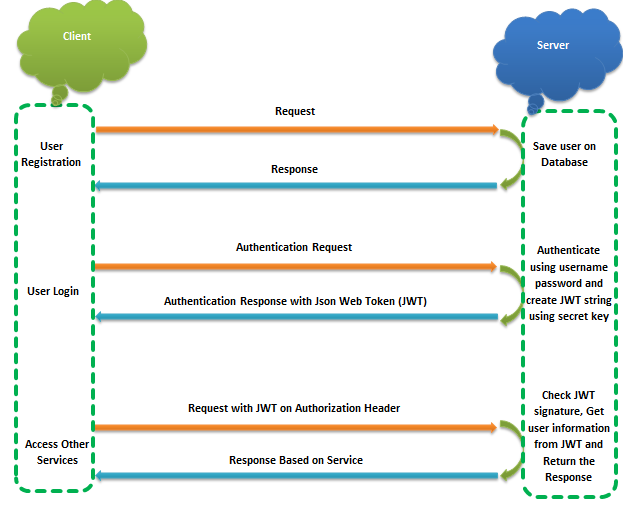
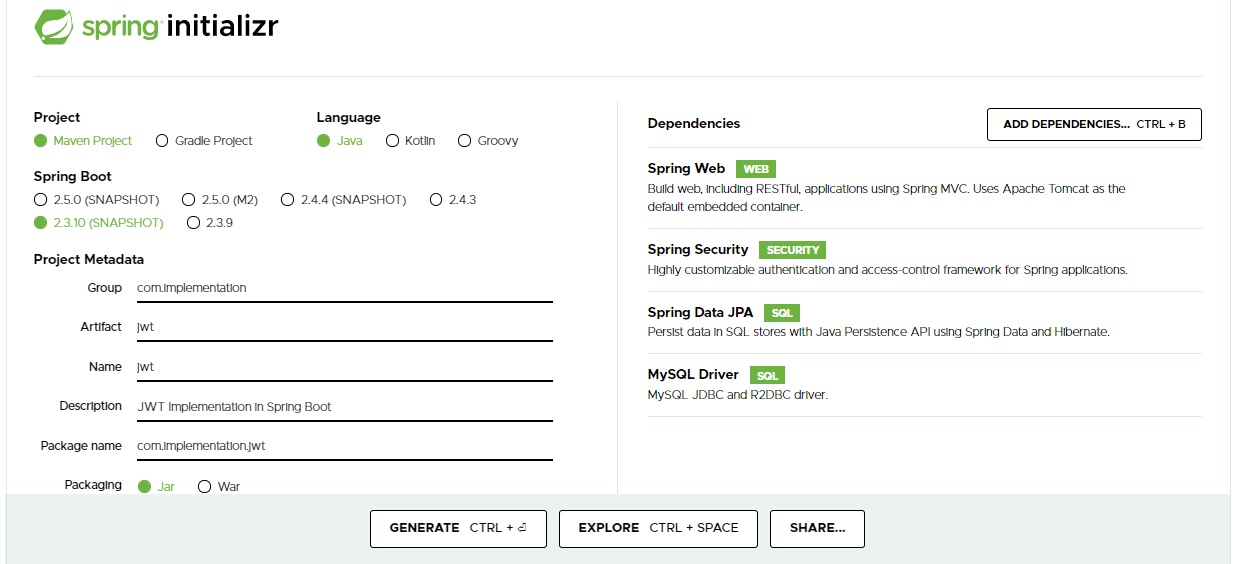
# Spring Boot Token and Role based Authentication with Spring Security, JWT and JPA



This is the diagram which we will implement in this article. We will prepare a spring boot application with role based JWT authentication. JWT authentication is very simple, it follows some basic steps:

* Client completes the registration process from an identity provider.
* Then client will send a request with valid credential to the identity provider. Identity provider verifies the credentials and if it is ok then it provide an encrypted token (JWT).
* Client stores the token (JWT) for a specified period of time depending on the expiration time set by the identity provider.
* Then clients send the stored token (JWT) with the authorization header for every request to access the provider services.
* For each request, the service provider takes the token (JWT) from the Authorization header and decrypts it. Then validates the signature and extracts the user data, check the permissions and if everything is OK then it accept the request. Here identity provider and service provider should have an agreement on secret key to decrypt the token.

Now let’s start the implementation from spring initializer (<https://start.spring.io/>). Follow the below screenshot and click generate button.



After importing the project, you might face some error related with maven dependency. To resolve this error, replace the properties tag with below code in pom.xml file:

<properties>

<java.version>1.8</java.version>

<maven-jar-plugin.version>3.1.1</maven-jar-plugin.version>

</properties>

Now include the following dependency in **pom.xml** file for JWT:

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

After completing the above step, select the project and right click on it and go to Maven > Update project.. from eclipse menu option.

Before writing any code we will configure the data source for mysql database and JWT configuration in **Application.properties** file:

jwt.signing.key.secret=mySecret

jwt.token.expiration.in.seconds=1800

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

spring.datasource.username=username

spring.datasource.password =password

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

**Database table Creation:**

**CREATE TABLE** users (

id int,

username varchar(56),

password varchar(256),

**PRIMARY KEY** (id)

);

**CREATE TABLE** user\_roles (

userid int,

roleid int,

**PRIMARY KEY** (userid,roleid)

);

**CREATE TABLE** roles (

id int,

name varchar(56),

**PRIMARY KEY** (id)

);

**INSERT INTO** roles(id,name) **VALUES** (1,'ROLE\_USER');

**INSERT INTO** roles(id,name) **VALUES** (3,'ROLE\_ADMIN');

Table **users** willstorethe registered user information, table **roles** will store the roles entry and **user\_roles** table will store the user and role id as a foreign key.

