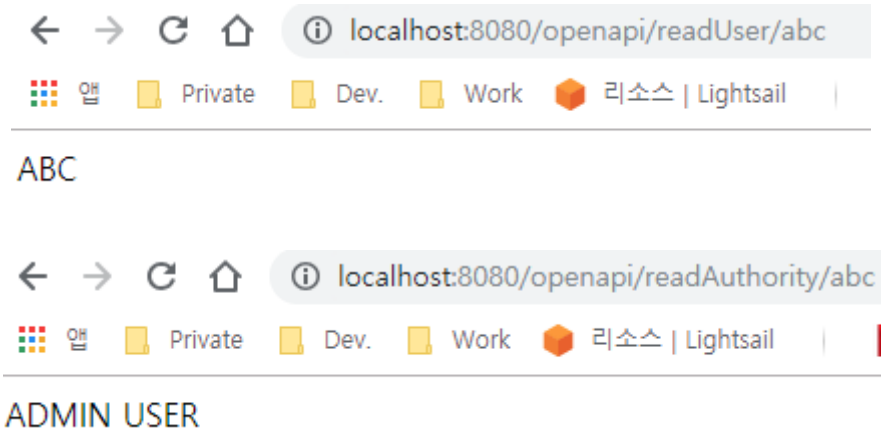
'이미 정의되어있는 xml파일에 다른 namespace를 사용하는 mapper를 저의하면 되는것이 아닌가??'' 하는 의문이 생긴다!!

하지만 안된다!
한 xml파일에는 한 namespace만을 사용한다.

위 캡처에는 readUser와 readAuthority를 읽어오기만 한다.(username을 이용)
이유는 username(즉 ID)는 유일성을 가진다. 다만 password는 다른 유저와 겹칠 수 있다.
따라서 유일성을 가지는 username을 가지고 접근하는 것이다.

2-3 Controller 구현

```java
@Controller
public class UserTestController {

    @Autowired
    UserMapper userMapper;

    @RequestMapping("/openapi/readUser/{username}")
    public @ResponseBody String  openApiReadUser(@PathVariable String username) {
        Member member = userMapper.readUser(username);
        return member.getName();
    }

    @RequestMapping("/openapi/readAuthority/{username}")
    public @ResponseBody String  openApiReadAuthority(@PathVariable String username) {
        List<String> auths = userMapper.readAuthority(username);

        StringBuffer buf = new StringBuffer();
        for(String auth : auths) {
            buf.append(auth);
            buf.append(" ");
        }
        return buf.toString();
    }
}
```



여기까지는 Spring security를 적용하지 않았다.

Mybatis를 사용해 MariaDB에 접근만 했다.

여기까지 결과만 확인해보자.

3. Spring security 적용!

3-1. 의존성(Dependency) 추가

```
<!-- SECURITY -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<!-- SECURITY -->
```


Spring security 사용을 위해 의존성을 추가한다. (너무도 당연!)

3-2. Member class 수정하기!

```
// 외부참조
import org.springframework.security.core.GrantedAuthority;
// 상태변수 추가
private Collection<? extends GrantedAuthority> authorities;
// 이에따른 메소드 추가
public Collection<? extends GrantedAuthority> getAuthorities() {
    return authorities;
}

public void setAuthorities(Collection<? extends GrantedAuthority> authorities) {
    this.authorities = authorities;
}
```

Security의 User를 구현하는 과정에서 Authority 정보가 필요하고 이를 위한 추가입니다.

위 내용물들을 Member.java에 적절한 위치에 추가합니다.

3-3. org.springframework.security.core.userdetails.User 를 상속받는 Domain 생성

```java
package com.example.demo.security.domain;

import org.springframework.security.core.userdetails.User;

public class SecurityMember extends User{

    private static final long serialVersionUID = 1L;

    private String ip;

    public SecurityMember(Member member) {
        super(member.getUsername(), member.getPassword(), member.getAuthorities());
    }

    public String getIp() {
        return ip;
    }

    public void setIp(String ip) {
        this.ip = ip;
    }

}
```

앞서 작성한 Member class는 그저 DB형태에 맞게 구현한 객체이고 Spring에서 인정하는 User의 형태는 별도로 존재한다.

