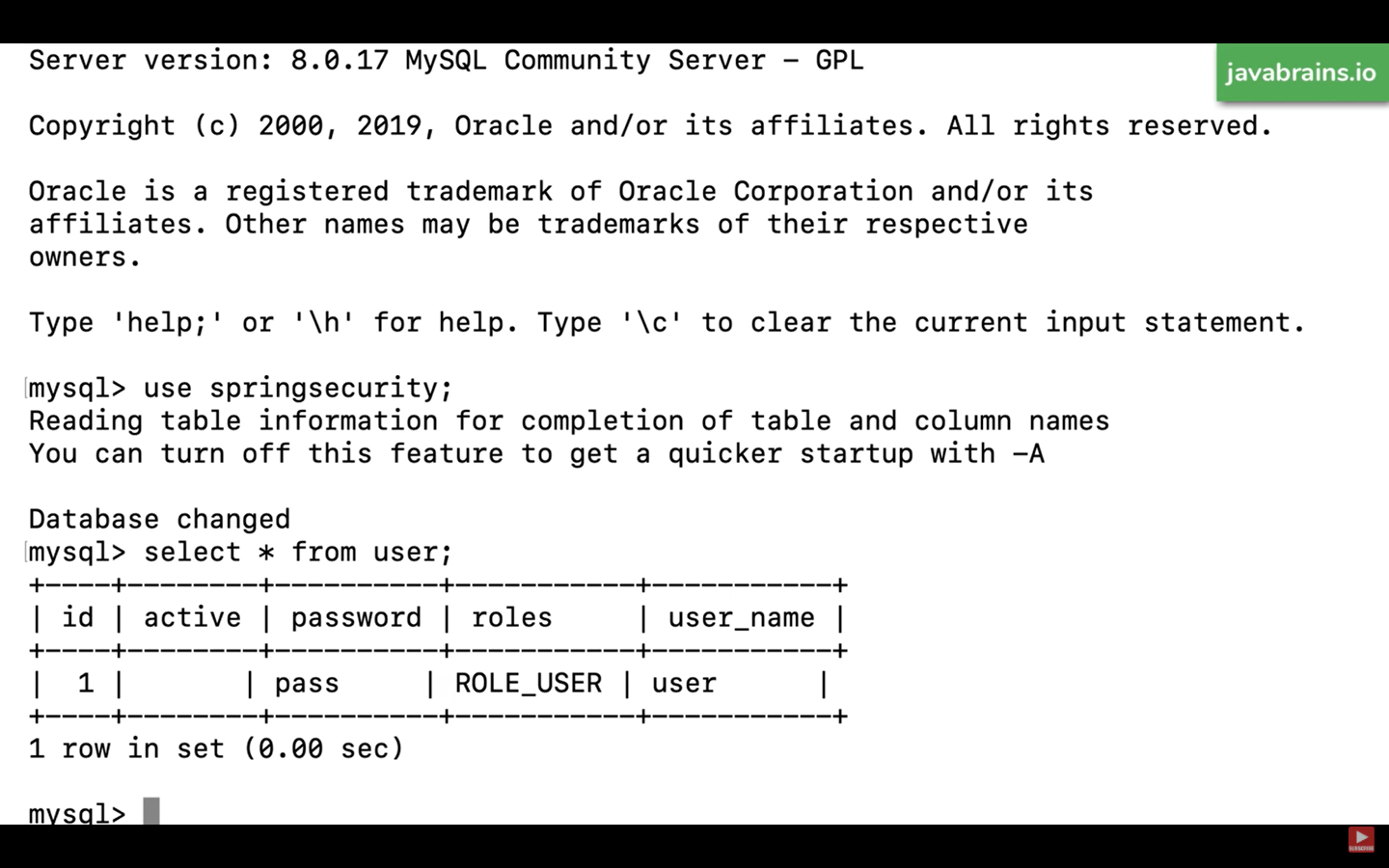
**Spring Security**

**Spring Boot + Spring Security with JPA authentication and MySQL**

Create user table in mysql database with columns : id , active , password,roles,user\_name

Insert data : 1,’’,’pass’,’ROLE\_USER’,user



Create new spring boot project from start.spring.io

Group id : io.javabrains

Artifact id : spring-security-jpa

Add Dependencies :

* Spring Web Starter
* Spring Security
* Spring Data Jpa
* MySQL Driver

Unzip and Import project in STS

Create new class name HomeResource (Controller) and Add Annotation @RestController

Create three api’s i.e. home , user , admin with

Home to mapped to root url – publicly accessible. User will have authenticated accessible by user and admin. Admin api to accessible by admin only.

**HomeResource.java**

import org.springframework.web.bind.annotation.RestController;

@RestController

public class HomeResource {

@GetMapping(“/”)

public String home(){

return (“<h1>Welcome</h1>”);

}

@GetMapping(“/user”)

public String user(){

return (“<h1>Welcome User</h1>”);

}

@GetMapping(“/admin”)

public String admin(){ return (“<h1>Welcome Admin</h1>”); }

}

Create new class named SecurityConfiguration extends WebSecurityConfigurerAdapter

Override configure(AuthenticationManagerBuilder auth) method to configure authentication

Also Override configure(HttpSecurity http) to setup authorization

Add code in configure(HttpSecurity http) method :

http.authorizeRequests()

.antMatchers(“/admin”).hasRole(“ADMIN”)

.antMatchers(“/user”).hasAnyRole(“ADMIN”,”USER”)

.antMatchers(“/”).permitAll()

.and().formLogin();

To setup jpa authentication :

Declare UserDetailsService userDetailsService and add Autowired Annotation

Add code in configure(AuthenticationManagerBuilder auth) method :

auth.userDetailsService(userDetailsService);

Create new class named MyUserDetailsService implements UserDetailsService

Implement loadUserByUsername method // if we add some logic here which returns user Details. it will work!

Create new class named MyUserDetails implements UserDetails

Implement all bunch of methods i.e. getAuthorities , getPassword , getUsername,etc . Spring security is going to take this method value that the user details service returns and use for authentication.

Create constructor in MyUserDetails which takes an argument String username and set username to member variable and also declare private String userName.

Create empty constructor as well and return userName in getUsername method and “pass” in getPassword and return true in isAccountNonExpired, return true in isAccountNonLocked, return true in isCredentialsNonExpired, return true in isEnabled. Hardcode everything except for username.

Return Arrays.asList(new SimpleGrantedAuthority(“ROLE\_USER”));

Go to MyUserDetailsService and return new MyUserDetails(s) in loadUserByUsername method

Add @Service annotation on MyUserDetailService class

Go to SecurityConfiguration and Add @EnableWebSecurity annotation on it.

Add getPasswordEncoder method in SecurityConfiguration

SecurityConfiguration.java

|  |  |
| --- | --- |
|  |  |

@EnableWebSecurity

public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

@Autowired

UserDetailsService userDetailsService;

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception{

auth.userDetailsService(userDetailsService);

}

@Override

protected void configure(HttpSecurity http) throws Exception{

http.authorizeRequests()

.antMatchers(“/admin”).hasRole(“ADMIN”)

.antMatchers(“/user”).hasAnyRole(“ADMIN”,”USER”)

.antMatchers(“/”).permitAll()

.and().formLogin();

}

@Bean

public PasswordEncoder getPasswordEncoder(){ return ~~NoOpPasswordEncoder~~.getInstance(); }

}

MyUserDetailsService.java

@Service

public class MyUserDetailsService implements UserDetailsService {

@Override

public UserDetails loadUserByUsername(String s) throws UsernameNotFoundException{

return new MyUserDetails(s);

}

}

MyUserDetails.java

public class MyUserDetails implements UserDetails {

private String userName;

public MyUserDetails(String username){

this.userName = username;

}

public MyUserDetails(){

}

@Override

public Collection<? extends GrantedAuthority> getAuthorities(){

return Arrays.asList(new SimpleGrantedAuthority(“ROLE\_USER”));

}

@Override

public String getPassword(){

return “pass”;

}

@Override

public String getUsername(){

return userName;

}

@Override

public boolean isAccountNonExpired(){

return true;

}

@Override

public boolean isAccountNonLocked(){

return true;

}

@Override

public boolean isCredentialsNonExpired(){

return true;

}

@Override

public boolean isEnabled(){

return true;

}

}

Run this application. Go to localhost:8080 -> it will response welcome (publicly accessible)

Go to localhost:8080/user , it will prompt /login.. enter any username and pass in password because we have hardcoded it. It will show Welcome User

We will setup JPA and user will be from database.

