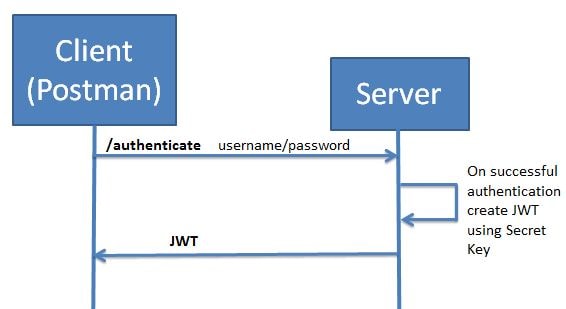
**Implement Spring Boot + JSON Web Token Security**

In this tutorial we will also be implementing Spring Boot + JSON Web Token Security. We will be modifying the [Spring Security project we had implemented in the previous tutorial](https://www.javainuse.com/webseries/spring-security-jwt/chap3) to make use of JSON Web Token Security. This implementation we will be dividing into 2 parts -

* Generate JSON Web Token
* Validate and use JWT for Authorization

Generate JSON Web Token

If the username and password sent by the user is successfully authenticated, server will then be generating a JSON Web Token and returning it to the client.  
  
  
In the pom.xml add the jwt dependency-

<groupId>com.javainuse</groupId>

<artifactId>spring-boot-security</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>spring-boot-security</name>

<description>Demo project for Spring Boot</description>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

**<dependency>**

**<groupId>io.jsonwebtoken</groupId>**

**<artifactId>jjwt</artifactId>**

**<version>0.9.1</version>**

**</dependency>**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

This is the JWT we will be creating if the authentication is successful.  
  
In the application.properties file specify the value for secret key and the expiration time.

* **Secret Key -**In [Understanding JWT Structure Tutorial](https://www.javainuse.com/webseries/spring-security-jwt/chap2) we had created the signature for the JWT using HS512Algo(Header + Payload + Secret Key).  
  The JWT has this signature present. If any user intercepts and tampers with the payload, then he cannot change the signature as he does not have the secret key. So the server which has the secret key will know that the JWT has been tampered with as the signature created by it using the secret key will not match the secret key in the JWT.  
  **Our secret key value will be javainuse**
* **Expiration Time -**This the time for which we want the generated JWT to be valid for. This will be in milliseconds. **Suppose we want the JWT to be valid for 5 hours, then we will specify this value as 18000000**

jwt.secret=javainuse

jwt.jwtExpirationInMs=18000000

Create the JWTUtil class. As the name suggest this class will have utility methods corresponding to the JWT like create JWT, check if JWT is valid.  
Currently we will be defining only the methods to create the JWT. Other methods we will be creating later as needed.

package com.javainuse.springbootsecurity.config;

import java.util.Collection;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

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

import org.springframework.security.core.GrantedAuthority;

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

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

import org.springframework.stereotype.Service;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

@Service

public class JwtUtil {

private String secret;

private int jwtExpirationInMs;

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

public void setSecret(String secret) {

this.secret = secret;

}

@Value("${jwt.expirationDateInMs}")

public void setJwtExpirationInMs(int jwtExpirationInMs) {

this.jwtExpirationInMs = jwtExpirationInMs;

}

// generate token for user

public String generateToken(UserDetails userDetails) {

Map<String, Object> claims = new HashMap<>();

Collection<? extends GrantedAuthority> roles = userDetails.getAuthorities();

if (roles.contains(new SimpleGrantedAuthority("ROLE\_ADMIN"))) {

claims.put("isAdmin", true);

}

if (roles.contains(new SimpleGrantedAuthority("ROLE\_USER"))) {

claims.put("isUser", true);

}

return doGenerateToken(claims, userDetails.getUsername());

}

private String doGenerateToken(Map<String, Object> claims, String subject) {

return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + jwtExpirationInMs)).signWith(SignatureAlgorithm.HS512, secret).compact();

}

}

Create the AuthenticationRequest class. The JSON input request provided by the user will be unmarshalled to Java Object using this class.

