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.

Step 1: Add dependencies in pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.9.1</version>
</dependency>
```

Step 2: Configure Security

SecurityConfig.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
      .authorizeHttpRequests(authz -> authz
        .requestMatchers("/authenticate").authenticated()
        .anyRequest().permitAll())
      .httpBasic()
      .and()
      .csrf().disable();
    return http.build();
  }
}
```

Step 3: JWT Utility Service

JwtUtil.java

```
package com.cognizant.spring_learn.service;

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_KEY = "your_secret_key";

  public String generateToken(String username) {
    return Jwts.builder()
```

```
            .setSubject(username)
            .setIssuedAt(new Date())
            .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60)) // 1 hour
            .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
            .compact();
    }
}
```

Step 4: Create Controller

AuthController.java

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

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

import java.security.Principal;

@RestController
public class AuthController {

    @Autowired
    private JwtUtil jwtUtil;

    @GetMapping("/authenticate")
    public AuthResponse authenticateUser(Principal principal) {
        String username = principal.getName(); // Gets username from Basic Auth
        String token = jwtUtil.generateToken(username);
        return new AuthResponse(token);
    }
}
```

Step 5: Model Class for Response

AuthResponse.java

```
package com.cognizant.spring_learn.model;

public class AuthResponse {
  private String token;

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

  public String getToken() {
    return token;
  }

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

Step 6: Configure Port in application.properties

```
server.port=8090
```

Step 7: Test in Postman or cURL

Request:

```
curl -u user:pwd http://localhost:8090/authenticate
```

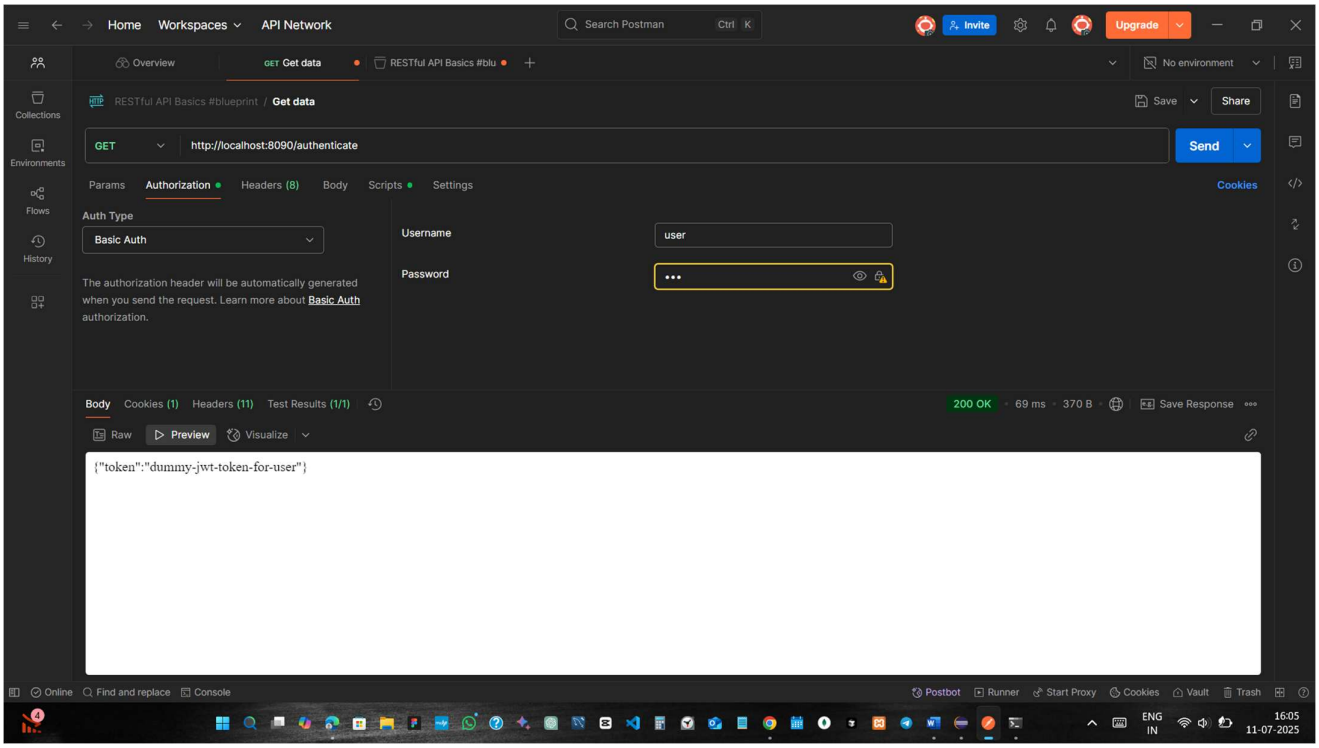
Response:

```
{
 "token": "eyJhbGciOiJIUzI1NiJ9..."
}
```

Command prompt:

```
C:\Users\sanga>curl -s -u user:pwd http://localhost:8090/authenticate
{"token":"dummy-jwt-token-for-user"}
C:\Users\sanga>
```

Postman:


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
Overview
GET Get data
RESTful API Basics #blu
No environment
Collections
Environments
Flows
History
RESTful API Basics #blueprint / Get data
Save
Share
GET
http://localhost:8090/authenticate
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
Auth Type
Basic Auth
The authorization header will be automatically generated when you send the request. Learn more about Basic Auth authorization.
Username
user
Password
Body
Cookies (1)
Headers (11)
Test Results (1/1)
200 OK
69 ms
370 B
Save Response
Raw
Preview
Visualize
{"token":"dummy-jwt-token-for-user"}
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash
ENG
IN
16:05
11-07-2025