**WEEK-04**

**HANDS-ON SOLUTIONS**

**5. JWT-HANDSON**

**EXERCISE:**

**CREATE AUTHENTICATION SERVICE THAT RETURNS JWT**

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.eyJzdWIiOiJ1c2VyIiwiaWF0IjoxNTcwMzc5NDc0LCJleHAiOjE1Nz  
AzODA2NzR9.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 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 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.

**SOLUTION:**

In Week 04 hands-on exercise, we implemented an authentication service in a Spring Boot application that returns a JSON Web Token (JWT) upon validating user credentials.

The service exposes a REST endpoint at /authenticate handled by the AuthenticationController.

This controller reads the Authorization header from incoming requests using HTTP Basic Authentication.

The credentials are extracted by decoding the Base64-encoded header, then parsing to retrieve the username.

We logged these values for debugging and verification purposes. This process ensures that user identity is correctly obtained from each request before token generation.

Using the JJWT library, we created a JWT token for the authenticated user with the username as the subject.

The token includes an issue timestamp and an expiration set to 20 minutes from issuance.

It is signed using the HS256 algorithm with a static secret key "secretkey". The generated JWT is then returned as a JSON response in the format { "token": "<JWT>" }.

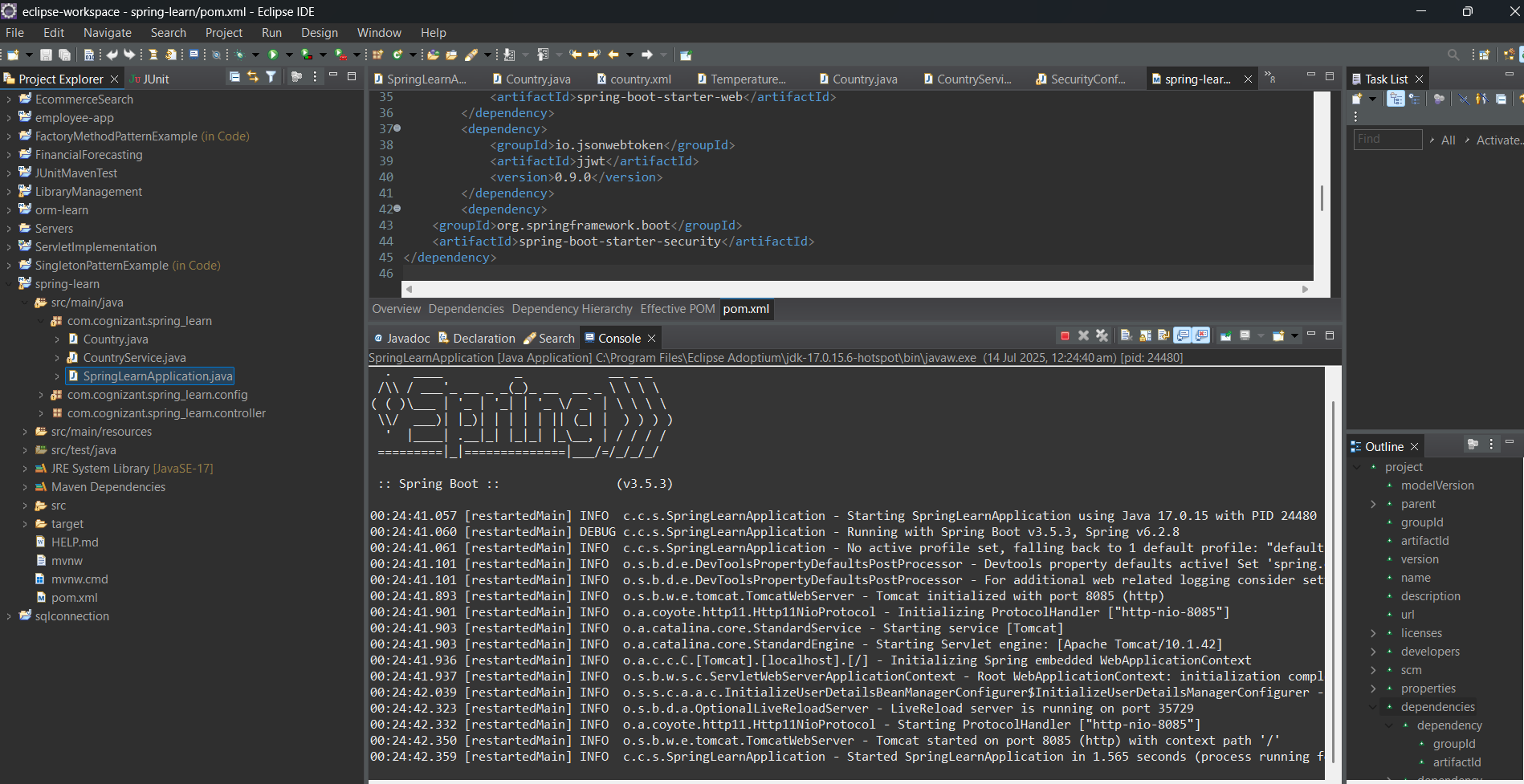
The security configuration in SecurityConfig defined two in-memory users—user with password pwd (role USER) and admin with password adminpwd (role ADMIN).

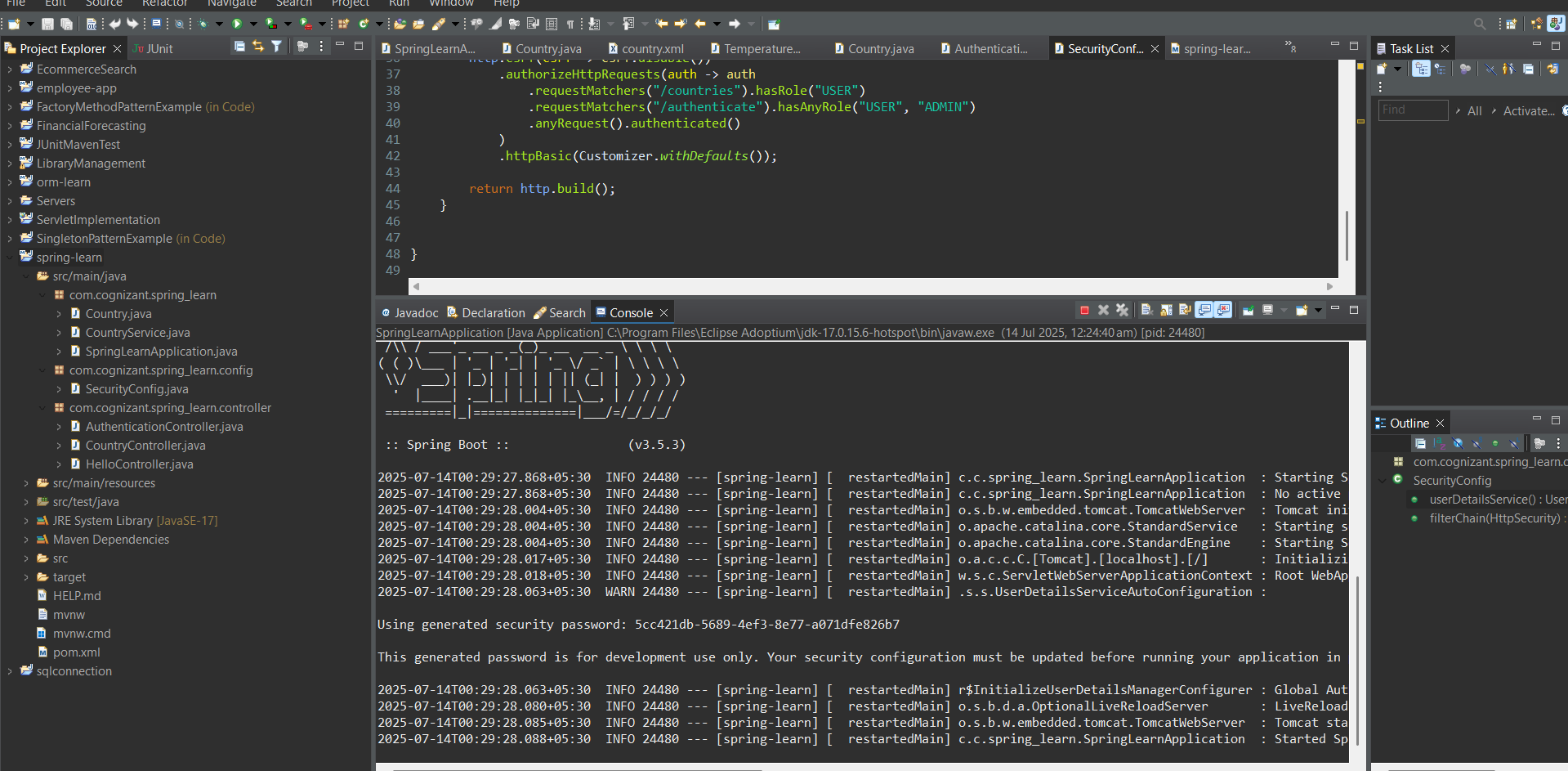
The /authenticate endpoint was configured to allow access to users with either USER or ADMIN roles, while the /countries endpoint remained restricted to USER role only.