**Create entity class:**

Now we will create entity class to persist the data for **users**, **roles** and **user\_roles** table into the package **com.implementation.jwt.model**

Role.java

**package** com.implementation.jwt.model;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.EnumType;

**import** javax.persistence.Enumerated;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name = "roles")

**public** **class** Role {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Integer id;

@Enumerated(EnumType.***STRING***)

@Column(length = 20)

**private** RoleEnum name;

**public** Role() {

}

**public** Role(RoleEnum name) {

**this**.name = name;

}

**public** Integer getId() {

**return** id;

}

**public** **void** setId(Integer id) {

**this**.id = id;

}

**public** RoleEnum getName() {

**return** name;

}

**public** **void** setName(RoleEnum name) {

**this**.name = name;

}

}

For users and user\_roles create User.java

**package** com.implementation.jwt.model;

**import** java.util.HashSet;

**import** java.util.Set;

**import** javax.persistence.Entity;

**import** javax.persistence.FetchType;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.JoinColumn;

**import** javax.persistence.JoinTable;

**import** javax.persistence.ManyToMany;

**import** javax.persistence.Table;

**import** javax.persistence.UniqueConstraint;

@Entity

@Table(name = "users", uniqueConstraints = { @UniqueConstraint(columnNames = "username") })

**public** **class** User {

@Id

@GeneratedValue(strategy = GenerationType.***SEQUENCE***)

**private** Long id;

**private** String username;

**private** String password;

@ManyToMany(fetch = FetchType.***LAZY***)

@JoinTable( name = "user\_roles", joinColumns = @JoinColumn(name = "userid"), inverseJoinColumns = @JoinColumn(name = "roleid"))

**private** Set<Role> roles = **new** HashSet<>();

**public** User() {

}

**public** User(String username, String password) {

**this**.username = username;

**this**.password = password;

}

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** Set<Role> getRoles() {

**return** roles;

}

**public** **void** setRoles(Set<Role> roles) {

**this**.roles = roles;

}

}

Now we need to know our purpose because it will help us to generate the request and response data model. We will create below API’s:

1. For user registration:

URL: <http://localhost:8080/api/authenticate/reg>

Method:POST

Conten-Type: application/json

Body sample:

{

"username":"Sara",

"password":"12345",

"role":["admin"]

}

Response sample: { "message": " Registration successfully! " }

**Note:** For this article we have two roles: admin and user

1. For user login:

URL: <http://localhost:8080/api/authenticate/login>

Method:POST

Conten-Type: application/json

Body sample:

{

"username":"Sara",

"password":"12345"

}

Response sample:

{

"id": 19,

"username": "Sara",

"roles": [

"ROLE\_ADMIN"

],

"tokenType": "Bearer",

"accessToken": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJTYXJhIiwiaWF0IjoxNjE1NjM0NjU3LCJleHAiOjE2MTU2MzY0NTd9.Tz8zS4qRziyZOLi01xRjDLW28yljx7JPgU9qhboSrvJ\_4dtaXf9i5SJtJDcZIoxgWKFCcExyCieLAlAPAhNrDA"

}

1. To check the role and token based accessibility we create some GET API:
   1. <http://localhost:8080/api/resource/content>
   2. <http://localhost:8080/api/resource/user>
   3. <http://localhost:8080/api/resource/admin>

Now according to the API requirements we will create some data model for request and response inside the package **com.implementation.jwt.model**

RegistrationRequest.java

**package** com.implementation.jwt.model;

**import** java.util.Set;

**public** **class** RegistrationRequest {

**private** String username;

**private** String password;

**private** Set<String> role;

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** Set<String> getRole() {

**return** role;

}

**public** **void** setRole(Set<String> role) {

**this**.role = role;

}

}

MessageResponse.java

**package** com.implementation.jwt.model;

**public** **class** MessageResponse {

**private** String message ;

**public** MessageResponse(String message) {

**this**.message = message;

}

**public** String getMessage() {

**return** message;

}

**public** **void** setMessage(String message) {

**this**.message = message;

}

}

RoleEnum.java

**package** com.implementation.jwt.model;

**public** **enum** RoleEnum {

***ROLE\_USER***,

***ROLE\_MODERATOR***,

***ROLE\_ADMIN***

}

JwtLoginRequest.java

**package** com.implementation.jwt.model;

**public** **class** JwtLoginRequest {

**private** String username;

**private** String password;

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

}

JwtResponse.java

**package** com.implementation.jwt.model;

**import** java.util.List;

**public** **class** JwtResponse {

**private** String token;

**private** String type = "Bearer";

**private** Long id;

**private** String username;

**private** List<String> roles;

**public** JwtResponse(String accessToken, Long id, String username, List<String> roles) {

**this**.token = accessToken;

**this**.id = id;

**this**.username = username;

**this**.roles = roles;

}

**public** String getAccessToken() {

**return** token;

}

**public** **void** setAccessToken(String accessToken) {

**this**.token = accessToken;

}

**public** String getTokenType() {

**return** type;

}

**public** **void** setTokenType(String tokenType) {

**this**.type = tokenType;

}

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** List<String> getRoles() {

**return** roles;

}

}

**Note:** Don’t panicking to see the compilation error. Just create the class file as we instructed and ignore the compile error because when we will be created all java class then the compilation dependency will resolve automatically. We are following this straight forward process because it will be very difficult to describe the process if we change the context from one partially completed class to another class.