package com.javainuse.springbootsecurity.model;

public class AuthenticationRequest {

private String username;

private String password;

public AuthenticationRequest(String username, String password) {

this.username = username;

this.password = password;

}

public AuthenticationRequest() {

}

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;

}

}

Create the AuthenticationResponse class which is a model class to return the token on successful authentication

package com.javainuse.springsecurity.model;

public class AuthenticationResponse {

private String token;

public AuthenticationResponse(String token) {

this.token = token;

}

public AuthenticationResponse() {

}

public String getToken() {

return token;

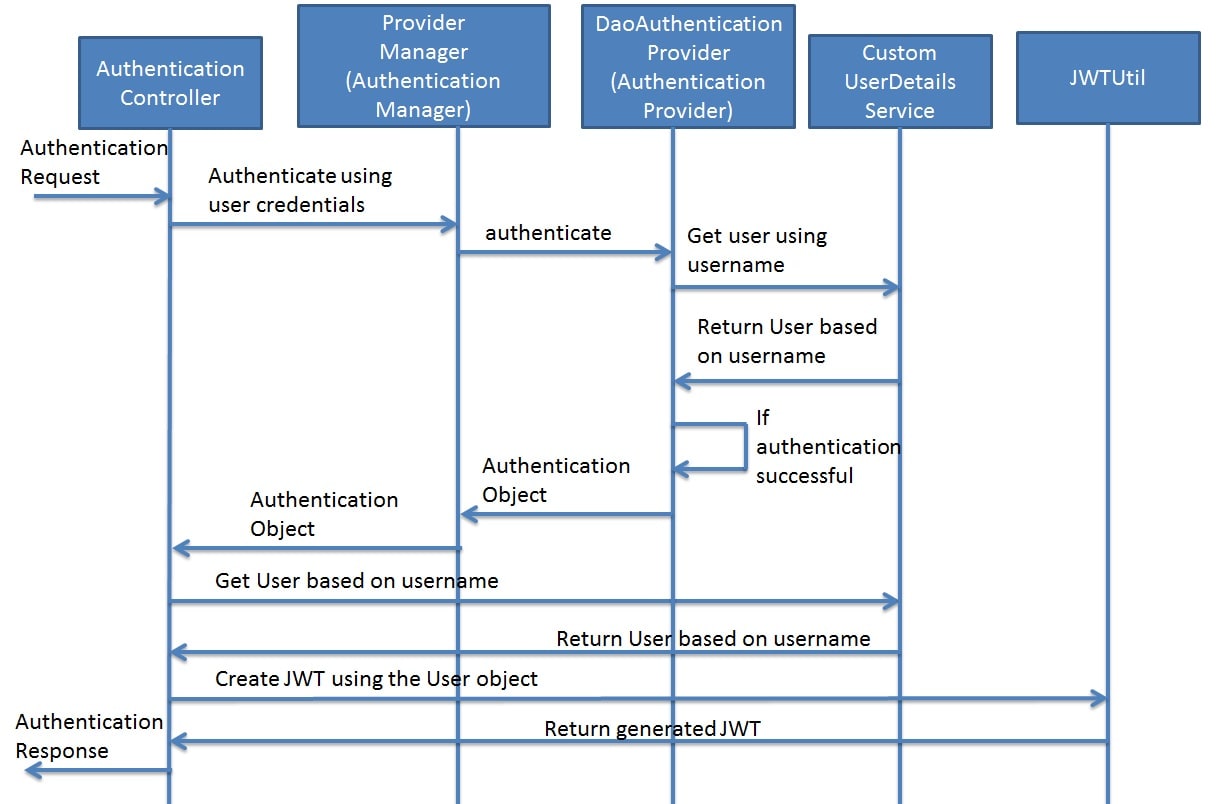
}

public void setToken(String token) {

this.token = token;

