**WEEK – 04**

**Spring REST using Spring Boot 3**

**Superset ID: 6410372**

**JWT-handson**

**Exercise 6:**

**Create authentication service that returns JWT**

**SOLUTION:**

***SpringLearnApplication.java:***

package com.cognizant.springlearn;

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);

}

}

***Purpose:***

* Entry point of the Spring Boot application.
* Boots up the embedded server and loads Spring context.

***SecurityConfig.java:***

package com.cognizant.springlearn.config;

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.configuration.AuthenticationConfiguration;

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

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

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

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

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

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

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

import org.springframework.security.web.SecurityFilterChain;

@Configuration

public class SecurityConfig {

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeHttpRequests()

.requestMatchers("/authenticate").permitAll()

.anyRequest().authenticated();

return http.build();

}

@Bean

public AuthenticationManager authManager(AuthenticationConfiguration config) throws Exception {

return config.getAuthenticationManager();

}

@Bean

public UserDetailsService userDetailsService() {

UserDetails user = User.withUsername("admin")

.password("password")

.roles("USER")

.build();

return new InMemoryUserDetailsManager(user);

}

@Bean

public PasswordEncoder passwordEncoder() {

return NoOpPasswordEncoder.getInstance(); // For testing only

}

}

***PURPOSE:***

* Configures Spring Security to:
* Disable CSRF.
* Allow unauthenticated access to /authenticate.
* Require authentication for other endpoints.
* Defines an in-memory user (admin / password) for testing.
* Sets up password encoder and authentication manager.

***AuthenticationController.java:***

package com.cognizant.springlearn.controller;

import com.cognizant.springlearn.service.JwtService;

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

import org.springframework.http.ResponseEntity;

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

import jakarta.servlet.http.HttpServletRequest;

import java.util.Base64;

import java.util.StringTokenizer;

@RestController

public class AuthenticationController {

@Autowired

private JwtService jwtService;

@GetMapping("/authenticate")

public ResponseEntity<?> authenticate(HttpServletRequest request) {

System.out.println(">>> /authenticate called");

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

System.out.println(">>> Authorization header: " + authHeader);

if (authHeader != null && authHeader.startsWith("Basic ")) {

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

byte[] credDecoded = Base64.getDecoder().decode(base64Credentials);

String credentials = new String(credDecoded);

StringTokenizer tokenizer = new StringTokenizer(credentials, ":");

String username = tokenizer.nextToken();

String password = tokenizer.nextToken();

System.out.println(">>> Decoded username: " + username);

System.out.println(">>> Decoded password: " + password);

if (username.equals("user") && password.equals("pwd")) {

String token = jwtService.generateToken(username);

System.out.println(">>> Generated token: " + token);

return ResponseEntity.ok("{\"token\":\"" + token + "\"}");

} else {

System.out.println(">>> Invalid credentials provided");

}

} else {

System.out.println(">>> Authorization header missing or malformed");

}

return ResponseEntity.status(401).body("Invalid credentials");

}

}

**Purpose:**

* Exposes /authenticate endpoint.
* Extracts Basic Auth credentials from HTTP header.
* Verifies credentials (user / pwd) and generates a JWT token if valid.
* Returns the token as JSON.

***JwtService.java:***

package com.cognizant.springlearn.service;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.security.Keys;

import org.springframework.stereotype.Service;

import java.security.Key;

import java.util.Date;

@Service

public class JwtService {

private static final Key key = Keys.secretKeyFor(SignatureAlgorithm.HS256);

public String generateToken(String username) {

return Jwts.builder()

.setSubject(username)

.setIssuer("spring-learn")

.setIssuedAt(new Date())

.setExpiration(new Date(System.currentTimeMillis() + 86400000)) // 1 day

.signWith(key)

.compact();

}

}

***Purpose:***

* Generates a signed JWT token for authenticated users.
* Sets expiration to 24 hours.

***AuthRequest.java:***

package com.cognizant.springlearn.model;

public class AuthRequest {

private String username;

private String password;

// Getters and setters

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;

}

}

***Purpose:***

* Model class to hold username and password in request body (if you prefer JSON input).

***AuthResponse.java:***

package com.cognizant.springlearn.model;

public class AuthResponse {

private String token;

public AuthResponse(String token) {

this.token = token;

}

public String getToken() {

return token;

}

}

***Purpose:***

* Model class to wrap the generated token in a response object.