**Create Repository:**

Now we will create two repository classes to persist the user and role data and place into the newly created package com.implementation.jwt.service:

UserDetailsImpl.java

**package** com.implementation.jwt.service;

**import** java.util.Collection;

**import** java.util.List;

**import** java.util.Objects;

**import** java.util.stream.Collectors;

**import** org.springframework.security.core.GrantedAuthority;

**import** org.springframework.security.core.authority.SimpleGrantedAuthority;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** com.fasterxml.jackson.annotation.JsonIgnore;

**import** com.implementation.jwt.model.User;

**public** **class** UserDetailsImpl **implements** UserDetails {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** Long id;

**private** String username;

@JsonIgnore

**private** String password;

**private** Collection<? **extends** GrantedAuthority> authorities;

**public** UserDetailsImpl(Long id, String username,String password,

Collection<? **extends** GrantedAuthority> authorities) {

**this**.id = id;

**this**.username = username;

**this**.password = password;

**this**.authorities = authorities;

}

**public** **static** UserDetailsImpl build(User user) {

List<GrantedAuthority> authorities = user.getRoles().stream()

.map(role -> **new** SimpleGrantedAuthority(role.getName().name()))

.collect(Collectors.*toList*());

**return** **new** UserDetailsImpl(

user.getId(),

user.getUsername(),

user.getPassword(),

authorities);

}

@Override

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

**return** authorities;

}

**public** Long getId() {

**return** id;

}

@Override

**public** String getPassword() {

**return** password;

}

@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**;

}

@Override

**public** **boolean** equals(Object o) {

**if** (**this** == o)

**return** **true**;

**if** (o == **null** || getClass() != o.getClass())

**return** **false**;

UserDetailsImpl user = (UserDetailsImpl) o;

**return** Objects.*equals*(id, user.id);

}

}

UserDetailsServiceImpl.java

**package** com.implementation.jwt.service;

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

**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** org.springframework.transaction.annotation.Transactional;

**import** com.implementation.jwt.model.User;

**import** com.implementation.jwt.repository.UserRepository;

@Service

**public** **class** UserDetailsServiceImpl **implements** UserDetailsService {

@Autowired

UserRepository userRepository;

@Override

@Transactional

**public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {

User user = userRepository.findByUsername(username)

.orElseThrow(() -> **new** UsernameNotFoundException("User Not Found with username: " + username));

**return** UserDetailsImpl.*build*(user);

}

}

**Configure Spring Security for JWT:**

Configure Spring Security for JWT create another package com.implementation.jwt.security. To expose REST POST API with mapping /api/authenticate/\*\* using which user will get a valid JSON Web Token and then, allow the user access to the API /api/resource/\*\* only if it has a valid token.

WebSecurityConfig.java

**package** com.implementation.jwt.security;

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

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.security.authentication.AuthenticationManager;

**import** org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

**import** org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

**import** org.springframework.security.config.annotation.web.builders.HttpSecurity;

**import** org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

**import** org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

**import** org.springframework.security.config.http.SessionCreationPolicy;

**import** org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

**import** org.springframework.security.crypto.password.PasswordEncoder;

**import** org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

**import** com.implementation.jwt.service.UserDetailsServiceImpl;

@Configuration

@EnableWebSecurity

@EnableGlobalMethodSecurity(prePostEnabled = **true**)

**public** **class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {

@Autowired

UserDetailsServiceImpl userDetailsService;

@Autowired

**private** JwtAuthenticationEntryPoint unauthorizedHandler;

@Bean

**public** JwtTokenAuthOncePerRequestFilter authenticationJwtTokenFilter() {

**return** **new** JwtTokenAuthOncePerRequestFilter();

}

@Override

**public** **void** configure(AuthenticationManagerBuilder authenticationManagerBuilder) **throws** Exception {

authenticationManagerBuilder.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());

}

@Bean

@Override

**public** AuthenticationManager authenticationManagerBean() **throws** Exception {

**return** **super**.authenticationManagerBean();

}

@Bean

**public** PasswordEncoder passwordEncoder() {

**return** **new** BCryptPasswordEncoder();

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.cors().and().csrf().disable()

.exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.***STATELESS***).and()

.authorizeRequests().antMatchers("/api/authenticate/\*\*").permitAll()

.antMatchers("/api/resource/\*\*").permitAll()

.anyRequest().authenticated();

http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswordAuthenticationFilter.**class**);

}

}

**Filter the Requests:**

Create **JwtTokenAuthOncePerRequestFilter.java** class and place into the package **com.implementation.jwt.security . JwtTokenAuthOncePerRequestFilter** classextend spring web filterOncePerRequestFilter class. For any incoming request this filter class will execute and it check the request has the valid JWT token, if the token is valid then it set the authentication context as authenticate to current user perspective.