}

}

Next create a AuthenticationController class which will expose a POST REST API to take username and password from the user and if user is authenticated then return a JSON Web Token. For Authentication we will be making use of the AuthenticationManager which we have already configured in the SecurityConfiguration. This flow is quite similar to the previous [Spring Boot Security Project](https://www.javainuse.com/webseries/spring-security-jwt/chap3) where we has seen the Spring Boot Security Architecture and the Authentication Manager authenticates the incoming HTTP request.  
Once the authentication is successful we will be making a call to the generateToken method of the JwtUtil class which will create the token. This token will be returned back to the user.  


package com.javainuse.springsecurity.controller;

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

import org.springframework.http.ResponseEntity;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.BadCredentialsException;

import org.springframework.security.authentication.DisabledException;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

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

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

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

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

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

import com.javainuse.springsecurity.config.CustomUserDetailsService;

import com.javainuse.springsecurity.config.JwtUtil;

import com.javainuse.springsecurity.model.AuthenticateRequest;

import com.javainuse.springsecurity.model.AuthenticationResponse;

@RestController

public class AuthenticationController {

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private CustomUserDetailsService userDetailsService;

@Autowired

private JwtUtil jwtTokenUtil;

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

public ResponseEntity<?> createAuthenticationToken(@RequestBody AuthenticateRequest authenticationRequest)

throws Exception {

try {

authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(

authenticationRequest.getUsername(), authenticationRequest.getPassword()));

} catch (DisabledException e) {

throw new Exception("USER\_DISABLED", e);

} catch (BadCredentialsException e) {

throw new Exception("INVALID\_CREDENTIALS", e);

}

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

final String token = jwtTokenUtil.generateToken(userDetails);

return ResponseEntity.ok(new AuthenticationResponse(token));

}

}

Finally in the Spring Security Configuration

* Allow the /authenticate url without any authentication
* We have already configured the Authentication Manager using the AuthenticationManagerBuilder. But Spring Security needs us to explicitly create the AuthenticationManager Bean.

package com.javainuse.springbootsecurity.config;

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.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.crypto.bcrypt.BCryptPasswordEncoder;

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

@Configuration

@EnableWebSecurity

public class SpringSecurityConfiguration extends WebSecurityConfigurerAdapter{

@Autowired

CustomUserDetailsService userDetailsService;

public PasswordEncoder passwordEncoder(){

return new BCryptPasswordEncoder();

}

@Override

public void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());

}

**@Bean**

**@Override**

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

**return super.authenticationManagerBean();**

**}**

@Override

public void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests().antMatchers("/helloadmin").hasRole("ADMIN")

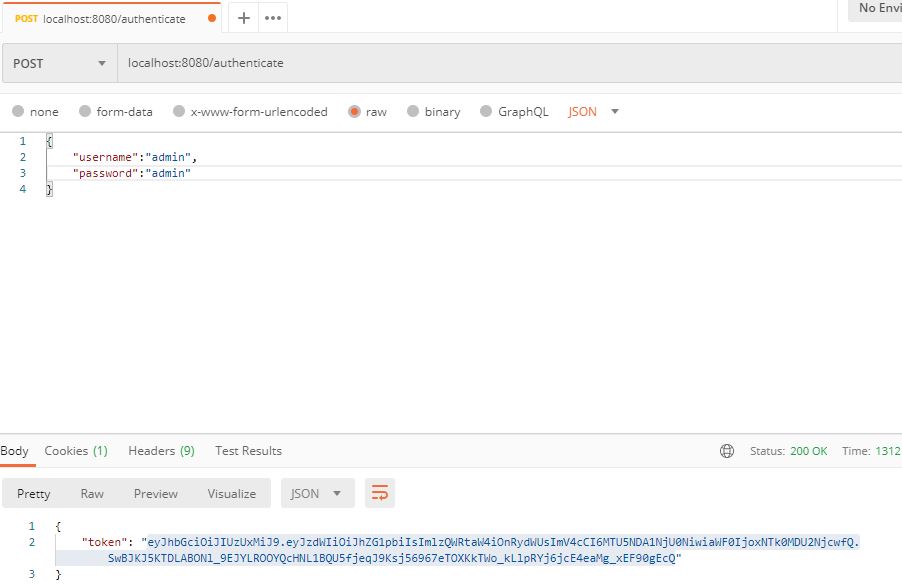
.antMatchers("/hellouser").hasAnyRole("USER","ADMIN")

**.antMatchers("/authenticate").permitAll().anyRequest().authenticated()**

.and().httpBasic();

}

}

Start the application and test the POST request /authenticate url-  
  
We can go to the [online JWT decoder](https://www.javainuse.com/decodeJWT) and check the JWT details-

Validation and Authorization using JWT

We will now be using the generated JWT to authorize user to perform operations. In [previous tutorial we have seen that any incoming request is first intercepted by Filters which perform authentication and authorization.](https://www.javainuse.com/webseries/spring-security-jwt/chap3) One example of such filter is the BasicAuthenticationFilter which is a type of OncePerRequestFilter. We will now be writing our own CustomJWTAuthenticationFilter which will also be type of OncePerRequestFilter.  
This CustomJWTAuthenticationFilter will intercept the incoming request and check if it contains a JSON Web Token(JWT). If JWT is present it will then call the validate method of the JWTUtil class to validate the token. If the validation is successful, then it will create a User Object using the JWT payload and store this object in the Security context, which indicates that the current user is authenticated.  
We will first be adding the some more utility methods like validateToken to the JWTUtil class.

package com.javainuse.springbootsecurity.config;

import java.util.Arrays;

import java.util.Collection;

import java.util.Date;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

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

import org.springframework.security.authentication.BadCredentialsException;

import org.springframework.security.core.GrantedAuthority;

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

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

import org.springframework.stereotype.Service;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.ExpiredJwtException;

import io.jsonwebtoken.Jws;

import io.jsonwebtoken.JwsHeader;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.MalformedJwtException;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.SignatureException;

import io.jsonwebtoken.UnsupportedJwtException;

@Service

public class JwtUtil {

private String secret;

private int jwtExpirationInMs;

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

public void setSecret(String secret) {

this.secret = secret;

}

@Value("${jwt.expirationDateInMs}")

public void setJwtExpirationInMs(int jwtExpirationInMs) {

this.jwtExpirationInMs = jwtExpirationInMs;

}

// generate token for user

public String generateToken(UserDetails userDetails) {

Map<String, Object> claims = new HashMap<>();

Collection<? extends GrantedAuthority> roles = userDetails.getAuthorities();

if (roles.contains(new SimpleGrantedAuthority("ROLE\_ADMIN"))) {

claims.put("isAdmin", true);

}

if (roles.contains(new SimpleGrantedAuthority("ROLE\_USER"))) {

claims.put("isUser", true);

}

return doGenerateToken(claims, userDetails.getUsername());

}

private String doGenerateToken(Map<String, Object> claims, String subject) {

return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + jwtExpirationInMs)).signWith(SignatureAlgorithm.HS512, secret).compact();

