# ****Create authentication service that returns JWT****

## ****1. Introduction****

JWT (JSON Web Token) is a compact and self-contained way for securely transmitting information between parties. This project implements a basic Spring Boot application that uses JWTs for user authentication and securing REST APIs.

## ****2. Objective****

The goal is to:

* Authenticate users with a predefined username and password.
* Generate a JWT token upon successful authentication.
* Protect sensitive API endpoints using JWT validation.
* Respond with 401 Unauthorized when a token is missing or invalid.

## ****3. Project Setup Summary****

* **Port Used**: 8080
* **Username**: user
* **Password**: pwd
* **Authentication Endpoint**: GET /authenticate
* **Protected Endpoint**: GET /hello

## ****4. Key Components & Class Descriptions****

### ****JwtAuthApplication.java****

**Purpose**: Entry point of the Spring Boot application.

package com.cognizant.spring\_learn;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class JwtAuthApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(JwtAuthApplication.class, args);  
 }  
}

### ****SecurityConfig.java****

**Purpose**: Configures security settings including endpoint access rules and authentication filter.

package com.cognizant.spring\_learn.config;  
  
import com.cognizant.spring\_learn.filter.JwtRequestFilter;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.config.Customizer;  
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.web.SecurityFilterChain;  
  
@Configuration  
public class SecurityConfig {  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http, JwtRequestFilter jwtFilter) throws Exception {  
 http  
 .csrf(csrf -> csrf.disable())  
 .authorizeHttpRequests(auth -> auth  
 .requestMatchers("/authenticate").permitAll()  
 .anyRequest().authenticated()  
 )  
 .addFilterBefore(jwtFilter, org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter.class)  
 .httpBasic(Customizer.*withDefaults*());  
  
 return http.build();  
 }  
  
 @Bean  
 public UserDetailsService userDetailsService() {  
 UserDetails user = User.*withUsername*("user")  
 .password("{noop}pwd")   
 .roles("USER")  
 .build();  
 return new InMemoryUserDetailsManager(user);  
 }  
}

### ****AuthService.java****

**Purpose**: Authenticates credentials and returns a JWT.

package com.cognizant.spring\_learn.service;  
  
import com.cognizant.spring\_learn.util.JwtUtil;  
import org.springframework.stereotype.Service;  
  
@Service  
public class AuthService {  
 public String authenticate(String username, String password) {  
 if ("user".equals(username) && "pwd".equals(password)) {  
 return JwtUtil.*generateToken*(username);  
 }  
 return null;  
 }  
}

### ****JwtUtil.java****

**Purpose**: Generates JWT using the secret key.

package com.cognizant.spring\_learn.util;  
  
import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.SignatureAlgorithm;  
import io.jsonwebtoken.security.Keys;   
  
import java.security.Key;  
import java.util.Date;  
  
public class JwtUtil {  
  
  
 private static final Key *SECRET\_KEY* = Keys.*hmacShaKeyFor*("MySecretKeyMySecretKeyMySecretKey".getBytes());  
 private static final long *EXPIRATION\_TIME* = 60 \* 60 \* 1000; // 1 hour  
  
 public static String generateToken(String username) {  
 return Jwts.*builder*()  
 .setSubject(username)   
 .setIssuedAt(new Date())  
 .setExpiration(new Date(System.*currentTimeMillis*() + *EXPIRATION\_TIME*))  
 .signWith(*SECRET\_KEY*, SignatureAlgorithm.*HS256*)  
 .compact();  
 }  
}

### ****JwtRequestFilter.java****

**Purpose**: Validates JWT for every request except /authenticate.

package com.cognizant.spring\_learn.filter;  
  
import io.jsonwebtoken.Claims;  
import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.security.Keys;  
import jakarta.servlet.\*;  
import jakarta.servlet.http.HttpServletRequest;  
import jakarta.servlet.http.HttpServletResponse;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.stereotype.Component;  
  
import java.io.IOException;  
import java.security.Key;  
import java.util.List;  
  
@Component  
public class JwtRequestFilter implements Filter {  
  
 private static final Key *SECRET\_KEY* = Keys.*hmacShaKeyFor*("MySecretKeyMySecretKeyMySecretKey".getBytes());  
  
 @Override  
 public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)  
 throws IOException, ServletException {  
  
 HttpServletRequest httpRequest = (HttpServletRequest) request;  
 HttpServletResponse httpResponse = (HttpServletResponse) response;  
  
 String path = httpRequest.getRequestURI();  
 System.*out*.println("Requested URI: " + path);  
  
 if (path.equals("/authenticate")) {  
 System.*out*.println("Skipping JWT check for /authenticate");  
 chain.doFilter(request, response);  
 return;  
 }  
  
 String authHeader = httpRequest.getHeader("Authorization");  
 System.*out*.println("Authorization Header: " + authHeader);  
  
 if (authHeader != null && authHeader.startsWith("Bearer ")) {  
 String jwt = authHeader.substring(7);  
 try {  
 Claims claims = Jwts  
 .*parserBuilder*()  
 .setSigningKey(*SECRET\_KEY*)  
 .build()  
 .parseClaimsJws(jwt)  
 .getBody();  
  
