Securing RESTful Web Services with Spring Security

Follow steps below to secure all web services using Spring Security:

- Open spring-learn project in Eclipse
- Include spring security related libraries by adding the below dependency in pom.xml

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

- Rebuild the project in command line using mvn clean package command (ensure to include proxy details in mvn command).
- To ensure the new libraries are enabled in Eclipse, right click the project and select Maven > Update Project
- Create a new package 'com.cognizant.spring-learn.security'
- Create a new class SecurityConfig in the new package created above which extends from WebSecurityConfigurerAdapter
- Include annotations @Configuration and @EnableWebSecurity at class level
- Import appropriate classes using Ctrl + Shift + O
- Start the application and check the logs and test the REST service. Refer command below:

```
curl -s http://localhost:8090/countries
```

- The following error message is the expected response:

```
{"timestamp":"2019-10-05T09:24:33.794+0000","status":401,"error":"Unauthorized
","message":"Unauthorized","path":"/countries"}
```

- The inclusion of @EnableWebSecurity has restricted access to all the web services with a common password.
- Refer the logs to find out the password generated. Now execute the invocation of the service with password as specified below, which should get the list of countries. include the password from the log file after user:.

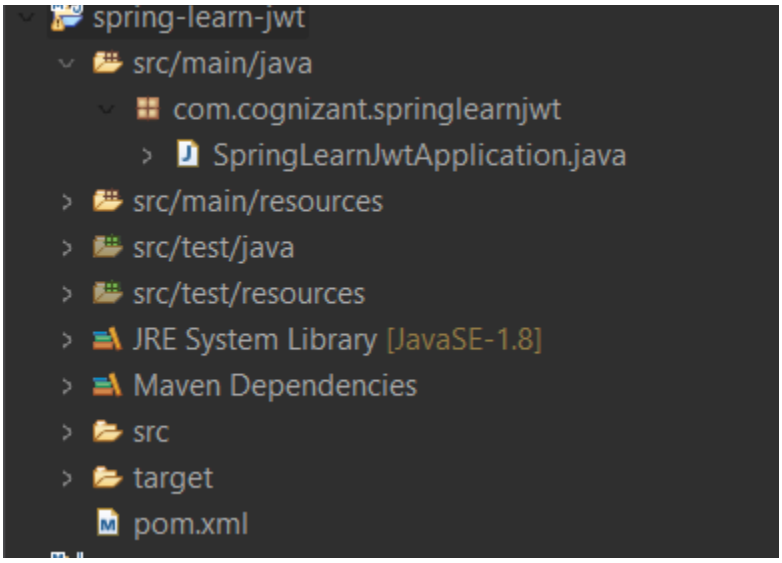
```
curl -s -v -u user:d27321a9-0751-4f59-8fc6-f8633847a9b8 http://localhost:8090/countries
```