이것은 상속받은 User객체입니다.
앞서 만든 Member class와 매핑해주어 사용해봅니다.

3-4. Service 구현하기

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

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
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 com.example.demo.security.domain.Member;
import com.example.demo.security.domain.SecurityMember;
import com.example.demo.security.Mapper.UserMapper;

@Service
public class CustomUserDetailsService implements UserDetailsService{

    private static final String ROLE_PREFIX = "ROLE_";

    @Autowired
    UserMapper userMapper;

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

        Member member = userMapper.readUser(username);
        if(member != null) {
            member.setAuthorities(makeGrantedAuthority(userMapper.readAuthority(username)));
        }
        return new SecurityMember(member);
    }

    private static List<GrantedAuthority> makeGrantedAuthority(List<String> roles){
        List<GrantedAuthority> list = new ArrayList<>();
        //SimpleGrantedAuthority객체 생성 시 ROLE를 붙여 Spring이 이해할 수 있도록 표현
        roles.forEach(role -> list.add(new SimpleGrantedAuthority(ROLE_PREFIX + role)));
        return list;
    }

}
```

UserDetailsService를 상속받는 구현체를 만듭니다.

중요!!! -> loadUserByUsername 함수
username을 파라미터로 해당 사용자의 존재여부를 증명하는 부분입니다.

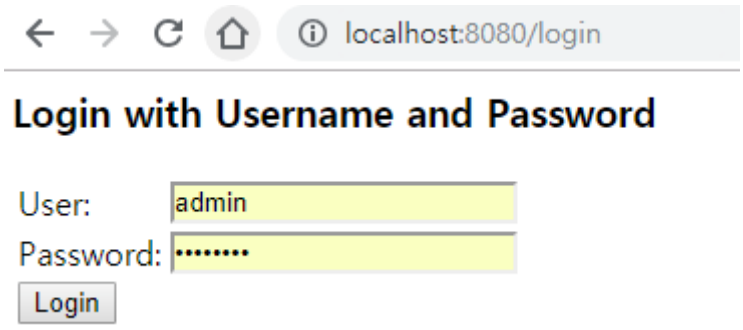
3-5. WebSecurityConfigurerAdapter를 상속받는 Adapter 구현

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.builders.WebSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

import com.example.demo.security.service.CustomUserDetailsService;

@EnableWebSecurity
public class CustomWebSecurityConfigurerAdapter extends WebSecurityConfigurerAdapter {

    @Autowired
    CustomUserDetailsService customUserDetailsService;

    @Bean
    public PasswordEncoder passwordEncoder() {
      return new BCryptPasswordEncoder();
    }

    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring()
        .antMatchers("/openapi/**");
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable().authorizeRequests().anyRequest().authenticated().and().formLogin();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(customUserDetailsService).passwordEncoder(passwordEncoder());
    }

}
```

이부분은 암호화방식인 password encoder를 정의합니다.
또한 각종 요청 ignore처리, login page에 대한 처리가 정의됩니다.

우선은 간단히 작성했지만 더 작성할 일이 있을겁니다.

3-5. Spring security 적용 확인하기!