 String username = claims.getSubject();  
 System.*out*.println("Token valid. Username: " + username);  
  
 httpRequest.setAttribute("username", username);  
 UsernamePasswordAuthenticationToken authentication =  
 new UsernamePasswordAuthenticationToken(  
 username, null, List.*of*(new SimpleGrantedAuthority("ROLE\_USER"))  
 );  
 SecurityContextHolder.*getContext*().setAuthentication(authentication);  
  
 chain.doFilter(request, response);   
 return;  
  
 } catch (Exception e) {  
 System.*out*.println("Token parsing failed: " + e.getMessage());  
 httpResponse.sendError(HttpServletResponse.*SC\_UNAUTHORIZED*, "Invalid or expired token");  
 return;  
 }  
 }  
  
 System.*out*.println("Missing or invalid Authorization header.");  
 httpResponse.sendError(HttpServletResponse.*SC\_UNAUTHORIZED*, "Missing or invalid Authorization header");  
 }  
}

### ****HelloController.java****

**Purpose**: A secured endpoint that returns a welcome message if JWT is valid.

package com.cognizant.spring\_learn.controller;  
  
import jakarta.servlet.http.HttpServletRequest;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class HelloController {  
  
  
  
 @GetMapping("/hello")  
 public String hello(HttpServletRequest request) {  
 String username = (String) request.getAttribute("username");  
 if (username == null) {  
 return "No username in request"; // helpful debug  
 }  
 return "Welcome, " + username + "! You have accessed a protected resource.";  
 }  
  
  
}

### ****AuthenticationController.java****

**Purpose**: Handles the /authenticate request and returns a JWT.

package com.cognizant.spring\_learn.controller;  
  
import com.cognizant.spring\_learn.service.AuthService;  
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;  
  
@RestController  
public class AuthenticationController {  
  
 @Autowired  
 private AuthService authService;  
  
 @GetMapping("/authenticate")  
 public ResponseEntity<?> authenticate(HttpServletRequest request) {  
 String authHeader = request.getHeader("Authorization");  
  
 if (authHeader == null || !authHeader.startsWith("Basic ")) {  
 return ResponseEntity.*status*(401).body("Missing or invalid Authorization header");  
 }  
  
 try {  
  
 String base64Credentials = authHeader.substring("Basic ".length());  
 String credentials = new String(Base64.*getDecoder*().decode(base64Credentials));  
 String[] values = credentials.split(":", 2);  
  
 if (values.length != 2) {  
 return ResponseEntity.*status*(400).body("Invalid basic auth format");  
 }  
  
 String username = values[0];  
 String password = values[1];  
  
 String token = authService.authenticate(username, password);  
 if (token != null) {  
 return ResponseEntity.*ok*().body("{\"token\":\"" + token + "\"}");  
 } else {  
 return ResponseEntity.*status*(401).body("Invalid credentials");  
 }  
  
 } catch (IllegalArgumentException e) {  
 return ResponseEntity.*status*(400).body("Failed to decode credentials");  
 } catch (Exception e) {  
 e.printStackTrace();  
 return ResponseEntity.*status*(500).body("Internal Server Error");  
 }  
 }  
}

## ****5. Conclusion****

In this project, we successfully implemented a **JWT-based authentication system** using Spring Boot. The authentication service was designed to accept credentials and respond with a secure JWT, which is then used to access protected resources.

To verify the functionality, the following test can be performed using a simple curl command:

### ****Authentication Request****

**curl -s -u user:pwd http://localhost:8080/authenticate**

### ****Response****

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

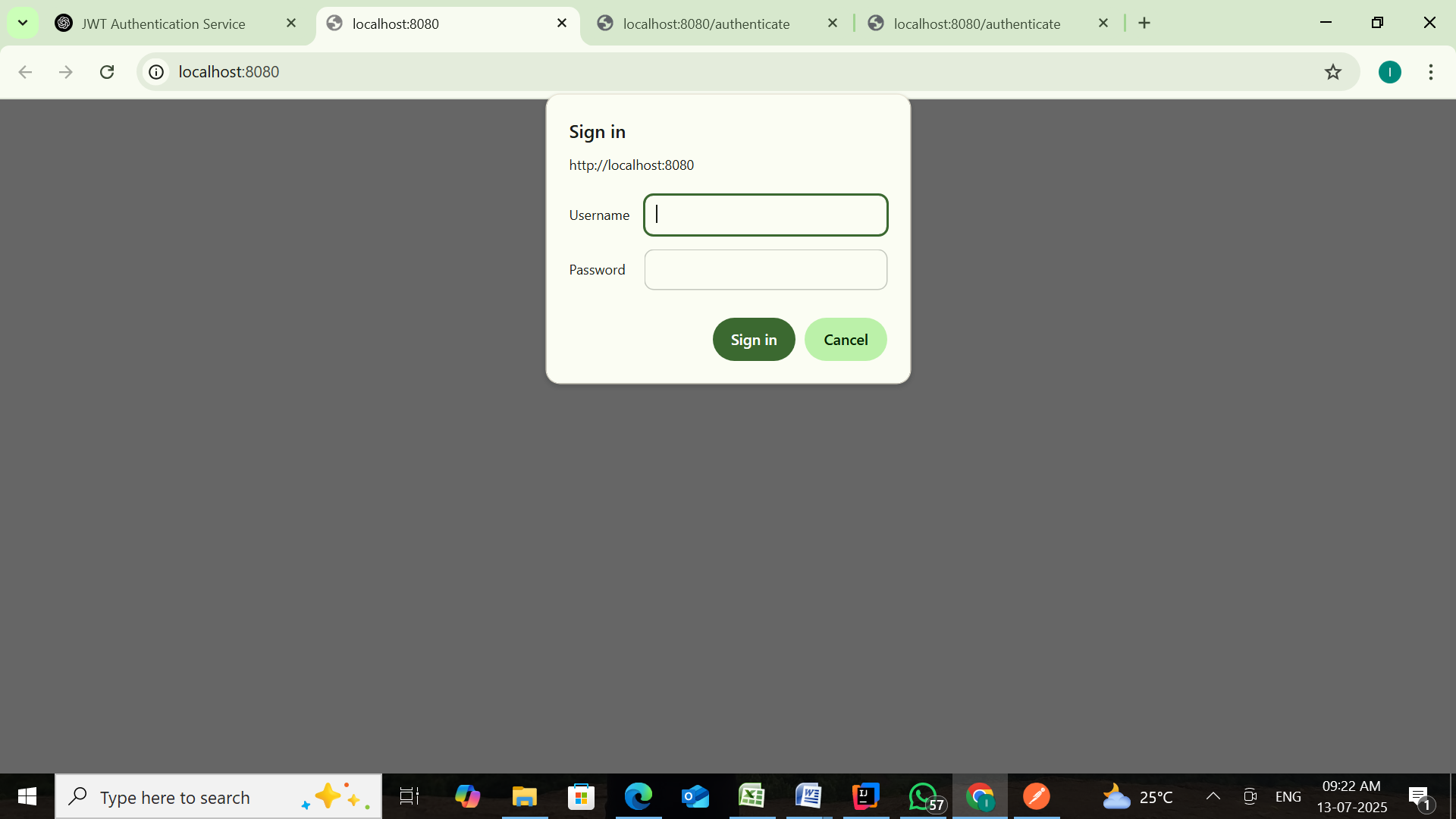
This token can then be passed in the Authorization header as a **Bearer Token** to access secure endpoints like /hello.

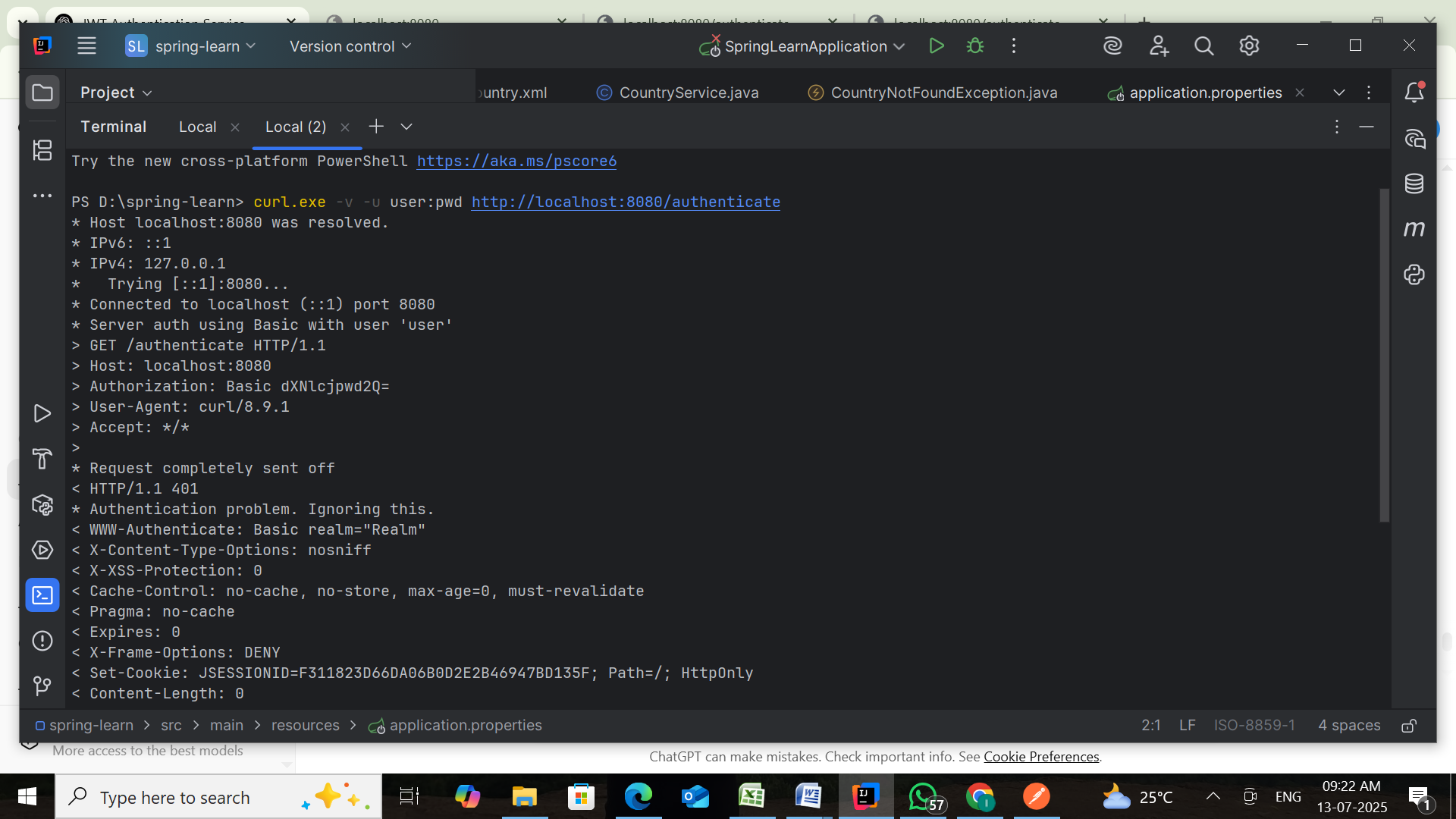
With this setup:

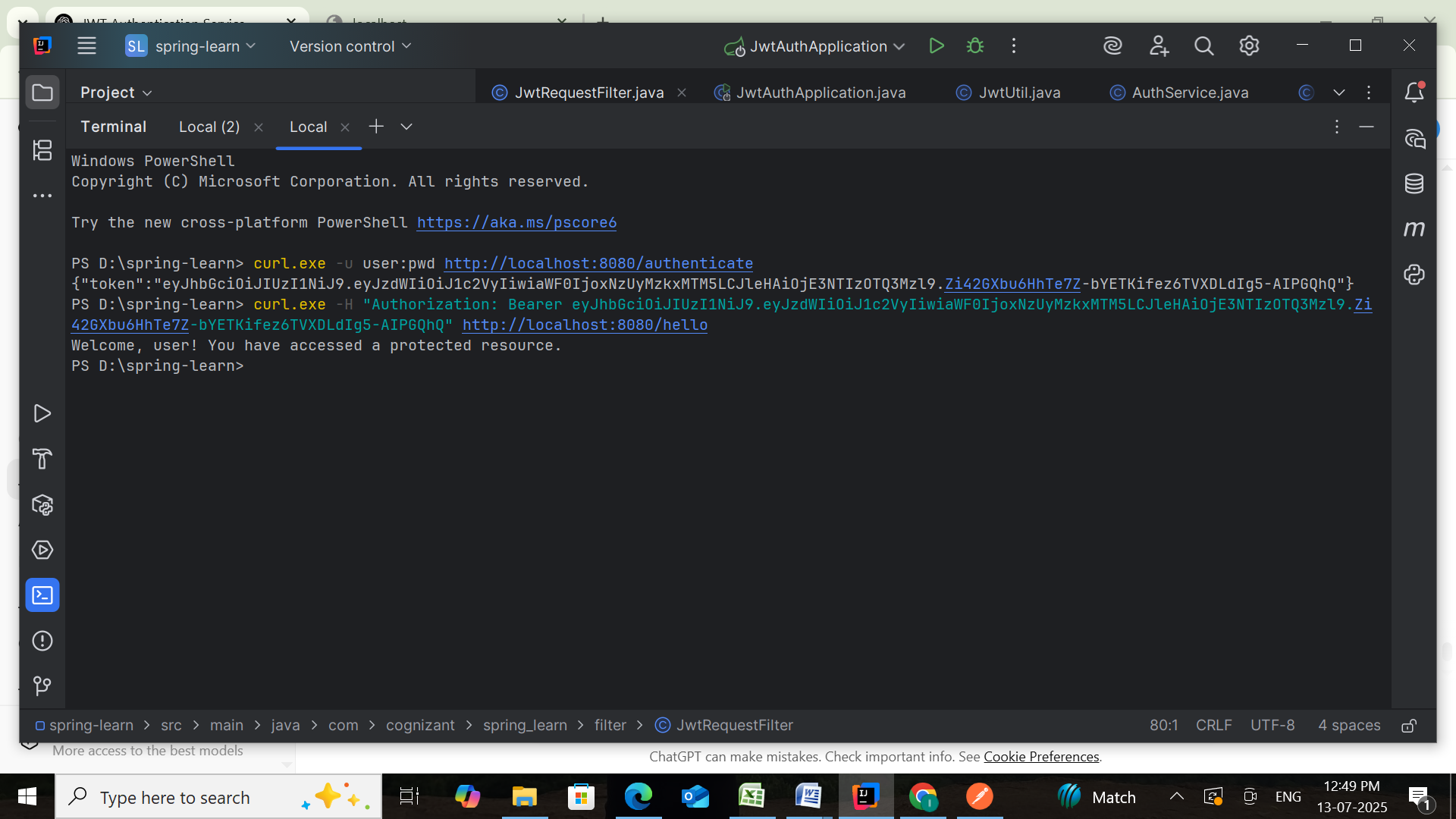
* The application avoids session-based state.
* Security is handled in a stateless, scalable manner.
* Tokens are verified with a secret key before granting access.

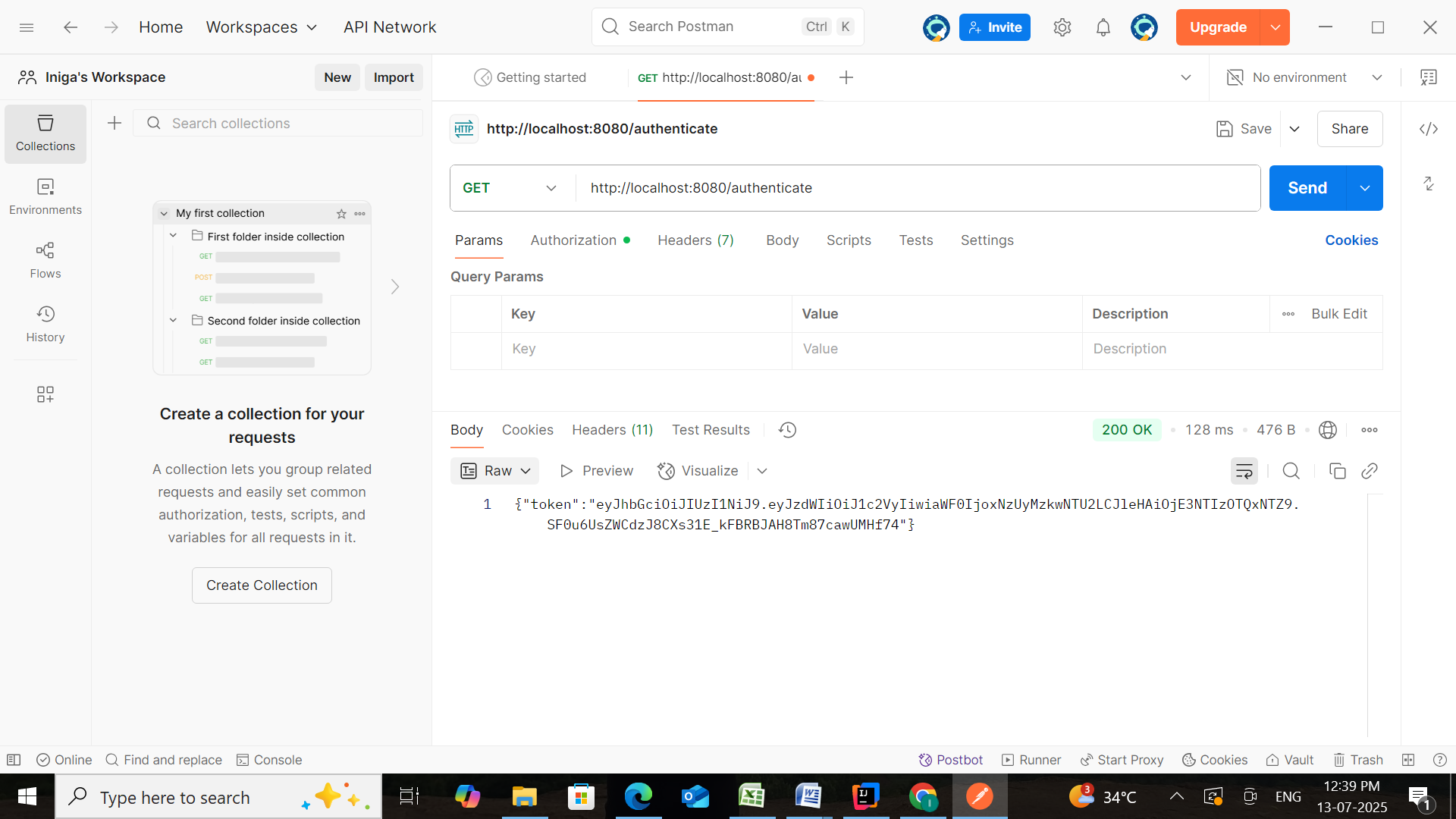
This lays a strong foundation for further enhancements like role-based access, database user integration, and refresh tokens.

**Outputs:**

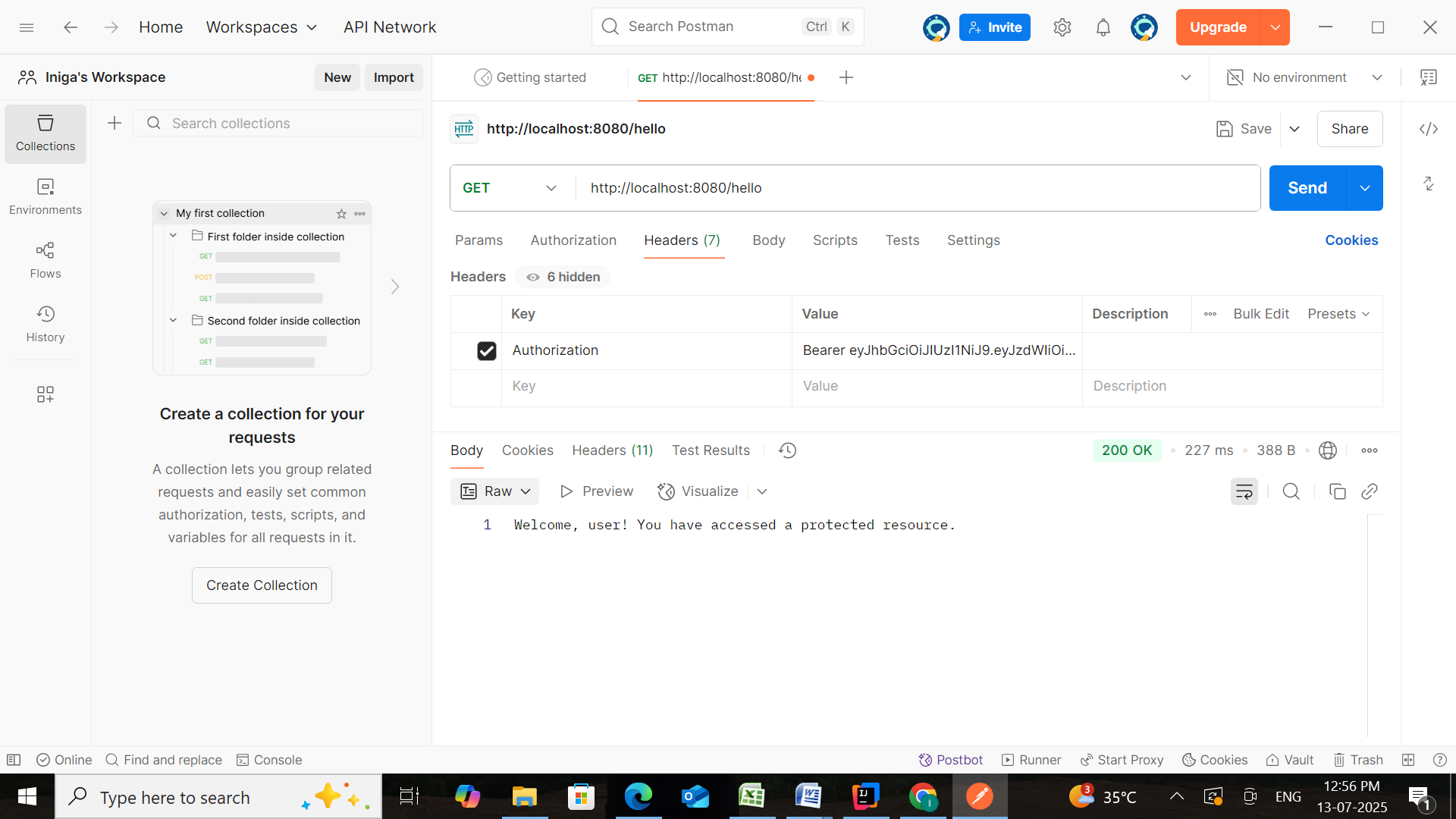


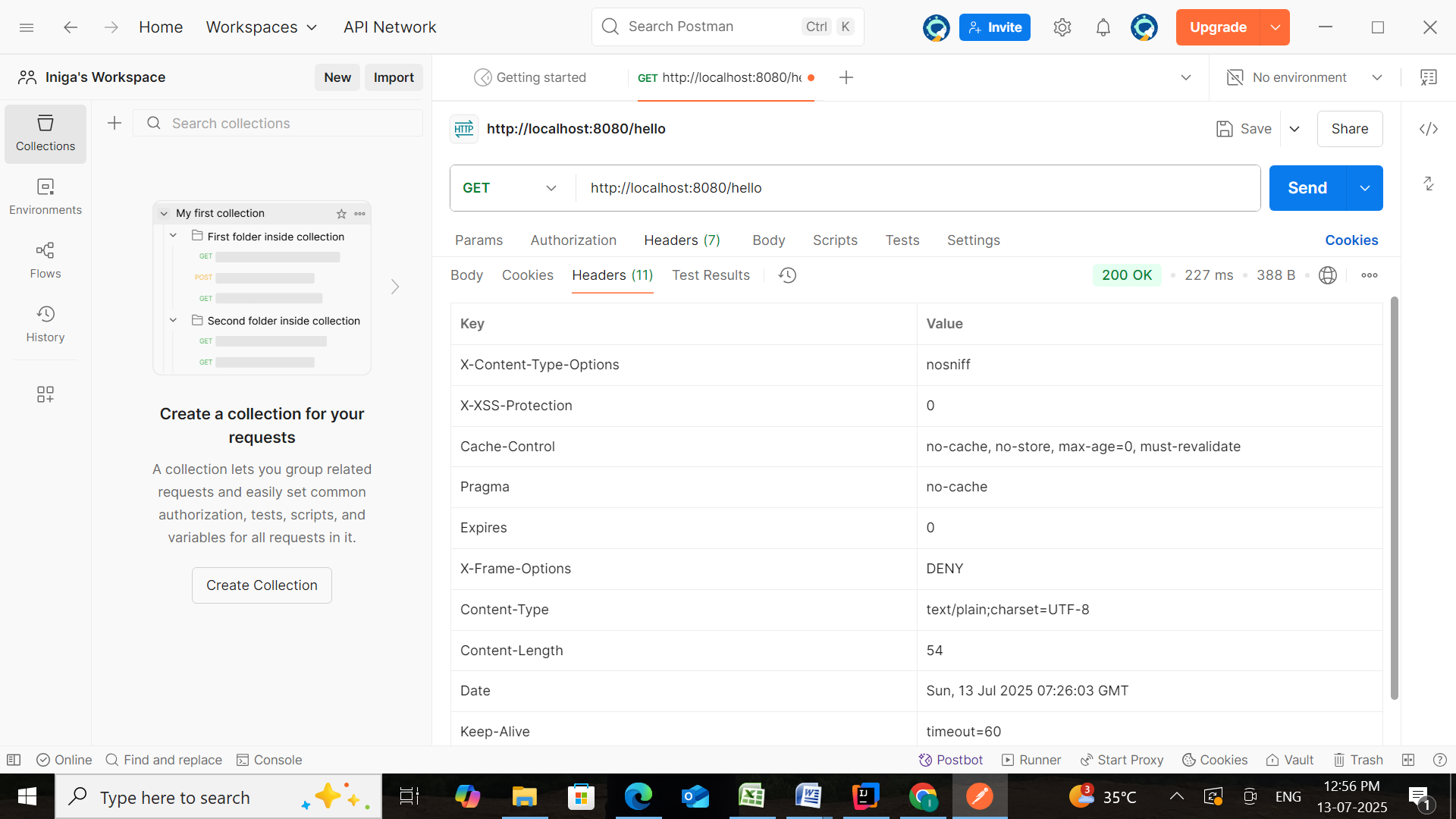