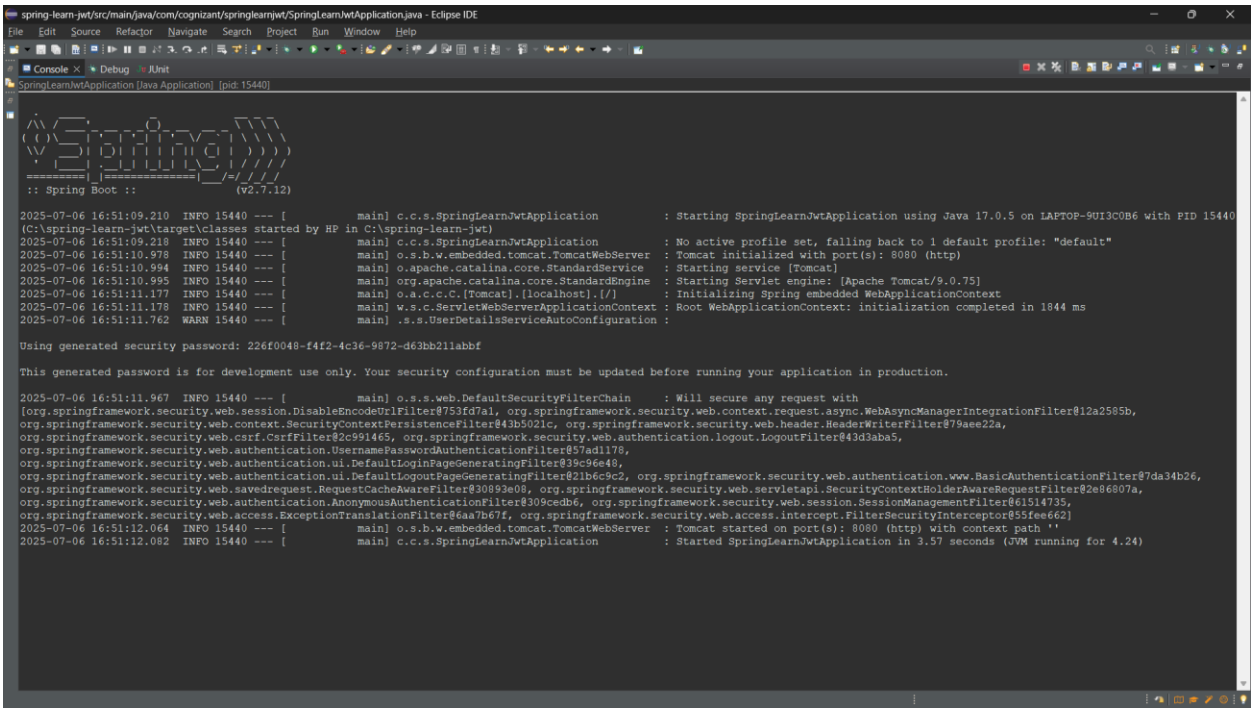
- Find below a sample response for the above command:

```
[{"code":"US","name":"United States"},{"code":"DE","name":"Germany"},{"code":"IN","name":"India"},{"code":"JP","name":"Japan"}]* timeout on name lookup is not supported

*   Trying ::1...

* TCP_NODELAY set

* Connected to localhost (::1) port 8090 (#0)

* Server auth using Basic with user 'user'

> GET /countries HTTP/1.1

> Host: localhost:8090

> Authorization: Basic dXNlcjpkMjczMjFhOS0wNzUxLTRmNTktOGZjNi1mODYzMzg0N2E5Yjg=

> User-Agent: curl/7.55.0

> Accept: */*

>

< HTTP/1.1 200

< Set-Cookie: JSESSIONID=C0C907417A21BBCA9F30BEEA4B512AEE; Path=/; HttpOnly

< X-Content-Type-Options: nosniff

< X-XSS-Protection: 1; mode=block

< Cache-Control: no-cache, no-store, max-age=0, must-revalidate

< Pragma: no-cache

< Expires: 0

< X-Frame-Options: DENY

< Content-Type: application/json;charset=UTF-8

< Transfer-Encoding: chunked

< Date: Sat, 05 Oct 2019 09:36:34 GMT

<

{ [133 bytes data]

* Connection #0 to host localhost left intact
```

- First line contains the country list responded successfully.
- The above result contains the request header and response header.

- The request lines starts with > and reponse lines starts with <
- Notice the Authorization header in the HTTP Request
- This denotes that it uses basic HTTP Authorisation. Whatever following Basic is Base64 encoding of the password that was supplied in the command line.






```
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] --- install:2.5.2:install (default-install) @ spring-learn-jwt ---
[INFO] Installing C:\spring-learn-jwt\target\spring-learn-jwt-0.0.1-SNAPSHOT.jar to C:\Users\HP\.m2\repository\com\cogni
zant\spring-learn\spring-learn-jwt\0.0.1-SNAPSHOT\spring-learn-jwt-0.0.1-SNAPSHOT.jar
[INFO] Installing C:\spring-learn-jwt\pom.xml to C:\Users\HP\.m2\repository\com\cognizant\spring-learn\spring-learn-jwt\
0.0.1-SNAPSHOT\spring-learn-jwt-0.0.1-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  17.613 s
[INFO] Finished at: 2025-07-07T20:27:02+05:30
[INFO] ------------------------------------------------------------------------
```

Creating users and roles in Spring Security

The earlier hands on demonstrated securing all URLs of the application with a common password. But it is not user and role specific.