}

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

**try {**

**// Jwt token has not been tampered with**

**Jws<Claims> claims = Jwts.parser().setSigningKey(secret).parseClaimsJws(authToken);**

**return true;**

**} catch (SignatureException | MalformedJwtException | UnsupportedJwtException | IllegalArgumentException ex) {**

**throw new BadCredentialsException("INVALID\_CREDENTIALS", ex);**

**} catch (ExpiredJwtException ex) {**

**throw new ex(header, claims, "Token has Expired", ex);**

**}**

**}**

**public String getUsernameFromToken(String token) {**

**Claims claims = Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();**

**return claims.getSubject();**

**}**

**public List<SimpleGrantedAuthority> getRolesFromToken(String authToken) {**

**List<SimpleGrantedAuthority> roles = null;**

**Claims claims = Jwts.parser().setSigningKey(secret).parseClaimsJws(authToken).getBody();**

**Boolean isAdmin = claims.get("isAdmin", Boolean.class);**

**Boolean isUser = claims.get("isUser", Boolean.class);**

**if (isAdmin != null && isAdmin == true) {**

**roles = Arrays.asList(new SimpleGrantedAuthority("ROLE\_ADMIN"));**

**}**

**if (isUser != null && isUser == true) {**

**roles = Arrays.asList(new SimpleGrantedAuthority("ROLE\_USER"));**

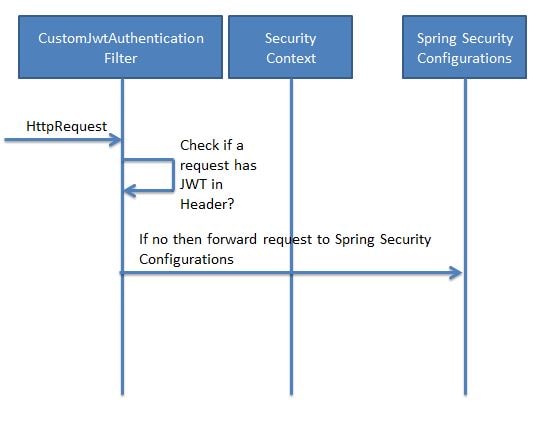
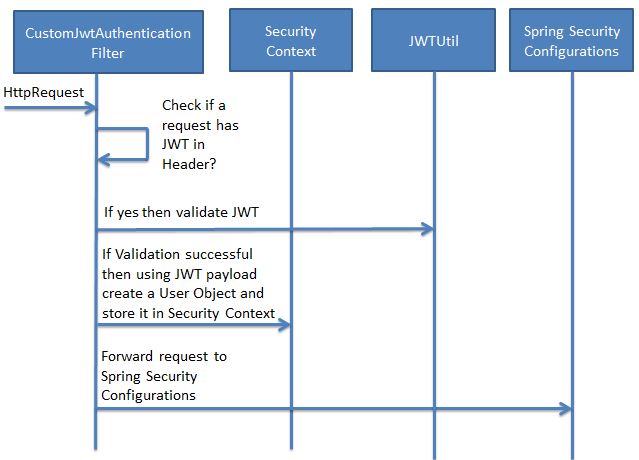
**}**

**return roles;**

**}**

}

Create the CustomJwtAuthenticationFilter class will be type of OncePerRequestFilter. This class will intercept the requests and

* Check if header contains a JWT. If no then let the usual Spring Security Flow take place.  
  
* If the header contains a JWT then it will validate the token. On successful validation it will add the User Object in the Spring Context to indicate that the user can be authorized to perform operation.  
  

package com.javainuse.springbootsecurity.config;

import java.io.IOException;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpServletResponseWrapper;

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

import org.springframework.security.authentication.BadCredentialsException;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

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

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

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

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

import org.springframework.stereotype.Component;

import org.springframework.util.StringUtils;

import org.springframework.web.filter.OncePerRequestFilter;

import io.jsonwebtoken.ExpiredJwtException;

@Component

public class CustomJwtAuthenticationFilter extends OncePerRequestFilter {

@Autowired

private JwtUtil jwtTokenUtil;

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)

throws ServletException, IOException {

// JWT Token is in the form "Bearer token". Remove Bearer word and

// get only the Token

String jwtToken = extractJwtFromRequest(request);

if (StringUtils.hasText(jwtToken) && jwtTokenUtil.validateToken(jwtToken)) {

UserDetails userDetails = new User(jwtTokenUtil.getUsernameFromToken(jwtToken), "",

jwtTokenUtil.getRolesFromToken(jwtToken));

UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(

userDetails, null, userDetails.getAuthorities());

// After setting the Authentication in the context, we specify

// that the current user is authenticated. So it passes the

// Spring Security Configurations successfully.

SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);

} else {

System.out.println("Cannot set the Security Context");

}

chain.doFilter(request, response);

}

private String extractJwtFromRequest(HttpServletRequest request) {

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

if (StringUtils.hasText(bearerToken) && bearerToken.startsWith("Bearer ")) {

return bearerToken.substring(7, bearerToken.length());

}

return null;

}

}

Create JwtAuthenticationEntryPoint class. This class is used to return a 401 unauthorized error to clients that try to access a protected resource without proper authentication. It implements Spring Security AuthenticationEntryPoint interface. In this class we will be creating the HttpResponse which should be returned to the user in case of an exception.

package com.javainuse.springbootsecurity.config;

import java.io.IOException;

import java.util.Collections;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.http.MediaType;

import org.springframework.security.core.AuthenticationException;

import org.springframework.security.web.AuthenticationEntryPoint;

import org.springframework.stereotype.Component;

import com.fasterxml.jackson.databind.ObjectMapper;

@Component

public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {

@Override

public void commence(HttpServletRequest request, HttpServletResponse response,

AuthenticationException authException) throws IOException, ServletException {

response.setStatus(HttpServletResponse.SC\_UNAUTHORIZED);

response.setContentType(MediaType.APPLICATION\_JSON\_VALUE);

String message;

if (authException.getCause() != null) {

message = authException.getCause().toString() + " " + authException.getMessage();

} else {

message = authException.getMessage();

}

byte[] body = new ObjectMapper().writeValueAsBytes(Collections.singletonMap("error", message));

response.getOutputStream().write(body);

}

}

Modify Security Configuration-

* Remove HttpBasic Security, as we will be using JWT for authorization.
* Configure Http Security to make use of the CustomJwtAuthenticationFilter and the JwtAuthenticationEntryPoint.

package com.javainuse.springbootsecurity.config;

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.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;

@Configuration

@EnableWebSecurity

public class SpringSecurityConfiguration extends WebSecurityConfigurerAdapter{

@Autowired

CustomUserDetailsService userDetailsService;

**@Autowired**

**private CustomJwtAuthenticationFilter customJwtAuthenticationFilter;**

**@Autowired**

**private JwtAuthenticationEntryPoint unauthorizedHandler;**

public PasswordEncoder passwordEncoder(){

return new BCryptPasswordEncoder();

}

@Override

public void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());

}

@Bean

@Override

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

@Override

public void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests().antMatchers("/helloadmin").hasRole("ADMIN")

.antMatchers("/hellouser").hasAnyRole("USER","ADMIN")

.antMatchers("/authenticate").permitAll().anyRequest().authenticated()