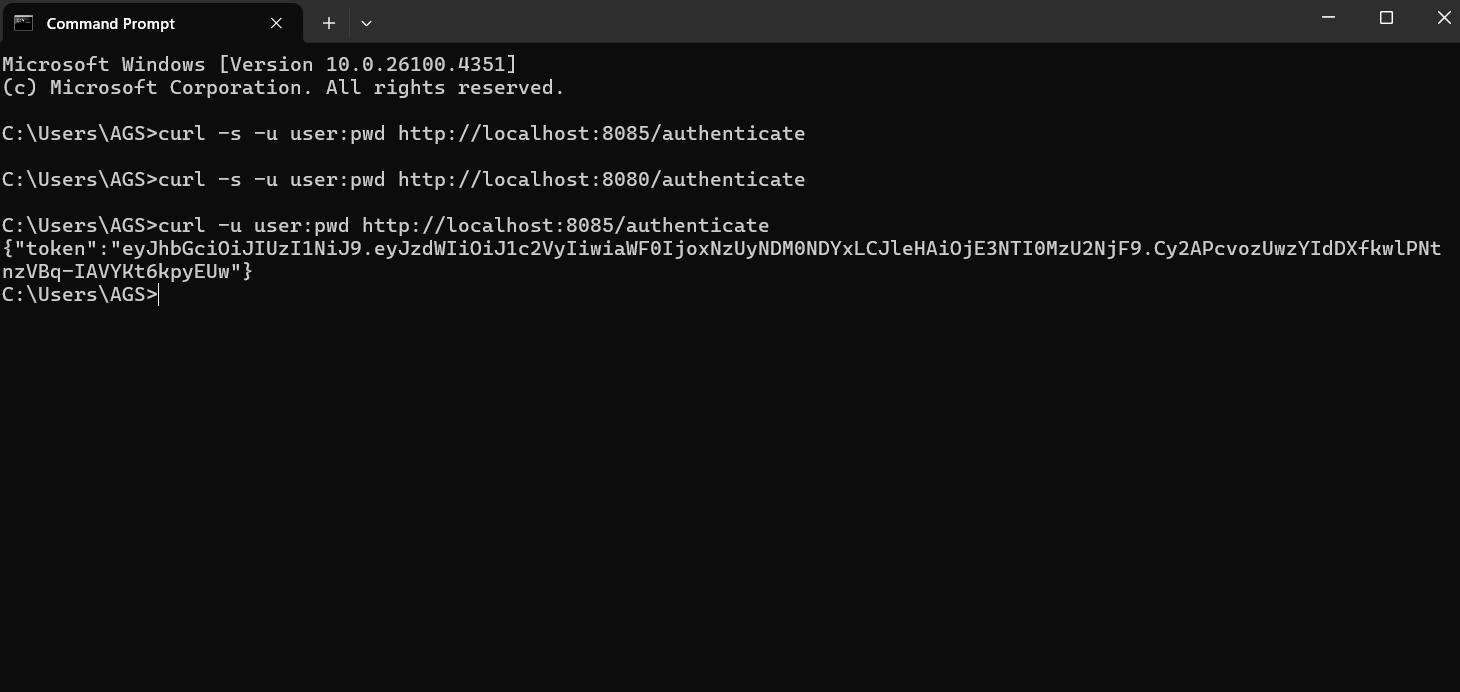
The implementation was successfully tested by executing the curl command curl -s -u user:pwd http://localhost:8085/authenticate, which returned a valid JWT token as expected.

