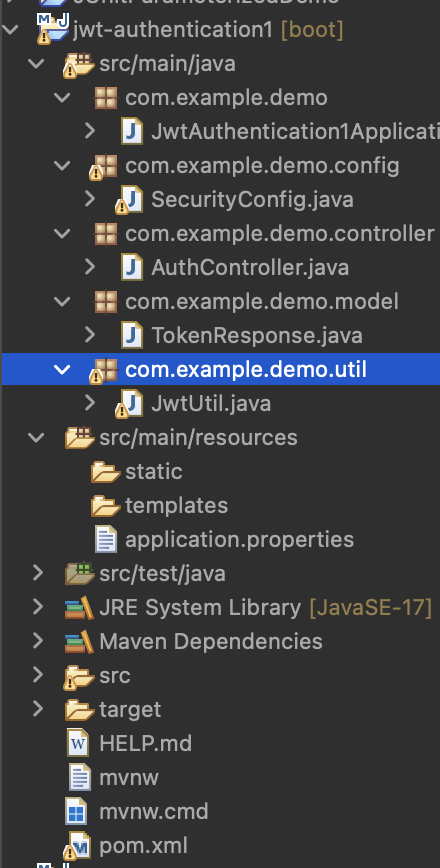
**Create authentication service that returns JWT**



package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class JwtAuthentication1Application {

public static void main(String[] args) {

SpringApplication.run(JwtAuthentication1Application.class, args);

}

}

package com.example.demo.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.builders.AuthenticationManagerBuilder; // Import this

import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration; // Keep this

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

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer; // Keep this

import org.springframework.security.config.http.SessionCreationPolicy; // Keep this

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.crypto.password.~~NoOpPasswordEncoder~~;

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

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

import org.springframework.security.web.SecurityFilterChain;

*@Configuration*

*@EnableWebSecurity*

public class SecurityConfig {

// Expose AuthenticationManager as a Bean

// This allows us to autowire it in AuthController

*@Bean*

public AuthenticationManager authenticationManager(AuthenticationConfiguration authenticationConfiguration) throws Exception {

return authenticationConfiguration.getAuthenticationManager();

}

// Configure an in-memory UserDetailsService for demonstration purposes

*@Bean*

public UserDetailsService userDetailsService() {

UserDetails user = User.*withUsername*("user")

.password("pwd") // Password is "pwd"

.roles("USER")

.build();

return new InMemoryUserDetailsManager(user);

}

// Configure a PasswordEncoder

// WARNING: NoOpPasswordEncoder is NOT secure for production environments.

*@Bean*

public PasswordEncoder passwordEncoder() {

return ~~NoOpPasswordEncoder~~.~~getInstance~~();

}

*@Bean*

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

http

// Disable CSRF for stateless API

.csrf(AbstractHttpConfigurer::disable)

// Configure session management to be stateless, crucial for JWT

.sessionManagement(session -> session.sessionCreationPolicy(*SessionCreationPolicy*.***STATELESS***))

// Configure authorization rules

.authorizeHttpRequests(authorize -> authorize

// Allow requests to "/authenticate" without authentication

// This is crucial for our authentication endpoint

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

// Require authentication for any other request

.anyRequest().authenticated()

)

// Enable HTTP Basic authentication.

// Spring Security's BasicAuthenticationFilter will process the Authorization header.

// It will use the UserDetailsService and PasswordEncoder we defined.

.~~httpBasic~~();

return http.build();

}

}

package com.example.demo.controller;

import com.example.demo.model.TokenResponse;

import com.example.demo.util.JwtUtil;

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

import org.springframework.http.HttpHeaders;

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

import java.util.Base64;

*@RestController*

public class AuthController {

*@Autowired*

private JwtUtil jwtUtil;

*@PostMapping*("/authenticate")

public TokenResponse authenticate(*@RequestHeader*(HttpHeaders.***AUTHORIZATION***) String authHeader) {

// Example: "Basic dXNlcjpwd2Q="

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

String credentials = new String(Base64.*getDecoder*().decode(base64Credentials));

String[] values = credentials.split(":", 2); // [0] = username, [1] = password

String username = values[0];

String password = values[1];

// Hardcoded validation

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

String token = jwtUtil.generateToken(username);

return new TokenResponse(token);

} else {

throw new RuntimeException("Invalid Credentials");

}

}

}

package com.example.demo.model;

public class TokenResponse {

private String token;

public TokenResponse(String token) {

this.token = token;

}

public String getToken() {

return token;

}

}

package com.example.demo.util;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.security.Keys;

import org.springframework.stereotype.Component;

import java.security.Key;

import java.util.Date;

*@Component*

public class JwtUtil {

// Use a 32-byte secret key (for HS256)

private static final String ***SECRET\_KEY*** = "mysecretkey1234567890123456789012";

private Key getSigningKey() {

return Keys.*hmacShaKeyFor*(***SECRET\_KEY***.getBytes());

}

public String generateToken(String username) {

return Jwts.*builder*()

.setSubject(username)

.setIssuedAt(new Date(System.*currentTimeMillis*()))