//if any exception occurs call this

**.and().exceptionHandling()**

**.authenticationEntryPoint(unauthorizedHandler).and().**

**// make sure we use stateless session; session won't be used to**

**// store user's state.**

**sessionManagement()**

**.sessionCreationPolicy(SessionCreationPolicy.STATELESS);**

**// Add a filter to validate the tokens with every request**

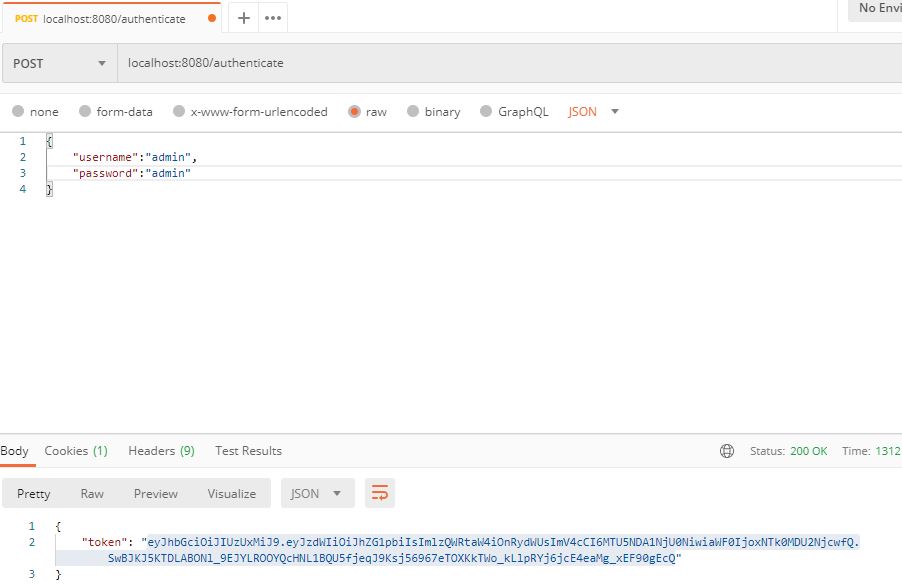
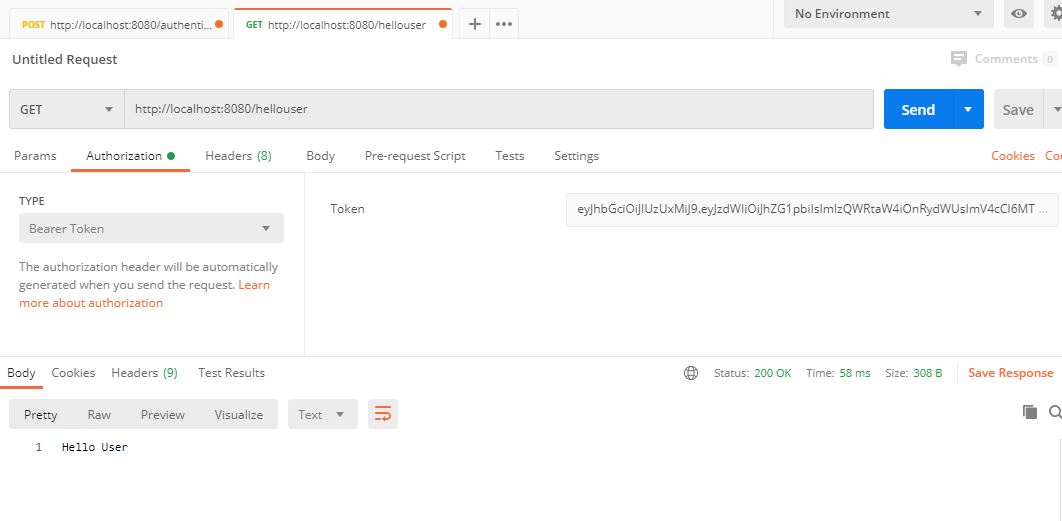
**http.addFilterBefore(customJwtAuthenticationFilter,**

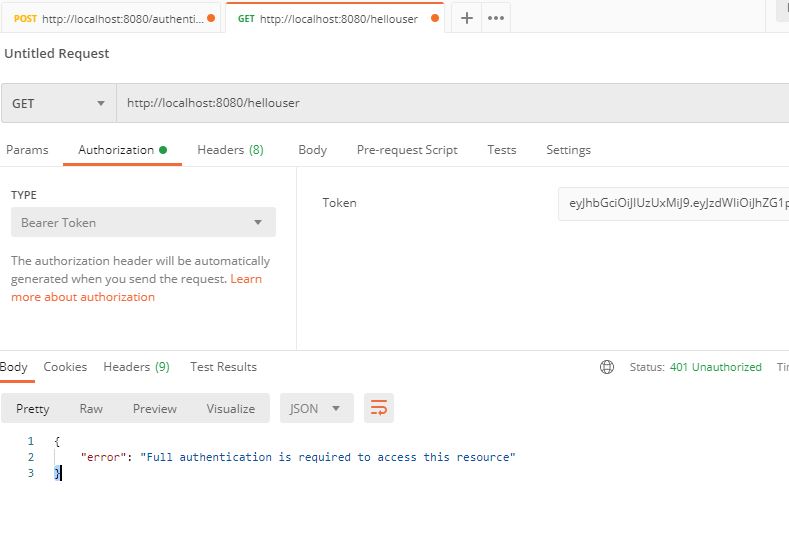
**UsernamePasswordAuthenticationFilter.class);**

}

}

Run the application.