**package** com.implementation.jwt.security;

**import** java.io.IOException;

**import** javax.servlet.FilterChain;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

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

**import** org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

**import** org.springframework.security.core.context.SecurityContextHolder;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

**import** org.springframework.util.StringUtils;

**import** org.springframework.web.filter.OncePerRequestFilter;

**import** com.implementation.jwt.service.UserDetailsServiceImpl;

**public** **class** JwtTokenAuthOncePerRequestFilter **extends** OncePerRequestFilter {

@Autowired

**private** JwtUtils jwtUtils;

@Autowired

**private** UserDetailsServiceImpl userDetailsService;

**private** **static** **final** Logger ***logger*** = LoggerFactory.*getLogger*(JwtTokenAuthOncePerRequestFilter.**class**);

@Override

**protected** **void** doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)

**throws** ServletException, IOException {

**try** {

String jwt = parseJwt(request);

**if** (jwt != **null** && jwtUtils.validateJwtToken(jwt)) {

String username = jwtUtils.getUserNameFromJwtToken(jwt);

UserDetails userDetails = userDetailsService.loadUserByUsername(username);

UsernamePasswordAuthenticationToken authentication = **new** UsernamePasswordAuthenticationToken(

userDetails, **null**, userDetails.getAuthorities());

authentication.setDetails(**new** WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.*getContext*().setAuthentication(authentication);

}

} **catch** (Exception e) {

***logger***.error("Cannot set user authentication: {}", e);

}

filterChain.doFilter(request, response);

}

**private** String parseJwt(HttpServletRequest request) {

String headerAuth = request.getHeader("Authorization");

**if** (StringUtils.*hasText*(headerAuth) && headerAuth.startsWith("Bearer ")) {

**return** headerAuth.substring(7, headerAuth.length());

}

**return** **null**;

}

}

This class will extend spring’s JwtAuthenticationEntryPoint.java class and override its method to commence. It rejects every unauthenticated request and sends error code 401. Also place this class into the package com.implementation.jwt.security

**package** com.implementation.jwt.security;

**import** java.io.IOException;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.security.core.AuthenticationException;

**import** org.springframework.security.web.AuthenticationEntryPoint;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** JwtAuthenticationEntryPoint **implements** AuthenticationEntryPoint {

**private** **static** **final** Logger ***logger*** = LoggerFactory.*getLogger*(JwtAuthenticationEntryPoint.**class**);

@Override

**public** **void** commence(HttpServletRequest request, HttpServletResponse response,

AuthenticationException authException) **throws** IOException, ServletException {

***logger***.error("Unauthorized error: {}", authException.getMessage());

response.sendError(HttpServletResponse.***SC\_UNAUTHORIZED***, "Error: Unauthorized");

}

}

**Create JWT Utility Class:**

JwtUtils.java class is responsible to generate and validating the JWT token. Here we generating the JWT using **HS512 algorithm and Secrete key.** And we are parsing the JWT using secrete key during the validation.

**package** com.implementation.jwt.security;

**import** java.util.Date;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.beans.factory.annotation.Value;

**import** org.springframework.security.core.Authentication;

**import** org.springframework.stereotype.Component;

**import** com.implementation.jwt.service.UserDetailsImpl;

**import** io.jsonwebtoken.ExpiredJwtException;

**import** io.jsonwebtoken.Jwts;

**import** io.jsonwebtoken.MalformedJwtException;

**import** io.jsonwebtoken.SignatureAlgorithm;

**import** io.jsonwebtoken.SignatureException;

**import** io.jsonwebtoken.UnsupportedJwtException;

@Component

**public** **class** JwtUtils {

**private** **static** **final** Logger ***logger*** = LoggerFactory.*getLogger*(JwtUtils.**class**);

@Value("${jwt.signing.key.secret}")

**private** String jwtSecret;

@Value("${jwt.token.expiration.in.seconds}")

**private** **int** jwtExpirationMs;

**public** String generateJwtToken(Authentication authentication) {

UserDetailsImpl userPrincipal = (UserDetailsImpl) authentication.getPrincipal();

**return** Jwts.*builder*()

.setSubject((userPrincipal.getUsername()))

.setIssuedAt(**new** Date())

.setExpiration(**new** Date((**new** Date()).getTime() + jwtExpirationMs \* 1000))

.signWith(SignatureAlgorithm.***HS512***, jwtSecret)

.compact();

}

**public** String getUserNameFromJwtToken(String token) {

**return** Jwts.*parser*().setSigningKey(jwtSecret).parseClaimsJws(token).getBody().getSubject();

}

**public** **boolean** validateJwtToken(String authToken) {

**try** {

Jwts.*parser*().setSigningKey(jwtSecret).parseClaimsJws(authToken);

**return** **true**;

} **catch** (SignatureException e) {

***logger***.error("Invalid JWT signature: {}", e.getMessage());

} **catch** (MalformedJwtException e) {

***logger***.error("Invalid JWT token: {}", e.getMessage());

} **catch** (ExpiredJwtException e) {

***logger***.error("JWT token is expired: {}", e.getMessage());

} **catch** (UnsupportedJwtException e) {

***logger***.error("JWT token is unsupported: {}", e.getMessage());

} **catch** (IllegalArgumentException e) {

***logger***.error("JWT claims string is empty: {}", e.getMessage());

}

**return** **false**;

}

}

**Create API Controller:**

Create a package com.implementation.jwt.controller and JwtController.java

In this controller we have implemented two POST API for user registration with mapping **/reg** and user login with post mapping **/login.**

To register, user should provide a user name, password and a role list. If the user does not specify the roles, then it will automatically pick the user role. Request body will user RegistrationRequest.java model class and Response will be providing by using MessageResponse.java model.