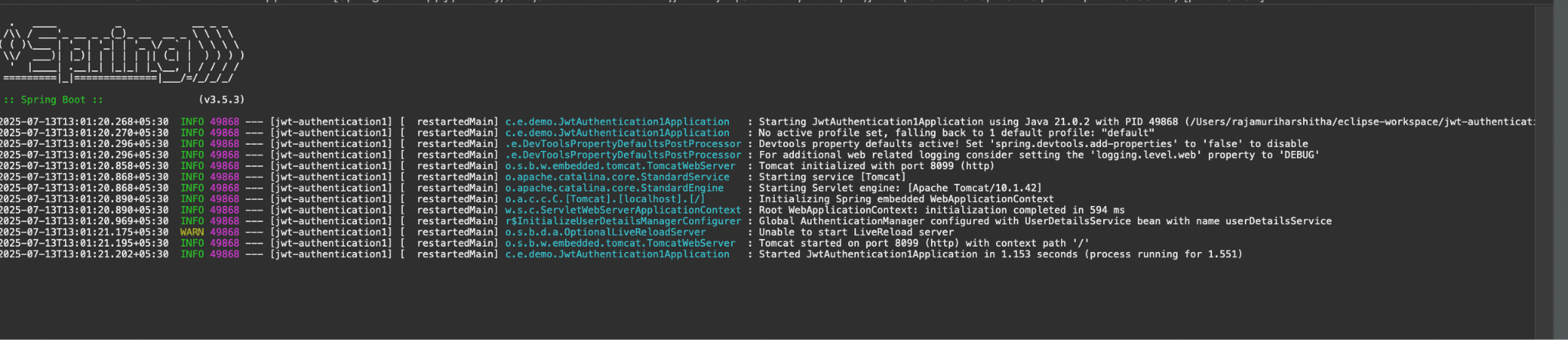
.setExpiration(new Date(System.*currentTimeMillis*() + 1000 \* 60 \* 60)) // 1 hour

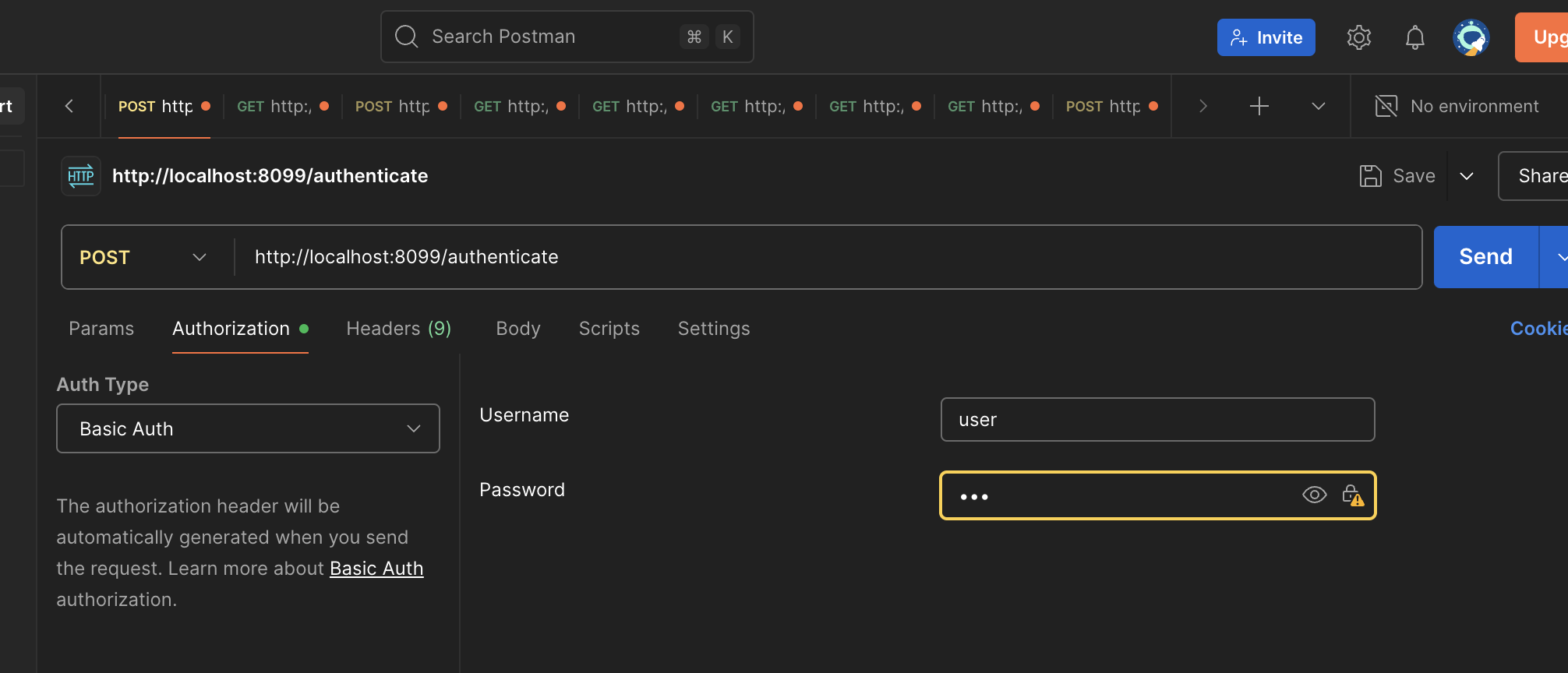
.signWith(*SignatureAlgorithm*.***HS256***, getSigningKey())

.compact();

}

}





Output:

{

"token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiaWF0IjoxNzEwMjM0NTYwLCJleHAiOjE3MTAyMzQ4NjB9.9ZIYv91e3xC7h\_T-6YO\_0V8bUpm8SH\_nBydpRY1ANt8"

}