Let us create two new in memory users with names 'admin' and 'user'. The password for both the users will be 'pwd'.

Let us define the rule that getting all countries can be accessed only 'user'.

Refer steps below to incorporate the above aspects:

- Include the below methods in the SecurityConfig class

```
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exceptio
n {
        auth.inMemoryAuthentication()
            .withUser("admin").password(passwordEncoder().encode("pwd")).roles(
"ADMIN")
            .and()
            .withUser("user").password(passwordEncoder().encode("pwd")).roles("
USER");
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        LOGGER.info("Start");
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.csrf().disable().httpBasic().and()
            .authorizeRequests().antMatchers("/countries").hasRole("USER");
```

```
    }
```

- The first configure() method defines two users admin and user with password as pwd. It also includes the specification of respective roles.

IMPORTANT NOTE: For learning purpose we are hard coding user details. When working on Spring Data JPA module, the credentials will be validate from the database.

- The password encoder is required to encrypt the password.
- The second configure() method defines that /countries services is accessible only to users of role "USER"
- For testing the service with right credentials:

```
curl -s -u user:pwd http://localhost:8090/countries
```

- For testing the service with incorrect credentials and response:

```
curl -s -u user:pwd1 http://localhost:8090/countries

{"timestamp":"2019-10-05T10:19:08.237+0000","status":401,"error":"Unauthorized"
,"message":"Unauthorized","path":"/countries"}
```

- For testing the service with correct credentials but a different role

```
curl -s -u admin:pwd http://localhost:8090/countries

{"timestamp":"2019-10-05T10:22:38.015+0000","status":403,"error":"Forbidden","m
essage":"Forbidden","path":"/countries"}
```

Limitations of this security approach

- RESTful Web Service is a stateless protocol, hence each request needs to be attached the with user id and password credentials.
- The credentials passed on the HTTP request is not secure. Refer steps below to understand this better:
- Execute the below command to display the request and response headers:

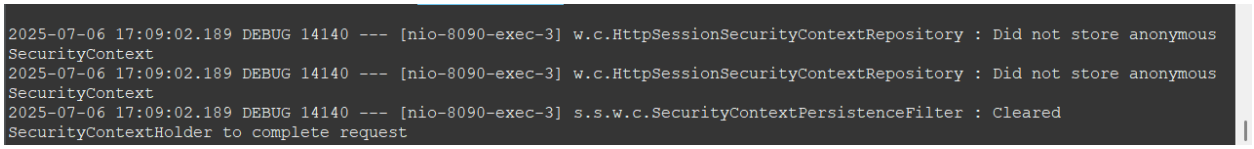
```
curl -s -v -u admin:pwd http://localhost:8090/countries
```

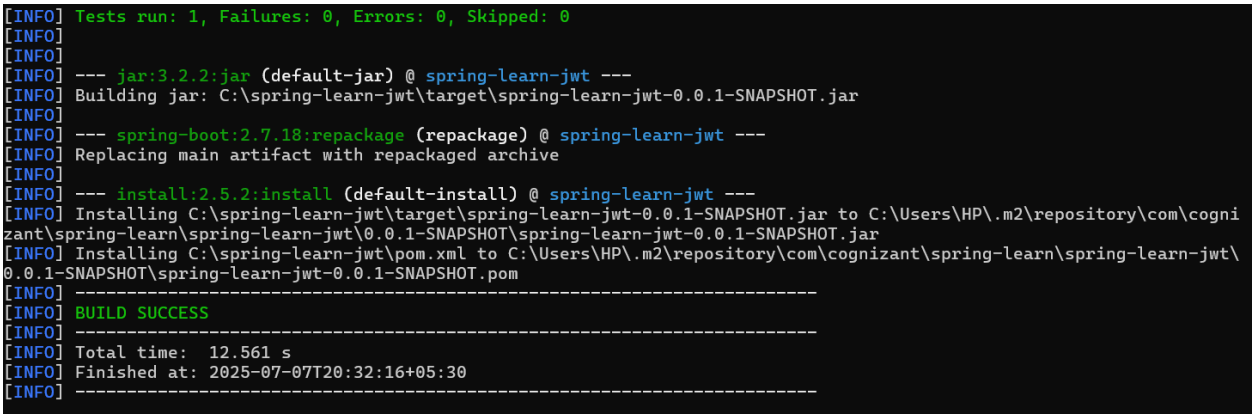
- In the result display, in the request section, refer the Authorization

```
> Authorization: Basic YWRtaW46cHdk
```