To get the JWT token, user have to provide valid username and password in /api/authenticate /logn post body. Here requests model is JwtLoginRequest.java and response model is JwtResponse.java

**package** com.implementation.jwt.controller;

**import** java.util.HashSet;

**import** java.util.List;

**import** java.util.Set;

**import** java.util.stream.Collectors;

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

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.security.authentication.AuthenticationManager;

**import** org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

**import** org.springframework.security.core.Authentication;

**import** org.springframework.security.core.context.SecurityContextHolder;

**import** org.springframework.security.crypto.password.PasswordEncoder;

**import** org.springframework.web.bind.annotation.CrossOrigin;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

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

**import** com.implementation.jwt.model.JwtLoginRequest;

**import** com.implementation.jwt.model.JwtResponse;

**import** com.implementation.jwt.model.MessageResponse;

**import** com.implementation.jwt.model.RegistrationRequest;

**import** com.implementation.jwt.model.Role;

**import** com.implementation.jwt.model.RoleEnum;

**import** com.implementation.jwt.model.User;

**import** com.implementation.jwt.repository.RoleRepository;

**import** com.implementation.jwt.repository.UserRepository;

**import** com.implementation.jwt.security.JwtUtils;

**import** com.implementation.jwt.service.UserDetailsImpl;

@CrossOrigin(origins = "\*", maxAge = 3600)

@RestController

@RequestMapping("/api/authenticate")

**public** **class** JwtController {

@Autowired

AuthenticationManager authenticationManager;

@Autowired

UserRepository userRepository;

@Autowired

RoleRepository roleRepository;

@Autowired

PasswordEncoder encoder;

@Autowired

JwtUtils jwtUtils;

@PostMapping("/login")

**public** ResponseEntity<?> authenticateUser(@RequestBody JwtLoginRequest loginRequest) {

Authentication authentication = authenticationManager.authenticate(

**new** UsernamePasswordAuthenticationToken(loginRequest.getUsername(), loginRequest.getPassword()));

SecurityContextHolder.*getContext*().setAuthentication(authentication);

String jwt = jwtUtils.generateJwtToken(authentication);

UserDetailsImpl userDetails = (UserDetailsImpl) authentication.getPrincipal();

List<String> roles = userDetails.getAuthorities().stream()

.map(item -> item.getAuthority())

.collect(Collectors.*toList*());

**return** ResponseEntity.*ok*(**new** JwtResponse(jwt,

userDetails.getId(),

userDetails.getUsername(),

roles));

}

@PostMapping("/reg")

**public** ResponseEntity<?> registerUser(@RequestBody RegistrationRequest signUpRequest) {

**if** (userRepository.existsByUsername(signUpRequest.getUsername())) {

**return** ResponseEntity

.*badRequest*()

.body(**new** MessageResponse("Username is already exist!"));

}

// Create new user's account

User user = **new** User(signUpRequest.getUsername(),

encoder.encode(signUpRequest.getPassword()));

Set<String> strRoles = signUpRequest.getRole();

Set<Role> roles = **new** HashSet<>();

**if** (strRoles == **null**) {

Role userRole = roleRepository.findByName(RoleEnum.***ROLE\_USER***)

.orElseThrow(() -> **new** RuntimeException("Role is not found."));

roles.add(userRole);

} **else** {

strRoles.forEach(role -> {

**switch** (role) {

**case** "admin":

Role adminRole = roleRepository.findByName(RoleEnum.***ROLE\_ADMIN***)

.orElseThrow(() -> **new** RuntimeException("Role is not found."));

roles.add(adminRole);

**break**;

**default**:

Role userRole = roleRepository.findByName(RoleEnum.***ROLE\_USER***)

.orElseThrow(() -> **new** RuntimeException("Role is not found."));

roles.add(userRole);

}

});

}

user.setRoles(roles);

userRepository.save(user);

**return** ResponseEntity.*ok*(**new** MessageResponse("Registration successfully!"));

}

}

**Create Resource Controller:**

Now we will create another controller which will use the JWT token within the authorization header to access the resource and it will also check the user role specified during the registration. Here we create 3 GET API to show the demo:

Public Access: <http://localhost:8080/api/resource/content>

Grant access whose are admin or user role and valid JWT: <http://localhost:8080/api/resource/user>

Grant access who has admin role only and valid JWT: <http://localhost:8080/api/resource/admin>

ResourceController.java

**package** com.implementation.jwt.controller;

**import** org.springframework.security.access.prepost.PreAuthorize;

**import** org.springframework.web.bind.annotation.CrossOrigin;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.RequestMapping;

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

@CrossOrigin(origins = "\*", maxAge = 3600)

@RestController

@RequestMapping("/api/resource")

**public** **class** ResourceController {

@GetMapping("/content")

**public** String publicContent() {

**return** "Public Content.";

}

@GetMapping("/user")

@PreAuthorize("hasRole('USER') or hasRole('ADMIN')")

**public** String userAccess() {

**return** "User Content.";

}

@GetMapping("/admin")

@PreAuthorize("hasRole('ADMIN')")