Create Entity class named User in package models

Move MyUserDetails in package models

Mark @Entity annotation and also @Table(name=”User”)

Declare int id,String userName, password , Boolean active , String roles variables

**User.java**

@Entity

@Table(name = “User”)

public class User{

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private int id;

private String userName;

private String passWord;

private boolean active;

private String roles;

Generate getters & setters

}

Create new interface named UserRepository extends JpaRepository<User,Integer>

Declare Optional<User> findByUserName(String userName)

**UserRepository.java**

public interface UserRepository extends JpaRepository<User, Integer>{

Optional<User> findByUserName(String userName) }

**Go to MyUserDetails.java**

Declare variables :

private String password;

private boolean active;

private List<GrantedAuthority> authorities;

Update constructor :

public MyUserDetails(User user){

this.userName = user.getUserName();

this.password = user.getPassword();

this.active = user.isActive();

this.authorities = Arrays.stream(user.getRoles().split(“,”))

.map(SimpleGrantedAuthority::new)

.collect(Collectors.toList());

}

Remove all other methods. Just keep constructor and declared variables & Implement methods again which UserDetails interface needs

Return authorities in getAuthorities , password in getPassword, username in getUsername method. Return true in isAccountNonExpired , Return true in isAccountNonLocked , Return true in isCredentialsNonExpired , Return active in isEnabled

**Go to MyUserDetailsService**

Declare UserRepository and autowired it

Update loadUserByUsername method

@Service

public class MyUserDetailsService implements UserDetailsService {

@Autowired

UserRepository userRepository;

@Override

public UserDetails loadUserByUsername(String s) throws UsernameNotFoundException{

Optional<User> user = userRepository.findByUserName(username);

user.orElseThrow(() -> new UsernameNotFoundException(“Not found: “+ userName));

return user.map(MyUserDetails::new).get();

// I have gotten the user instance from the database from the repository convert it into a my user details instance and I'm returning this

}

}

Go to SpringSecurityJpaApplication (Main Application)

Add annotation @EnableJpaRepositories(basePackageClasses = UserRepository.class)

Go to application.properties and add

spring.datasource.url=jdbc:mysql://localhost:3306/springsecurity

spring.datasource.username = root

spring.datasource.password = password

spring.jpa.hibernate.ddl.auto = update

spring.jpa.naming-strategy= org.hibernate.cfg.ImprovedNamingStrategy

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySql5Dialect

Relaunch Server. Go to localhost:8080/user and Enter user in username and 1234 (database password) in password and we will be login.

**Spring Boot + Spring Security + JWT**

Create new spring boot project from start.spring.io

Group id : io.javabrains

Artifact id : spring-security-jwt

Add Dependencies :

* Spring Web
* Spring Security

Generate & Unzip and Import project in STS

Create new class named HelloResource

Add @Controller annotation

Add new RequestMapping – ‘/hello’ and return “Hello World”

Create new class named SecurityConfigurer extends WebSecurityConfigurerAdapter

Overide method configure(AuthenticationManagerBuilder auth) and add :

auth.userDetailsService(myUserDetailsService);

Declare private MyUserDetailsService myUserDetailsService and autowired it.

Create new class named MyUserDetailsService implements UserDetailsService

Override method loadUserByUsername(String username) and return newUser(“foo”,”foo”,new ArrayList<>()); and Add Annotation service

Go to SecurityConfigurer and add annotation @EnableWebSecurity and add new method i.e. public PasswordEncoder passwordEncoder & add annotation @Bean and return ~~NoOpPassword~~.getInstance();

**HelloResource.java**

@RestController

public class HelloResource {

@RequestMapping( “/hello” )

public String hello(){

return “Hello World”;

}

}

**MyUserDetailsService.java**

@Service

public class MyUserDetailsService implements UserDetailsService {

@Override

public UserDetails loadUserByUsername(String userName) throws UsernameNotFoundException {

return new User(“foo”,”foo”,new ArrayList<>());

}

}

**SecurityConfigurer.java**

@EnableWebSecurity

public class SecurityConfigurer extends WebSecurityConfigurerAdapter {

@Autowired

private MyUserDetailsService myUserDetailsService;

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(myUserDetailsService);

}

@Bean

public PasswordEncoder passwordEncoder(){

return ~~NoOpPasswordEncoder~~.getInstance();

}

}

Launch application and go to localhost:8080/hello , it will prompt login ,Enter foo,foo to login.