- If "admin:pwd" is encoded with Base64 it results in "YWRtaW46cHdk"
- Search using google and find out a online website that can decode Base64. (Example website, https://www.base64decode.net/)
- Try decoding YWRtaW46cHdk using the web site and one can obtain "admin:pwd"

These liminations can be overcome by incorporating security using JWT. Subsequent hands on will address this issue.





Understanding JWT

What is JWT?

- JWT stands for JSON Web Token
- Internet standard (**IETF 7519 Link**) for creating JSON-based access tokens
- JWT can be typically used to pass identity of authenticated users and service provider,

JWT Process Flow (**diagram link**)

- Client sends username and password to server
- Servers validates credentials, creates token (JWT) and reponds it back
- Client attaches the token in the subsequent requests to server
- Server validates the token (JWT) on each client request

Structure of JSON Web Token

- Reference: https://en.wikipedia.org/wiki/JSON_Web_Token#Structure
- Header: Contains the encryption algorithm
- Payload: Contains application specific data. Usually this contains the user id and role.
- Signature: Computed based on the formula defined using header and payload

Exercise to check how JWT token is created

- Open link https://en.wikipedia.org/wiki/JSON_Web_Token#Structure in browser
- Open link https://jwt.io/ in another browser tab and scroll down to the Encoded, Decoded section
- Copy and paste the header content from wikipedia article and paste it in header section of https://jwt.io
- Copy and paste the payload content from wikipedia article and paste it in payload section of https://jwt.io
- Type "secretkey" in the textbox within Verify Signature section
- Check if the token generated in the Encoded section of https://jwt.io matches with the generated token displayed in the Structure section of wikipedia article

Create authentication service that returns JWT

As part of first step of JWT process, the user credentials needs to be sent to authentication service request that generates and returns the JWT.

Ideally when the below curl command is executed that calls the new authentication service, the token should be responded. Kindly note that the credentials are passed using -u option.

Request

```
curl -s -u user:pwd http://localhost:8090/authenticate
```

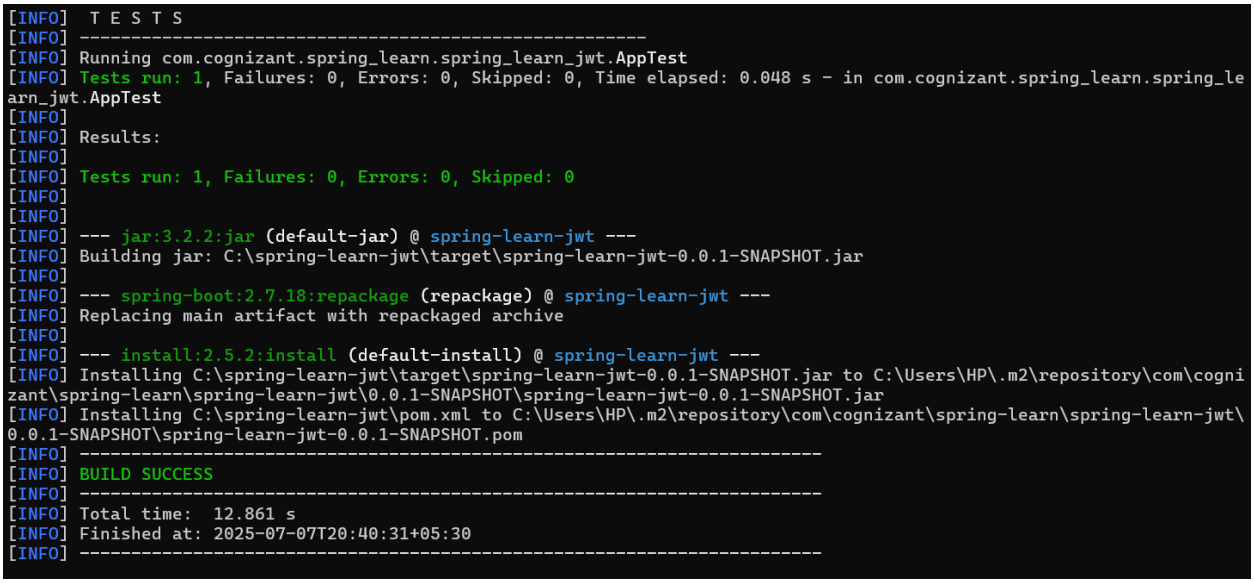
Response

```
{"token":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiaWF0IjoxNTcwMzc5NDc0LCJleHAiOjE1NzAzODA2NzR9.t3LRvlCV-hwKfoqZYlaVQqEUiBloWcWn0ft3tgv0dL0"}
```