**public** String adminAccess() {

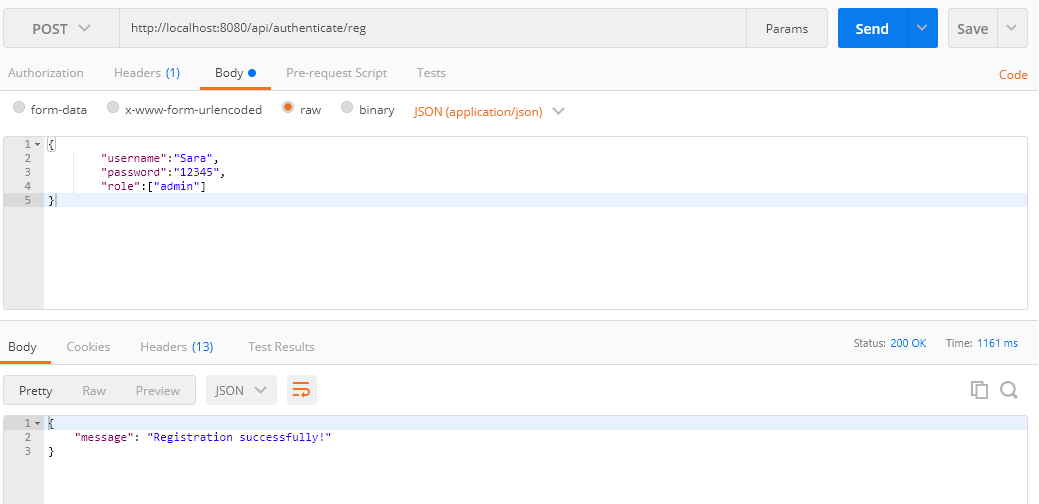
**return** "Admin content";

}

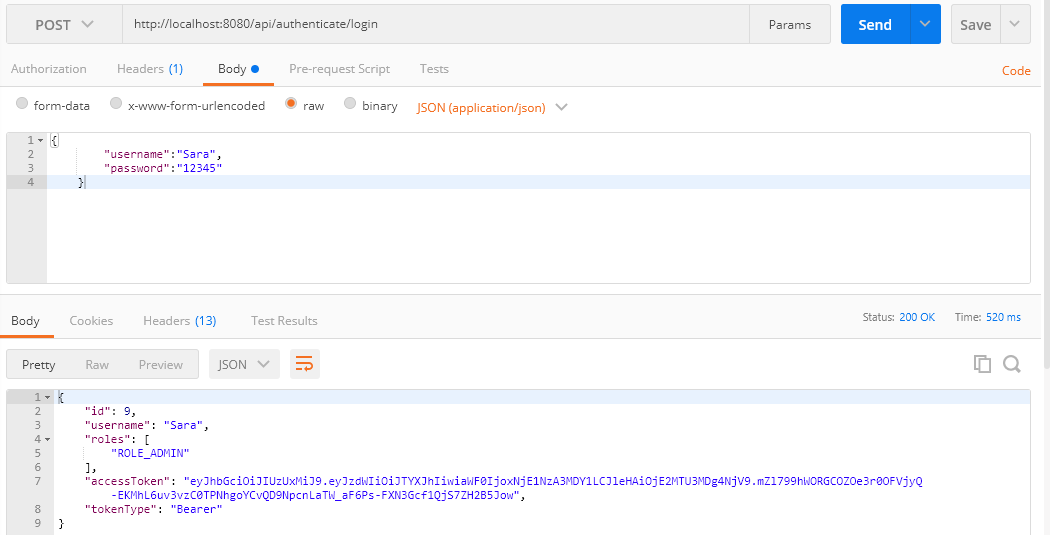
}

**Now check how it’s work using postman:**

For user registration:



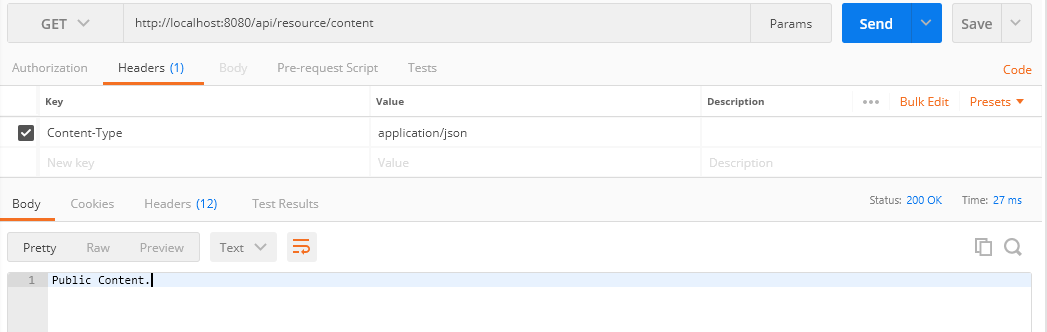
**For user login and generation JWT token:**



