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.eyJzdWIiOiJ1c2VyIiwiaWF0IjoxNTcwMzc5NDc0LCJleHAiOjE
1NzAzODA2NzR9.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.

Code:

SecurityConfig.java

```java
package com.cognizant.spring_learn.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
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()
            .and()
            .httpBasic();

        return http.build();
    }
}
```

AuthenticationController.java

```java
package com.cognizant.spring_learn.controller;

import com.cognizant.spring_learn.util.JwtUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.Base64;

import jakarta.servlet.http.HttpServletRequest;

@RestController
public class AuthenticationController {

    @Autowired
    private JwtUtil jwtUtil;

    @GetMapping("/authenticate")
    public TokenResponse authenticate(HttpServletRequest request) {
        String authHeader = request.getHeader("Authorization");
        if (authHeader == null || !authHeader.startsWith("Basic ")) {
            throw new RuntimeException("Missing or invalid Authorization header");
        }

        // Decode base64
        String base64Credentials = authHeader.substring("Basic ".length());
        byte[] credDecoded = Base64.getDecoder().decode(base64Credentials);
        String credentials = new String(credDecoded);
        String[] values = credentials.split(":", 2);

        String username = values[0];
        String password = values[1];

        // Basic check: (replace with DB check or UserDetailsService)
        if (username.equals("user") && password.equals("pwd")) {
            String token = jwtUtil.generateToken(username);
            return new TokenResponse(token);
        } else {
            throw new RuntimeException("Invalid credentials");
        }
    }
    static class TokenResponse {
        private String token;

        public TokenResponse(String token) {
            this.token = token;
        }

        public String getToken() {
            return token;
        }

        public void setToken(String token) {
            this.token = token;
        }
    }
}
```

JwtUtil.java

```java
package com.cognizant.spring_learn.util;

import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.stereotype.Component;

import java.util.Date;

@Component
public class JwtUtil {
    private final String secret = "my_secret_key"; // Use strong secret in prod
    private final long expirationTime = 1000 * 60 * 60; // 1 hour

    public String generateToken(String username) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() +
expirationTime))
                .signWith(SignatureAlgorithm.HS256, secret)
                .compact();
    }
}
```

Output: