JWT HandsOn

**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

**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 user 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>

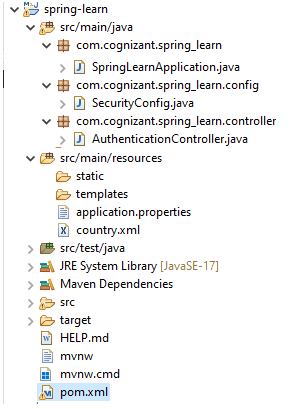
        </dependency>

* 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 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.

**Project Structure:**

****

**SpringLearnApplication.java**

package com.cognizant.spring\_learn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnApplication {

public static void main(String[] args) {

SpringApplication.run(SpringLearnApplication.class, args);

}

}

**AuthenticationController.java  
  
package** com.cognizant.spring\_learn.controller;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

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

**import** java.util.HashMap;

**import** java.util.Map;

**import** io.jsonwebtoken.JwtBuilder;

**import** io.jsonwebtoken.Jwts;

**import** io.jsonwebtoken.SignatureAlgorithm;

**import** java.util.Date;

@RestController

**public** **class** AuthenticationController {

**private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(AuthenticationController.**class**);

@GetMapping("/authenticate")

**public** Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {

***LOGGER***.info("START - authenticate()");

***LOGGER***.debug("Authorization Header: {}", authHeader);

Map<String, String> map = **new** HashMap<>();

String user = getUser(authHeader);

String token = generateJwt(user);

map.put("token", token);

***LOGGER***.info("END - authenticate()");

**return** map;

}

**private** String getUser(String authHeader) {

// authHeader format: "Basic base64encoded(username:password)"

String base64Credentials = authHeader.substring("Basic ".length());

**byte**[] decodedBytes = java.util.Base64.*getDecoder*().decode(base64Credentials);

String decodedCredentials = **new** String(decodedBytes);

// decodedCredentials format: "username:password"

String username = decodedCredentials.split(":")[0];

***LOGGER***.debug("Decoded username: {}", username);

**return** username;

}

**private** String generateJwt(String user) {

JwtBuilder builder = Jwts.*builder*();

builder.setSubject(user);

builder.setIssuedAt(**new** Date());

// Token expires 20 minutes from now

builder.setExpiration(**new** Date(System.*currentTimeMillis*() + 20 \* 60 \* 1000));

builder.signWith(SignatureAlgorithm.***HS256***, "secretkey");

**return** builder.compact();

}

}

**SecurityConfig.java**

**package** com.cognizant.spring\_learn.config;

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

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

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

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

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

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

**import** org.springframework.security.provisioning.InMemoryUserDetailsManager;

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

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

**import** **static** org.springframework.security.config.Customizer.*withDefaults*;

@Configuration

**public** **class** SecurityConfig {

@Bean

**public** SecurityFilterChain filterChain(HttpSecurity http) **throws** Exception {

**return** http

.httpBasic(*withDefaults*())

.authorizeHttpRequests(auth -> auth

.requestMatchers("/countries").hasRole("USER")

.requestMatchers("/authenticate").hasAnyRole("USER", "ADMIN")

.anyRequest().authenticated()

)

.csrf(csrf -> csrf.disable())

.build();

}

@Bean

**public** UserDetailsService userDetailsService() {

PasswordEncoder encoder = *passwordEncoder*();

**return** **new** InMemoryUserDetailsManager(

User.*withUsername*("user")

.password(encoder.encode("pwd"))

.roles("USER")

.build(),

User.*withUsername*("admin")

.password(encoder.encode("pwd"))

.roles("ADMIN")

.build()

);

}

@Bean

**public** **static** PasswordEncoder passwordEncoder() {

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

}

}

**pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>3.5.3</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.cognizant</groupId>

<artifactId>spring-learn</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>spring-learn</name>

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

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<!-- Spring Security -->

<dependency>

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

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

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.0</version>

</dependency>

<dependency>

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

<artifactId>jaxb-api</artifactId>

<version>2.3.1</version>

</dependency>

<dependency>

<groupId>org.glassfish.jaxb</groupId>

<artifactId>jaxb-runtime</artifactId>

<version>2.3.1</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

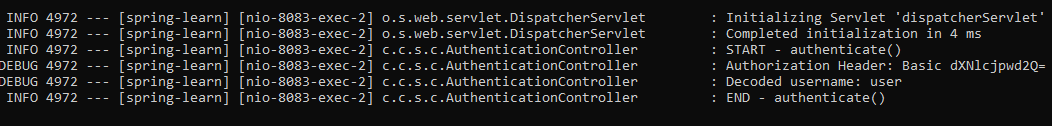
</plugin>

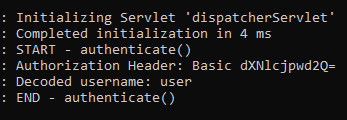
</plugins>

</build>

</project>

**Output:**

**Console logs**



**Token Generated:**

