**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

Let incorporate the above as separate hands on exercises.

**Solution**

**AuthenticationController.java**

package com.cognizant.spring\_learn.controller;  
  
import com.cognizant.spring\_learn.util.JwtUtil;  
import jakarta.servlet.http.HttpServletRequest;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.Authentication;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.nio.charset.StandardCharsets;  
import java.util.Base64;  
import java.util.HashMap;  
import java.util.Map;  
  
@RestController  
public class AuthenticationController {  
  
 @Autowired  
 private AuthenticationManager authenticationManager;  
  
 @Autowired  
 private JwtUtil jwtUtil;  
  
 @GetMapping("/authenticate")  
 public Map<String, String> authenticate(HttpServletRequest request) {  
 try {  
 String authHeader = request.getHeader("Authorization");  
 if (authHeader == null || !authHeader.startsWith("Basic ")) {  
 throw new RuntimeException("Missing or invalid Authorization header");  
 }  
  
 String base64Credentials = authHeader.substring("Basic ".length());  
 byte[] credDecoded = Base64.getDecoder().decode(base64Credentials);  
 String credentials = new String(credDecoded, StandardCharsets.UTF\_8);  
 String[] values = credentials.split(":", 2);  
  
 String username = values[0];  
 String password = values[1];  
  
 Authentication auth = authenticationManager.authenticate(  
 new UsernamePasswordAuthenticationToken(username, password));  
  
 String token = jwtUtil.generateToken(username);  
  
 Map<String, String> tokenMap = new HashMap<>();  
 tokenMap.put("token", token);  
 return tokenMap;  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 throw new RuntimeException("Error in /authenticate: " + e.getMessage());  
 }  
 }  
  
}

**SecurityConfig.java**

package com.cognizant.spring\_learn.security;  
  
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.config.Customizer;  
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.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.web.SecurityFilterChain;  
  
@Configuration  
public class SecurityConfig {  
  
 @Bean  
 public UserDetailsService userDetailsService(PasswordEncoder encoder) {  
 var uds = new InMemoryUserDetailsManager();  
 uds.createUser(User.withUsername("user").password(encoder.encode("pwd")).roles("USER").build());  
 uds.createUser(User.withUsername("admin").password(encoder.encode("pwd")).roles("ADMIN").build());  
 return uds;  
 }  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
 **@Bean**  
 **public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {**  
 **return config.getAuthenticationManager();**  
 **}**  
  
 **@Bean**  
 **public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {**  
 **http.csrf(csrf -> csrf.disable())**  
 **.authorizeHttpRequests(auth -> auth**  
 **.requestMatchers("/authenticate").permitAll()**  
 **.anyRequest().authenticated()**  
 **)**  
 **.httpBasic(Customizer.withDefaults());**  
 **return http.build();**  
 **}**  
}

**Output**