This can be incorporated as three major steps:

- Create authentication controller and configure it in SecurityConfig
- Read Authorization header and decode the username and password
- Generate token based on the user retrieved in the previous step

Let incorporate the above as separate hands on exercises.



Create authentication controller and configure it in SecurityConfig

AuthenticationController.java

- Create new rest controller named AuthenticationController in controller package
- Include method authenticate with "/authenticate" as the URL with @GetMapping.
- To read the Authorization value from HTTP Header, include a parameter for authenticate method as specified below. Spring takes care of reading the Authorization value from HTTP Header and pass it as parameter.

```
@RequestHeader("Authorization") String authHeader
```

- The return type of this method should be Map<String, String>
- Include start and end logger in this method
- Include a debug log for displaying the authHeader parameter
- Create a new HashMap<String, String> and assign it to a map.
- Put a new item into the map with key as "token" and value as empty string.

SecurityConfig.java

- In the second configure method, include authenticate URL just after the countries URL defined earlier. Refer code below:

```
            .antMatchers("/countries").hasRole("USER")
            .antMatchers("/authenticate").hasAnyRole("USER", "ADMIN")
```

- The above configuration sets that users of both roles can access /authenticate URL.

Testing

curl command:

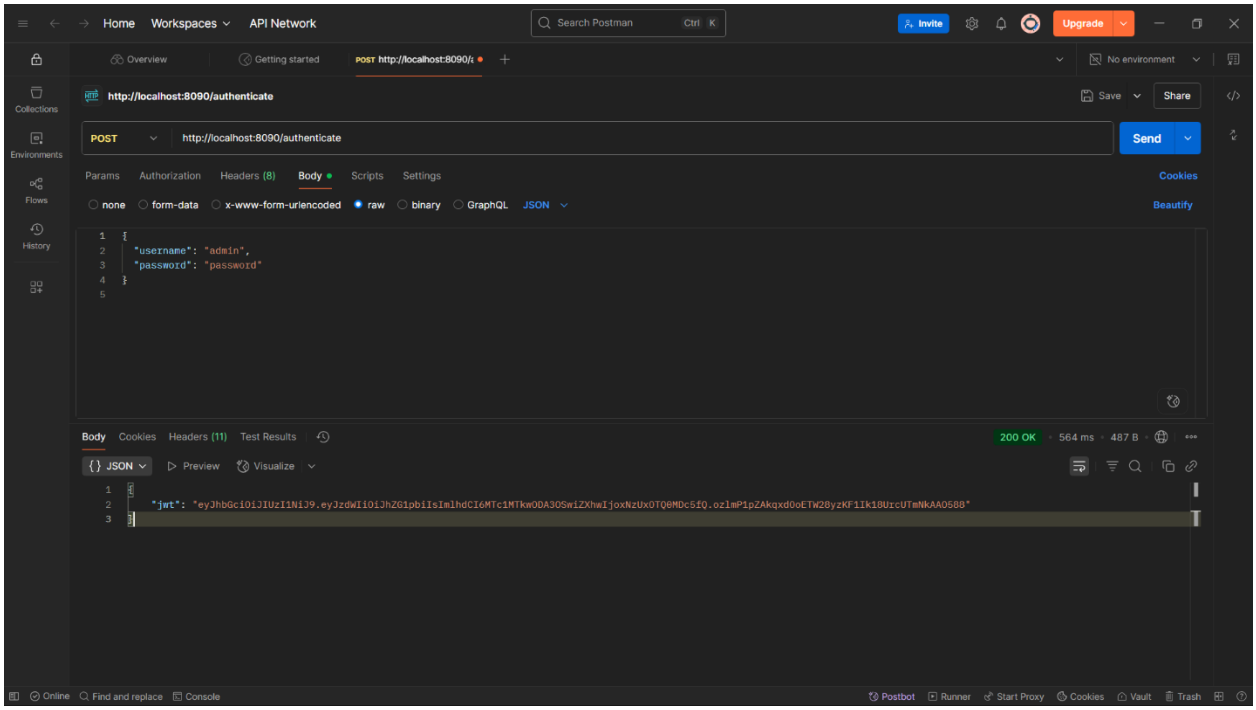
```
curl -s -u user:pwd http://localhost:8090/authenticate
```