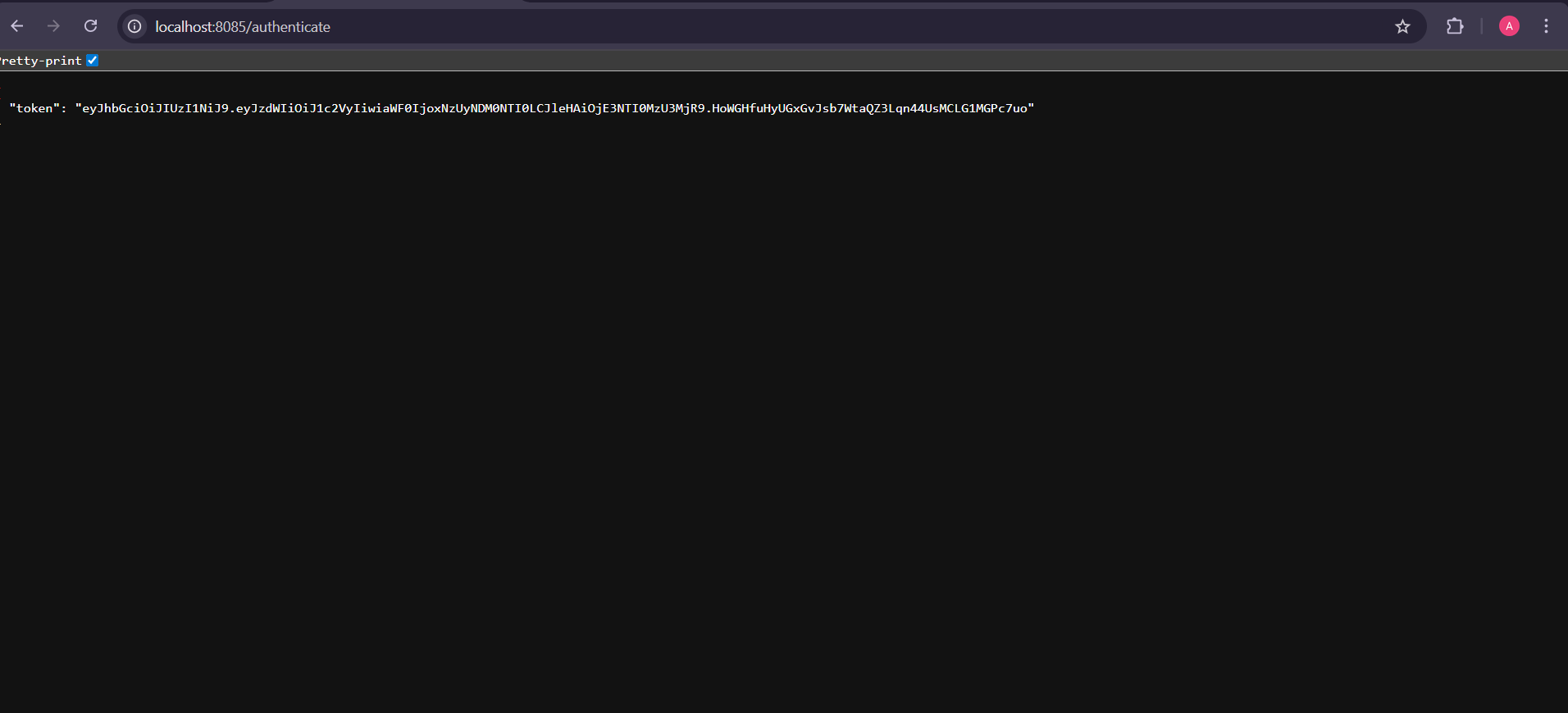
Throughout development, logging statements confirmed that the Authorization header was correctly received, the username was decoded properly, and the JWT was generated and returned without errors. This exercise demonstrates the core steps of JWT authentication—creating a controller, decoding credentials, generating tokens, and securing endpoints—providing a solid foundation for building secure authentication services in Spring Boot.

**OUTPUT:**



****

****

****