* Create the token using the url localhost:8080/authenticate  
  
* Use this token to get response from the url localhost:8080/hellouser  
  

If we now make the jwt.expirationDateInMs property in the application.properties as 0 and create a token, then the created JWT should be expired as soon as it is created. Use this token to access the url localhost:8080/hellouser  
  
**We can see that instead of JWT exception we still get the exception that "Full Authentication is required". This is because of Spring Security issue/bug. We will next be some changes to the code so that the user gets the exception informing that the JWT has expired.**

Configuring and testing JWT Expiration Date

If the JWTUtil class throws the JWTExpiration Exception or the Bad Credentials Exception, we will be catching it in the CustomJwtAuthenticationFilter and saving it as a Request Attribute.

package com.javainuse.springbootsecurity.config;

import java.io.IOException;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpServletResponseWrapper;

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

import org.springframework.security.authentication.BadCredentialsException;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

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

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

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

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

import org.springframework.stereotype.Component;

import org.springframework.util.StringUtils;

import org.springframework.web.filter.OncePerRequestFilter;

import io.jsonwebtoken.ExpiredJwtException;

@Component

public class CustomJwtAuthenticationFilter extends OncePerRequestFilter {

@Autowired

private JwtUtil jwtTokenUtil;

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)

throws ServletException, IOException {

**try {**

// JWT Token is in the form "Bearer token". Remove Bearer word and

// get only the Token

String jwtToken = extractJwtFromRequest(request);

if (StringUtils.hasText(jwtToken) && jwtTokenUtil.validateToken(jwtToken)) {

UserDetails userDetails = new User(jwtTokenUtil.getUsernameFromToken(jwtToken), "",

jwtTokenUtil.getRolesFromToken(jwtToken));

UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(

userDetails, null, userDetails.getAuthorities());

// After setting the Authentication in the context, we specify

// that the current user is authenticated. So it passes the

// Spring Security Configurations successfully.

SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);

} else {

System.out.println("Cannot set the Security Context");

}

**} catch (ExpiredJwtException ex) {**

**request.setAttribute("exception", ex);**

**throw ex;**

**} catch (BadCredentialsException ex) {**

**request.setAttribute("exception", ex);**

**throw ex;**

**}**

chain.doFilter(request, response);

}

private String extractJwtFromRequest(HttpServletRequest request) {

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

if (StringUtils.hasText(bearerToken) && bearerToken.startsWith("Bearer ")) {

return bearerToken.substring(7, bearerToken.length());

}

return null;

}

}

Later in the JwtAuthenticationEntryPoint class we will be checking if the request has an attribute named exception. If it does have it then we will be making use of it to create the response object.

package com.javainuse.springbootsecurity.config;

import java.io.IOException;

import java.util.Collections;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.http.MediaType;

import org.springframework.security.core.AuthenticationException;

import org.springframework.security.web.AuthenticationEntryPoint;

import org.springframework.stereotype.Component;

import com.fasterxml.jackson.databind.ObjectMapper;

@Component

public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {

@Override

public void commence(HttpServletRequest request, HttpServletResponse response,

AuthenticationException authException) throws IOException, ServletException {

response.setStatus(HttpServletResponse.SC\_UNAUTHORIZED);

response.setContentType(MediaType.APPLICATION\_JSON\_VALUE);

String message;

// Check if the request as any exception that we have stored in Request

**final Exception exception = (Exception) request.getAttribute("exception");**

// If yes then use it to create the response message else use the authException

**if (exception != null) {**

**byte[] body = new ObjectMapper().writeValueAsBytes(Collections.singletonMap("cause", exception.toString()));**

**response.getOutputStream().write(body);**

**}** else {

if (authException.getCause() != null) {

message = authException.getCause().toString() + " " + authException.getMessage();

} else {

message = authException.getMessage();

}

byte[] body = new ObjectMapper().writeValueAsBytes(Collections.singletonMap("error", message));

response.getOutputStream().write(body);

}

}

}

If we now run the application and use an expired JWT we get the following custom exception -  