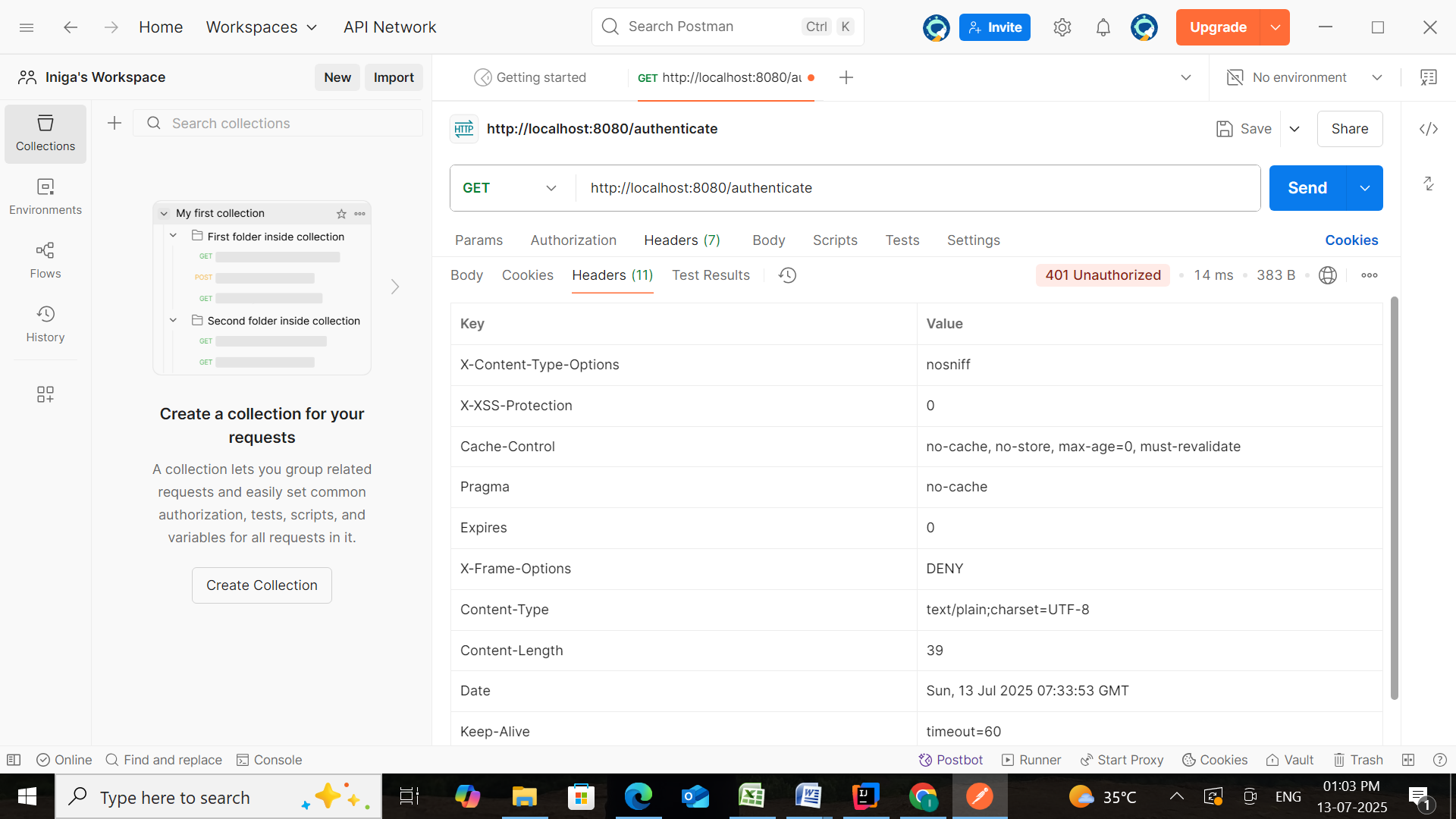


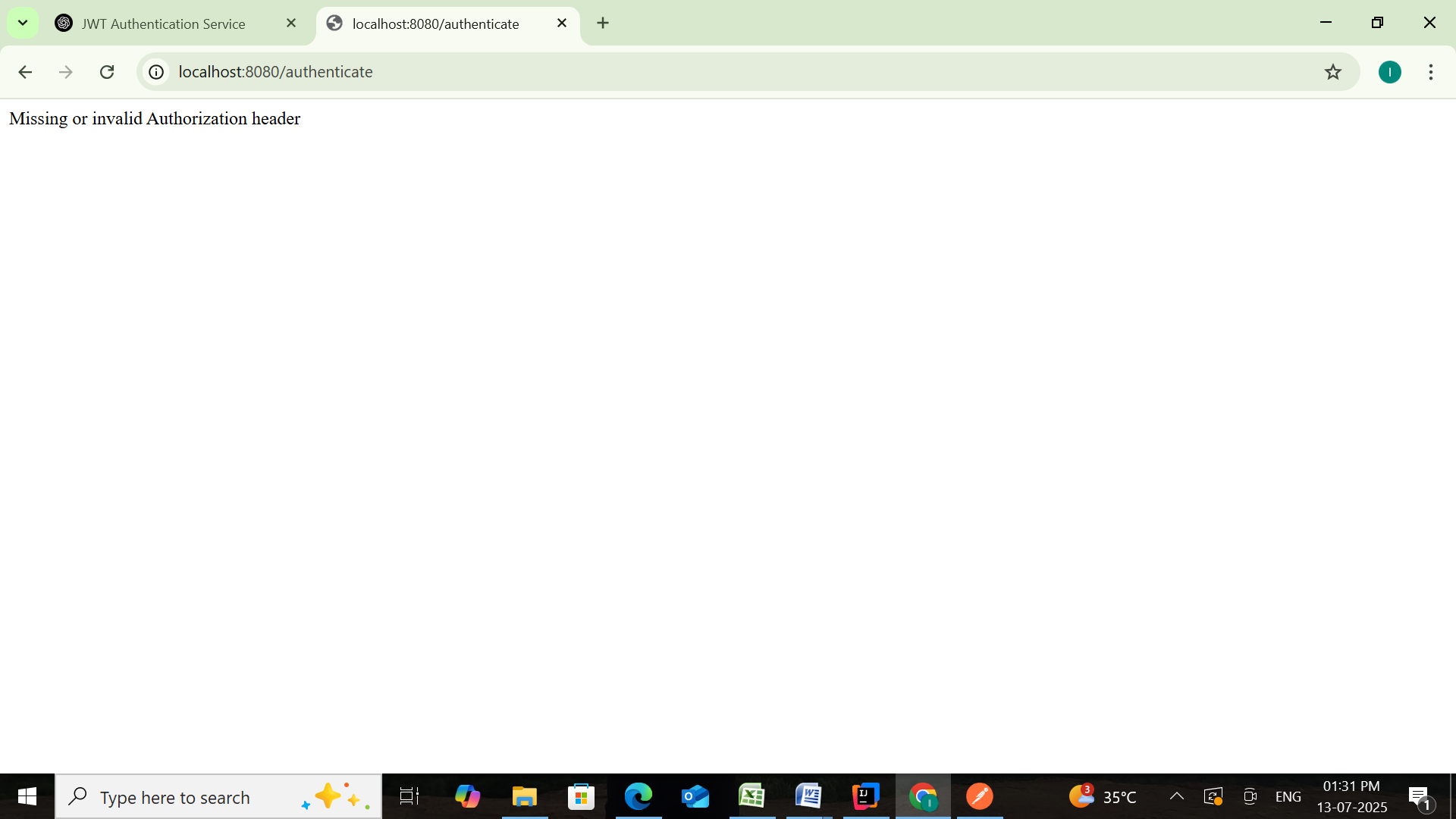
Successful run of /authenticate





Successful run of /hello





401 Missing Error