**Implementing JWT**

Add Dependencies :

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

<dependency>

<groupId>javax.xml.bind</groupId>

<artifactId>jaxb-api</artifactId>

<version>2.3.0</version>

</dependency>

Create class named JwtUtil in util package

Copy JwtUtil code from github and paste it.

Imp method : generateToken , validateToken

Create class named AuthenticationRequest in models package

Declare username , password & create getters & setters & constructor using fields & blank constructor

Create class AuthenticationResponse and declare private final String jwt;

Create getters & setters & constructor using fields

Go to HelloResource and Create /authenticate RequestMapping

@RequestMapping(value=”/authenticate”,method=RequestMethod.POST)

public ResponseEntity<?> createAuthenticationToken(@RequestBody AuthenticationRequest authenticationRequest) throws Exception{

try{

authenticateManager.authenticate(

new UsernamePasswordAuthenticationToken( authenticationRequest.getUsername(), authenticationRequest.getPassword())

);

} catch (BadCredentialsException e){

throw new Exception(“Incorrect username or password”, e);

}

final UserDetails userDetails = userDetailsService. loadUserByUsername(authenticationRequest.getUsername());

final String jwt = jwtTokenUtil.generateToken(userDetails);

return ResponseEntity.ok(new AuthenticationRespone(jwt));

}

Declare private AuthenticationManager authenticateManager and autowired it to authenticate.

Declare JwtUtil jwtTokenUtil and autowired it.

To allow authenticate without jwt

Go to **SecurityConfigurer.java**

Override method

Configure(HttpSecurity http)

@Override

protected void configure(HttpSecurity http) throws Exception{

http.csrf().disable()

.authorizeRequests().antMatchers(“/authenticate”).permitAll().

anyRequest().authenticated(); //keep others authenticated

}

We might have a issue to work with AuthenticationManager because of new version. To fix that Go to SecurityConfigurer.java and override method

@Override

@Bean

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

Launch Application and Go to localhost:8080/authenticate in postman – POST

Go to Body and add : { “username”: “foo”, “password”:”foo”}

Response : { “jwt”: “eytrgdfgdfgdssd” }

Copy jwt and hit api : localhost:8080/hello -- GET

Headers:

Content-Type application/json

Authorization Bearer eytrgdfgdfgdssd

Response : Access Denied

It will give forbidden because we have not create logic to validate token

We will create filter (there are bunch of filters)

Create class named JwtRequestFilter extends OncePerRequestFilter

We will examine incoming request for jwt header and we will check if it is valid

**JwtRequestFilter.java**

@Component

public class JwtRequestFilter extends OncePerRequestFilter{

@Autowired

private MyUserDetailsService userDetailsService;

@Autowired

private JwtUtil jwtUtil;

@Override

protected void doFilterInternal(HttpServletRequest request , HttpServletResponse response , FilterChain chain) throws ServletException , IOException{

final String authorizationHeader = request.getHeader(“Authorization”);

String username = null;

String jwt =null;

If (authorizationHeader != null && authorizationHeader.startsWith(“Bearer “)){

jwt = authorizationHeader.substring(7);

username = jwtUtil.extractUsername(jwt);

}

If(username != null && SecurityContextHolder.getContext().getAuthentication() == null){

UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);

If(jwtUtil.validateToken(jwt, userDetails)){

UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(userDetails , null , userDetails.getAuthorities());

usernamePasswordAuthenticationToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication (usernamePasswordAuthenticationToken);

}

}

chain.doFilter(request , response);

}

}

Go to Securityconfigurer.java and then tell it to use that filter change

To interject it

Update configure(HttpSecurity http) and not to create session

Declare

@Autowired

private JwtRequestFilter jwtRequestFilter;

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests().antMatchers(“/authenticate”).permitAll()

.anyRequest().authenticated().and().sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.STATELESS);

http.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);

}

Run application & Generate jwt token with authenticate api

raw: { “username”: “foo” , “password”: “foo” }

Copy jwt and go to hello api and hit first without jwt..you will see access denied

Add in Headers:

Content-Type application/json

Authorization Bearer eyfsdfsdhdsfks

You will see response : Hello World

Connect with live user jpa by changing MyUserDetailsService loadUserByUsername method

<https://github.com/koushikkothagal/spring-security-jpa>

<https://github.com/koushikkothagal/spring-security-jwt>