Expected Response:

```
{"token":""}
```

Log verification:

Check if Authorization header value is displayed with "Basic" prefix and Base64 encoding of "user:pwd"



Read Authorization header and decode the username and password

Steps to read and decode header:

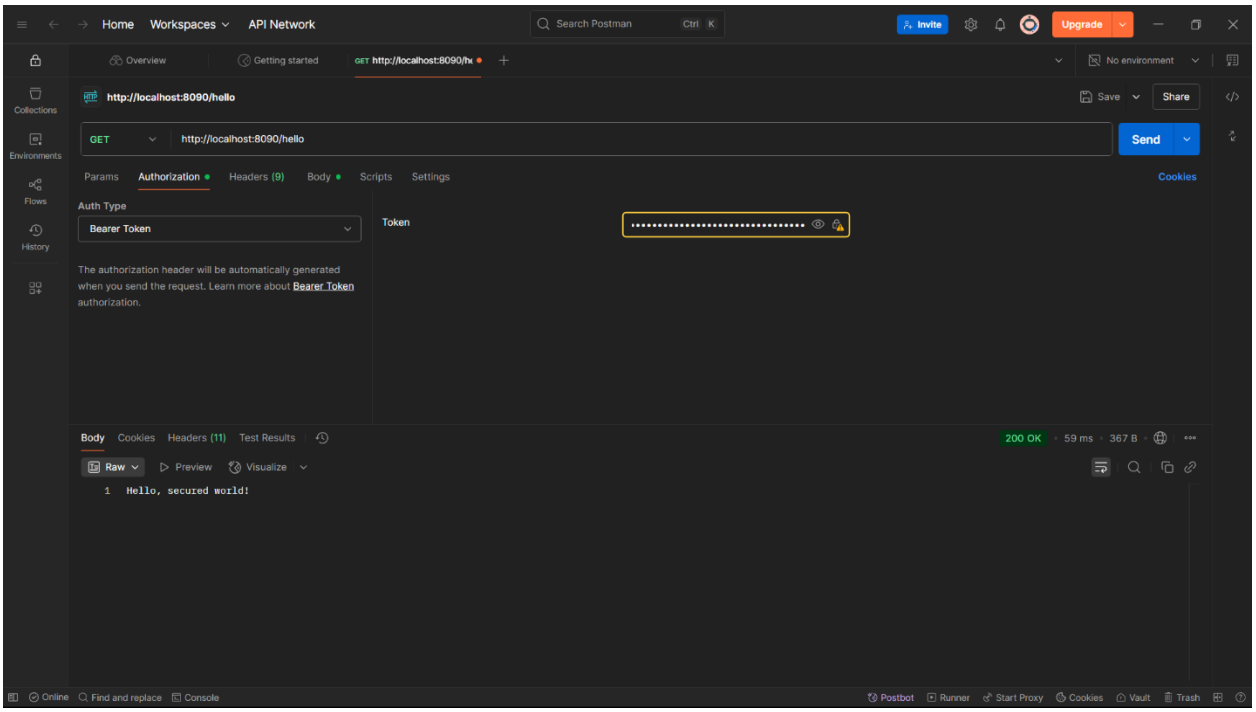
- Create a new private method in AuthenticationController with below method signature

```
private String getUser(String authHeader)
```

- Get the Base64 encoded text after "Basic "
- Decode it using the library available in Java 8 API. Refer code below.

```
Base64.getDecoder().decode(encodedCredentials)
```

- The above call returns a byte array, which can be passed as parameter to string constructor to convert to string.
- Get the text until colon on the string created in previous step to get the user
- Return the user obtained in previous step
- Include appropriate debug logs within this method
- Invoke the getUser() method from authenticate method
- Execute the curl command used in the previous step and check the logs if the use r information is obtained successfully.