***SpringLearnApplicationTests.java:***

package com.cognizant.springlearn;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

class SpringLearnApplicationTests {

@Test

void contextLoads() {

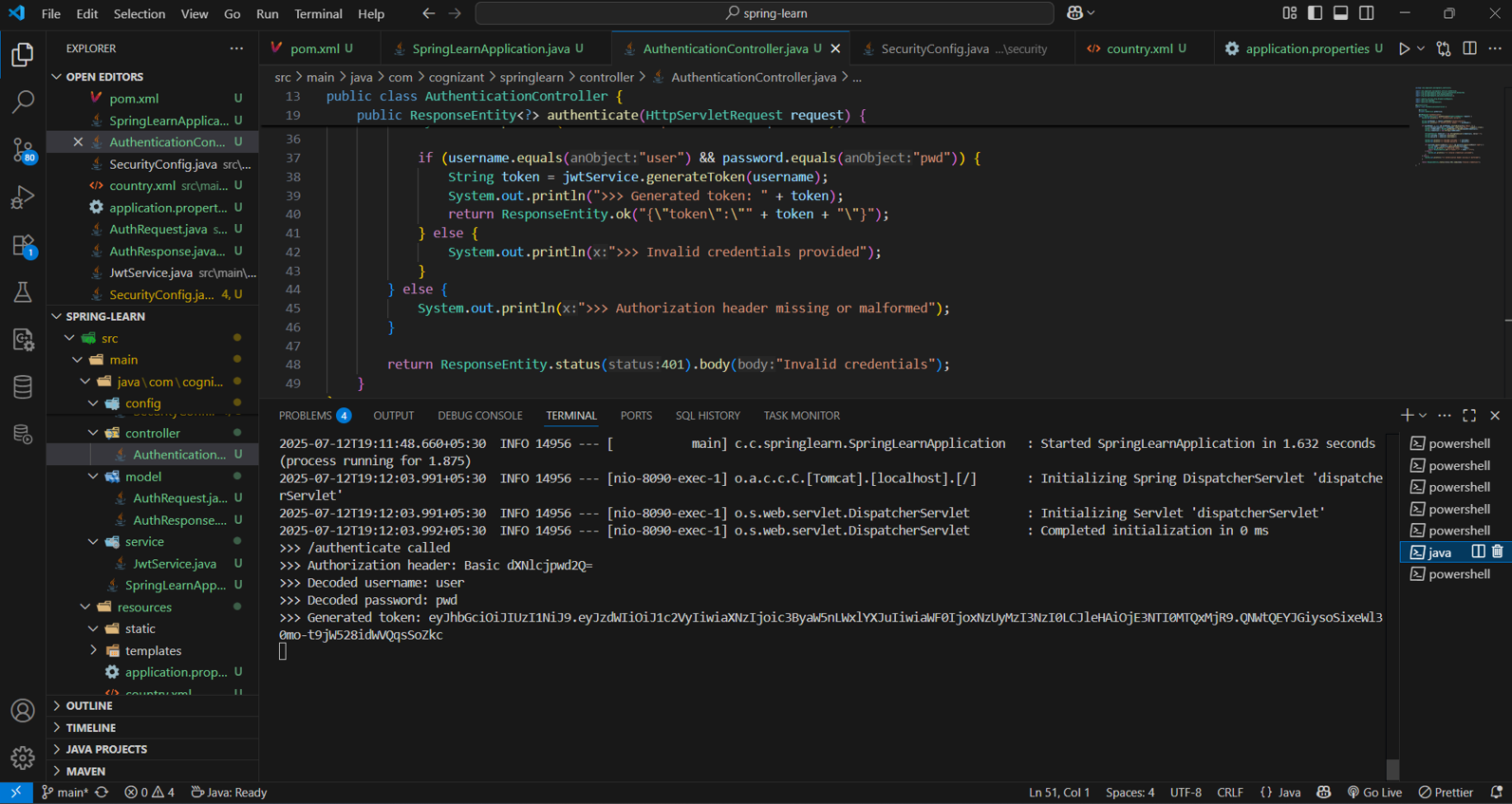
}

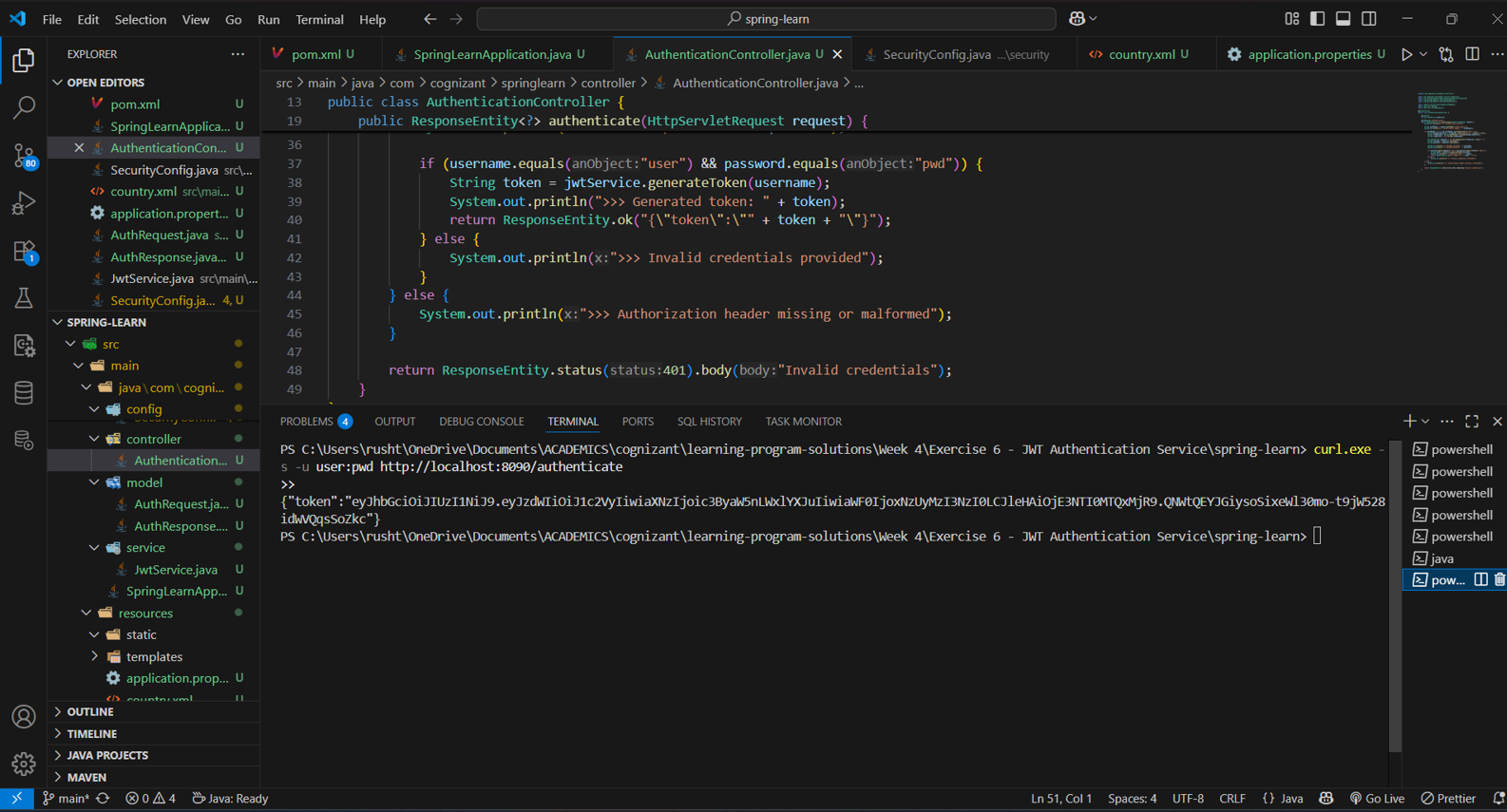
}

***Purpose:***

* Verifies the Spring Boot application context loads successfully.

***OUTPUT:***

******

******

***EXPLANATION:***

* I configured Spring Security to allow unauthenticated access to /authenticate and protect other endpoints.
* I built a controller that decodes Basic Auth headers, validates credentials, and generates JWT tokens.
* I created service classes and models to manage authentication and represent tokens cleanly.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**