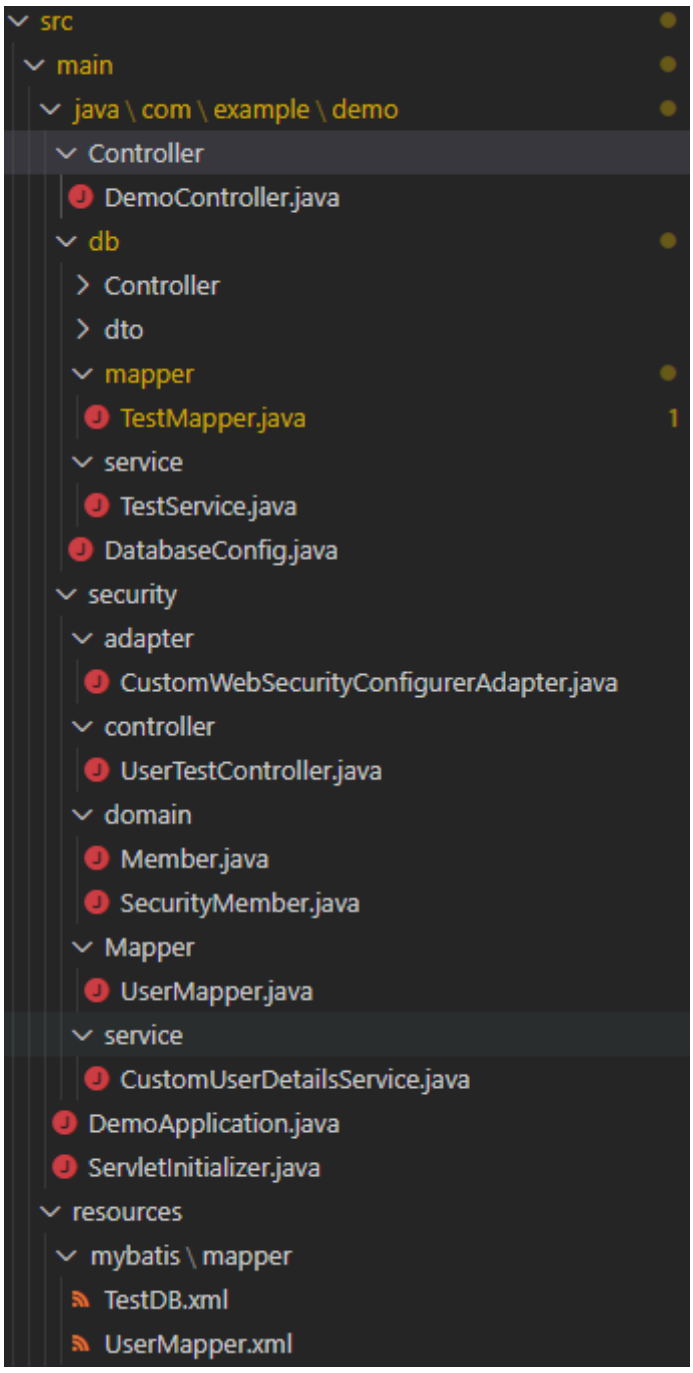
현재 Controller에 정의된 url 어느곳에 접근해도 로그인을 하지 않았다면 login UI로 이동하게 됩니다.

현재 DB에 저장해놓은 username (abc), password (해쉬화된 abcd)으로 접속확인을 위해 abc, abcd를 입력합니다.

그럼 정상동작!!!

4. 필요한 지식들

4-1. 내가 작성한 소스들 경로 사진!



이렇게 정리해봤습니다.

4-2. Mybatis에서 mapper객체를 bean으로 정의해야한다는데요??

```java
@Configuration
@MapperScan(basePackages="com.example.demo.db.mapper")
@MapperScan(basePackages="com.example.demo.security.Mapper")
@EnableTransactionManagement
public class DatabaseConfig {


  // 원래라면 root-context.xml에서 정의했을 부분이다.
  // application.properties에서 정의한 dataSource의 정보를 이용한다.
  // mybatis 기본이용법이니 참고자료는 구글링하시길

    @Bean
    public SqlSessionFactory sqlSessionFactory(DataSource dataSource) throws Exception {
        final SqlSessionFactoryBean sessionFactory = new SqlSessionFactoryBean();
        sessionFactory.setDataSource(dataSource);
        PathMatchingResourcePatternResolver resolver = new PathMatchingResourcePatternResolver();
        sessionFactory.setMapperLocations(resolver.getResources("classpath:mybatis/mapper/*.xml"));
        return sessionFactory.getObject();
    }

    @Bean
    public SqlSessionTemplate sqlSessionTemplate(SqlSessionFactory sqlSessionFactory) throws Exception {
      final SqlSessionTemplate sqlSessionTemplate = new SqlSessionTemplate(sqlSessionFactory);
      return sqlSessionTemplate;
    }


}
```

상단 @MapperScan 어노테이션을 보면 security.Mapper 객체를 스캔한 것을 볼 수 있습니다.

그럼 Autowired 정상작동합니다.

Made By 장경석