Generate token based on the user

Steps to generate token:

- Include JWT library by including the following maven dependency.

```
 <dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.0</version>
</dependency>
```

- After inclusion in pom.xml, run the maven package command line and update the project in Eclipse. View the dependency tree and check if the library is added.
- Create a new method in AuthenticationController with below method signature:

```
private String generateJwt(String user)
```

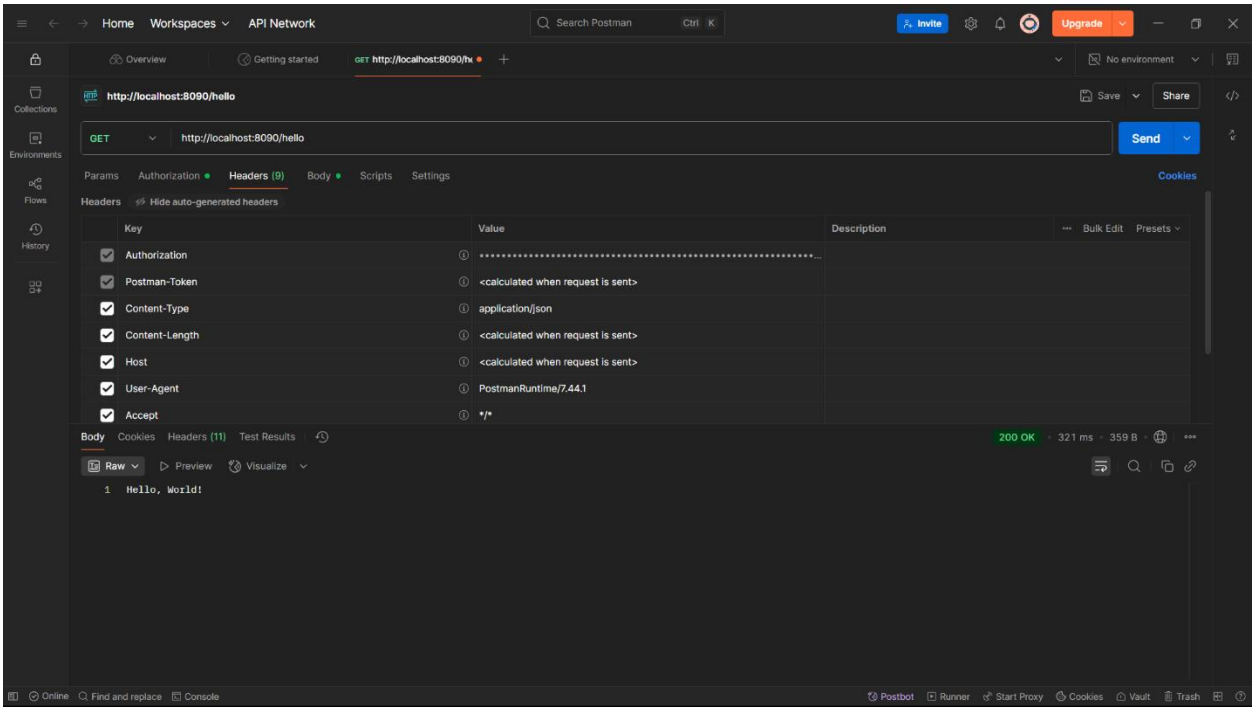
- Generate the token based on the code specified below.

```
JwtBuilder builder = Jwts.builder();
builder.setSubject(user);

// Set the token issue time as current time
builder.setIssuedAt(new Date());

// Set the token expiry as 20 minutes from now
builder.setExpiration(new Date((new Date()).getTime() + 1200000));
builder.signWith(SignatureAlgorithm.HS256, "secretkey");

String token = builder.compact();

return token;
```

- Import reference for the above code

```
import io.jsonwebtoken.JwtBuilder;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;
```

- Invoke this method from authenticate() method passing the user obtained from getUser() method.
- Add the token into the map using put method.
- Include appropriate logs
- Execute the curl command for authenticate and check if the generated token is returned.