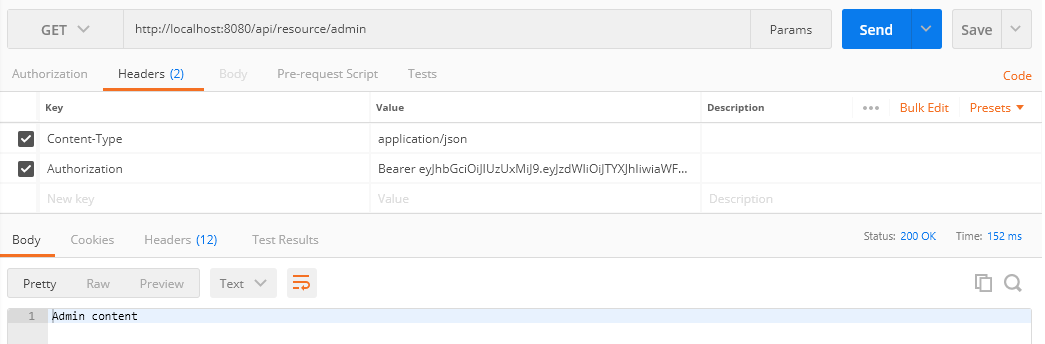
**To check the role and token based accessibility call the GET API:**

**URL:** <http://localhost:8080/api/resource/content>

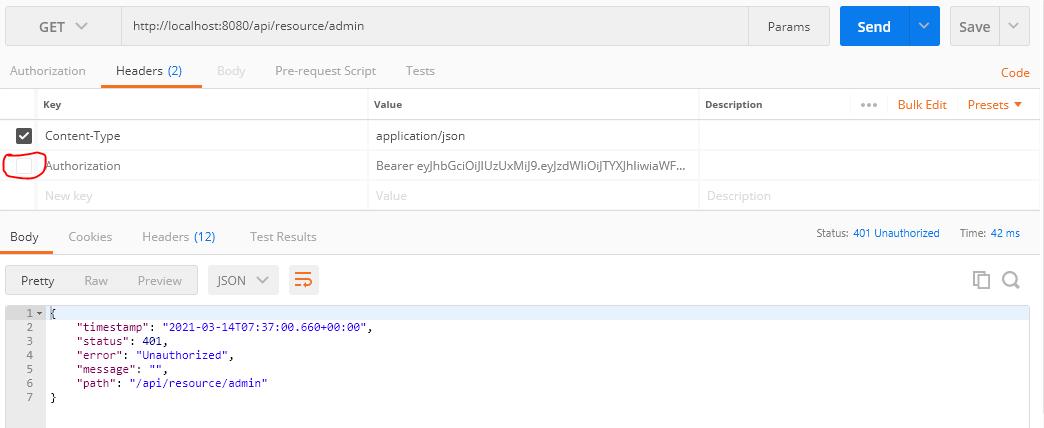
**We make it public and no need to set the authorization header for this API**



**URL:** [**http://localhost:8080/api/resource/admin**](http://localhost:8080/api/resource/admin)



**Let’s try without JWT:**



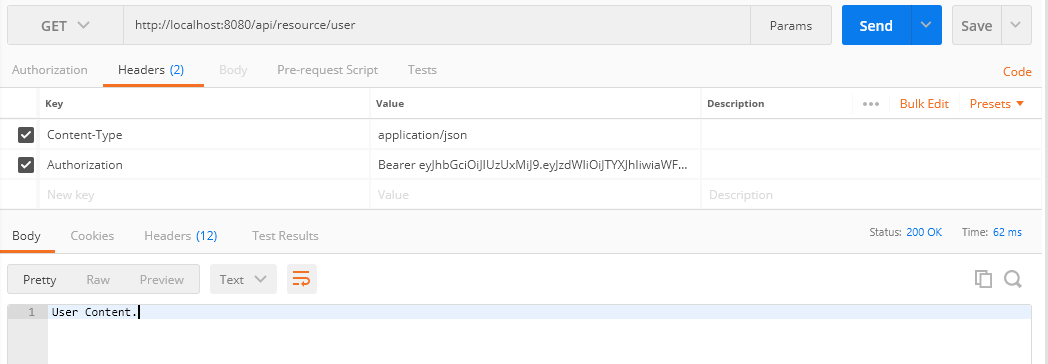
**Let try to access the user resource with admin JWT token and It should give the access. Because we permit it in the role controller:**

@PreAuthorize("hasRole('USER') or hasRole('ADMIN')")

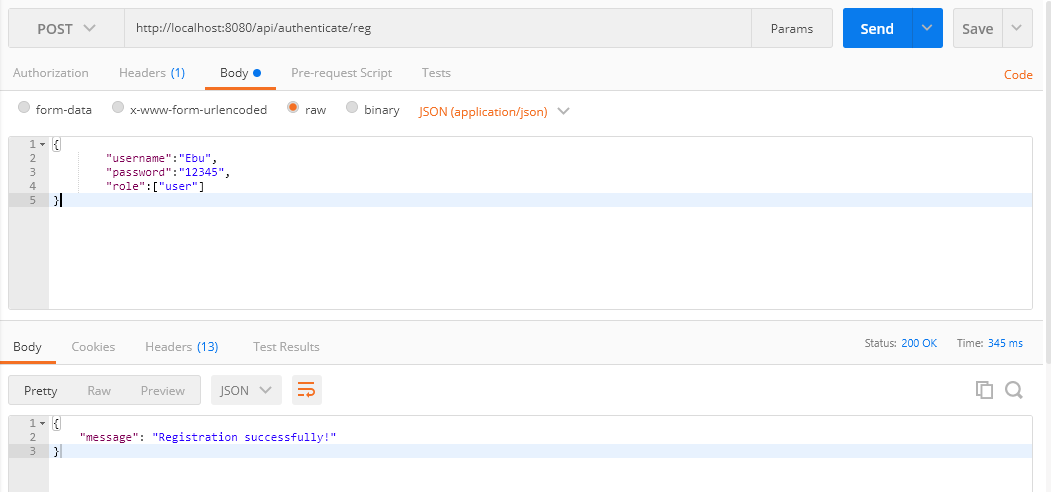
**public** String userAccess() {

**return** "User Content.";

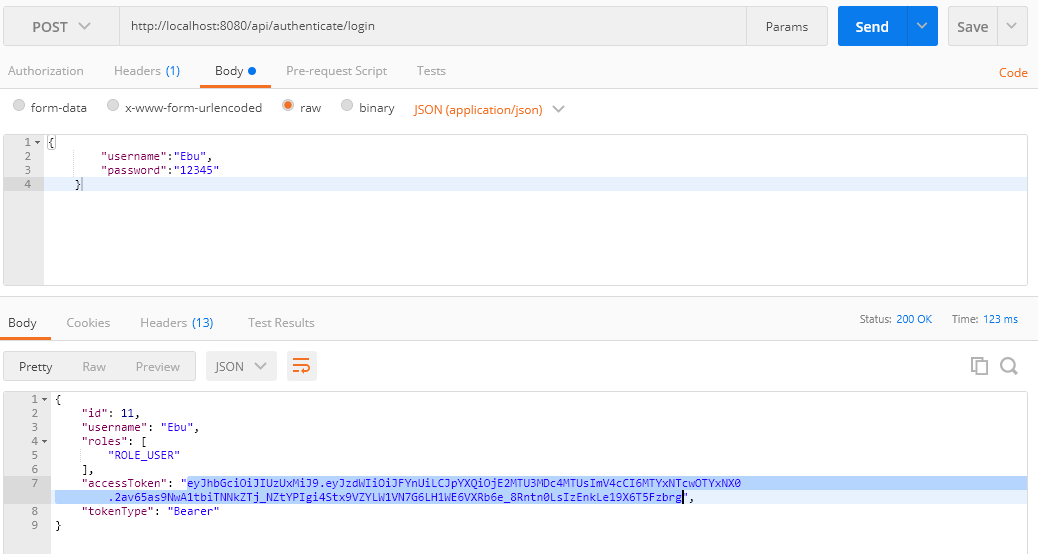
}



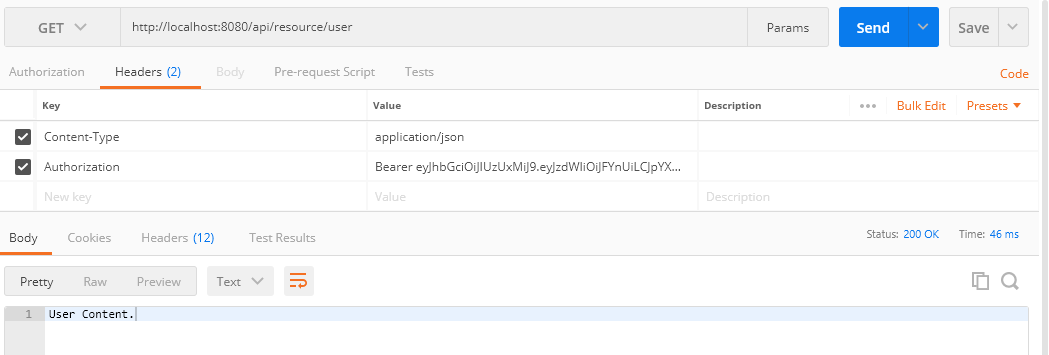
**Let’s create a user with user role:**



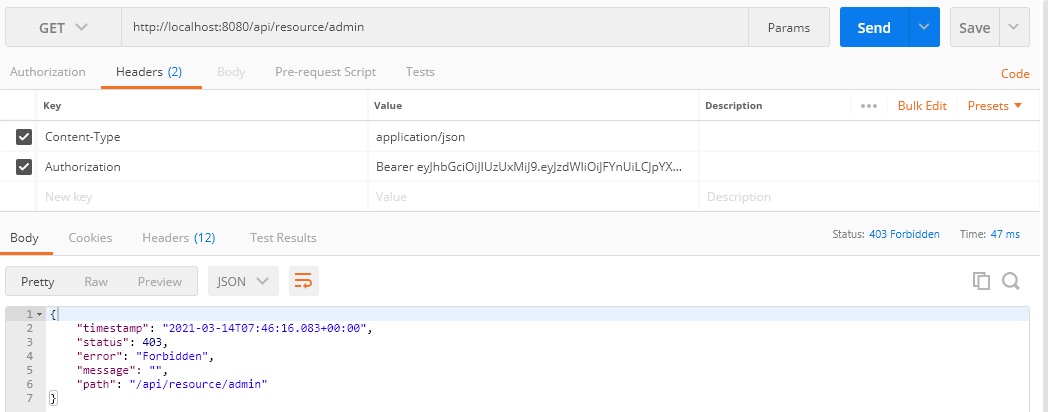
**Generate token (JWT) using this user:**



**Now try to access user API with this token:**



**Now try to access admin resource using this user role JWT token and It should restrict the access.**



So, this all about the JSON WEB TOKEN (JWT) with role based authentication. Now, let’s try yourself.