Authorize based on JWT

Let us recollect the JWT Process

- Client sends username and password to server
- Servers validates credentials, creates token (JWT) and reponds it back
- Client attaches the token in the subsequent requests to server
- Server validates the token (JWT) on each client request

The points highlighted in blue above are already implemented.

Now all the application related requested coming in should send the token received and the server needs to be incorporate this.

So far, whatever we have implemented are service specific and we introduced respective controller methods, but now the requirment is the validate all the other services provided by this application to be validated for JWT, hence we cannot use a controller here. The ideal solution would be to use a filter as it can intercept all the requests received by this application.

Follow steps below to get this incorporated:

- Create a new class JwtAuthorizationFilter in package com.cognizant.springlearn.security
- This new class has to extend from BasicAuthenticationFilter. This parent class is available in spring security library.
- Include the below constructor that sets the authentication manager

```
    public JwtAuthorizationFilter(AuthenticationManager authenticationManager)
{

        super(authenticationManager);

        LOGGER.info("Start");

        LOGGER.debug("{}: ", authenticationManager);

    }
```

- Override the below method to check if Authorization header contains Bearer and initiates the validation. If the validation is successful, it sets the status in spring security as authenticated.

```
    @Override
```

```
    protected void doFilterInternal(HttpServletRequest req, HttpServletResponse
res,
            FilterChain chain) throws IOException, ServletException {
        LOGGER.info("Start");
        String header = req.getHeader("Authorization");
        LOGGER.debug(header);

        if (header == null || !header.startsWith("Bearer ")) {
            chain.doFilter(req, res);
            return;
        }
        UsernamePasswordAuthenticationToken authentication = getAuthentication(
req);

        SecurityContextHolder.getContext().setAuthentication(authentication);
        chain.doFilter(req, res);
        LOGGER.info("End");
    }
```

- The getAuthentication() method invoked in the above code has to be implemented within this same class as private method. Refer code below.

```
    private UsernamePasswordAuthenticationToken getAuthentication(HttpServletRe
quest request) {
        String token = request.getHeader("Authorization");
        if (token != null) {
            // parse the token.
            Jws<Claims> jws;
            try {
                jws = Jwts.parser()
                        .setSigningKey("secretkey")
                        .parseClaimsJws(token.replace("Bearer ", ""));
                String user = jws.getBody().getSubject();
                LOGGER.debug(user);
                if (user != null) {
```

```
                    return new UsernamePasswordAuthenticationToken(user, null,
new ArrayList<>());
                }
            } catch (JwtException ex) {
                return null;
            }
            return null;
        }
        return null;
    }
```

- Necessary imports for the above code

```
import java.io.IOException;
import java.util.ArrayList;


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.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticati
onToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.web.authentication.www.BasicAuthenticationF
ilter;


import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jws;
import io.jsonwebtoken.JwtException;
import io.jsonwebtoken.Jwts;
```

- Now the final step is to configure the security to use the above specified filter. Modify code of 2nd configure method in SecurityConfig class.

```
    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.csrf().disable().httpBasic().and()
            .authorizeRequests()
            //.antMatchers("/countries").hasRole("USER")
            .antMatchers("/authenticate").hasAnyRole("USER", "ADMIN")
            .anyRequest().authenticated()
            .and()
            .addFilter(new JwtAuthorizationFilter(authenticationManager()));
    }
```

- First line retains the HTTP Basic authentication
- The last three lines includes the new filter to validate JWT

Test

Execute below command to create a fresh token. Copy the token generated to be used for the next command.

```
curl -s -u user:pwd http://localhost:8090/authenticate
```

Execute below command to invoke any service of the application with JWT. Notice how authorization header is added with bearer and the token in request.

```
curl -s -H "Authorization: Bearer REPLACE_TOKEN_HERE" http://localhost:8090/countries
```

Execute the above command with minor modification the token and check if Unauthorized reponse is received.

Submitted BY:

Lingaraj Nayak

Superset ID : 6387607