Student Name: SHIVENDRA KUMAR KUKKUDAPU

Registration No: 22072A04P3

Course/Batch: VNRVJIET (ECE)

CREATE AUTHENTICATION SERVICE THAT RETURNS JWT

Introduction:

This Spring Boot application implements a basic **JWT-based authentication service**. It allows a client to send user credentials via HTTP Basic Auth to an /authenticate endpoint and receive a signed **JSON Web Token (JWT)** in response. The token can later be used to access protected resources securely.

Objective:

- To create a secure RESTful /authenticate endpoint that accepts user credentials via HTTP Basic Authentication.
- To validate the credentials and generate a JWT token using the io.jsonwebtoken (JJWT) library.
- To configure Spring Security to permit public access to the authentication endpoint while securing other routes.

Implementation Breakdown:

JwtAuthApplication.java:

```
package com.cognizant.jwt.jwt_auth;

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);
	}
}
```

JwtUtil.java:

```
package com.cognizant.jwt.jwt_auth;

import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;

import java.util.Date;

public class JwtUtil {
    private static final String SECRET_KEY = "secret-key";
    private static final long EXPIRATION_TIME = 1000 * 60 * 60;

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

AuthenticationController.java:

```
package com.cognizant.jwt.jwt_auth;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.servlet.http.HttpServletRequest;
import java.util.Base64;

@RestController
public class AuthenticationController {

    @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");
        }
        String base64Credentials = authHeader.substring("Basic ".length());
        String credentials = new
String(Base64.getDecoder().decode(base64Credentials));
        String[] values = credentials.split(":", 2);

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

        if ("user".equals(username) && "pwd".equals(password)) {
            String token = JwtUtil.generateToken(username);
            return ResponseEntity.ok().body("{\"token\":\"" + token + "\"}");
        } else {
            return ResponseEntity.status(401).body("Invalid credentials");
        }
    }
}
```

SecurityConfig.java:

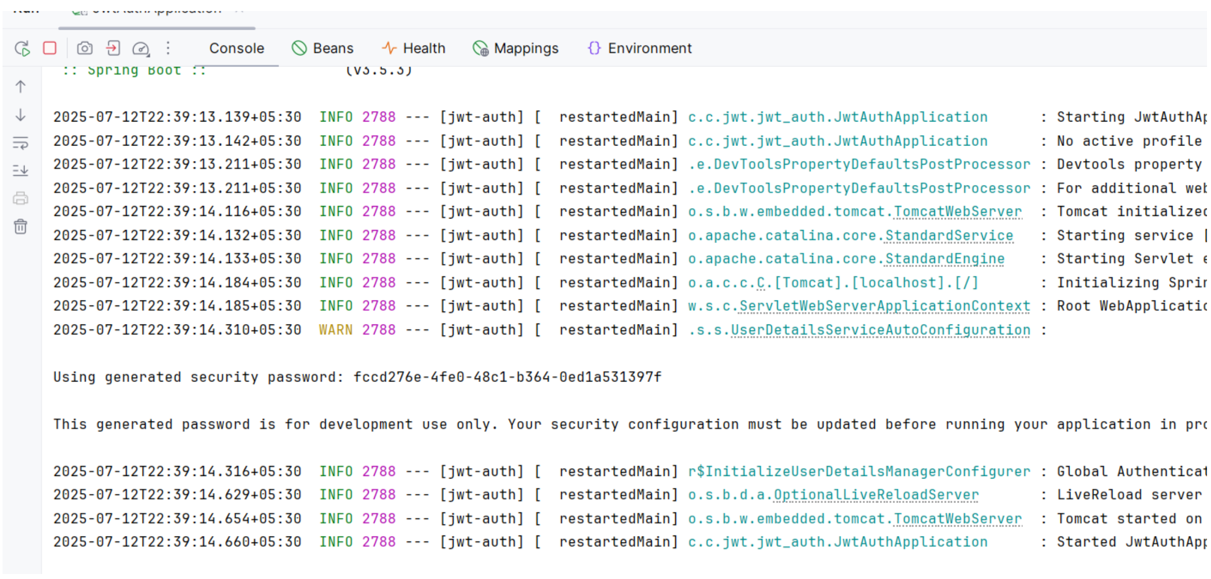
```
package com.cognizant.jwt.jwt_auth;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;

@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception
{
        http
                .csrf(csrf -> csrf.disable())
                .authorizeHttpRequests(auth -> auth
                        .requestMatchers("/authenticate").permitAll()
                        .anyRequest().authenticated()
                );
        return http.build();
    }
}
```

Output:



```
 :: Spring Boot ::                (v3.5.3)

2025-07-12T22:39:13.139+05:30  INFO 2788 --- [jwt-auth] [  restartedMain] c.c.jwt.jwt_auth.JwtAuthApplication      : Starting JwtAuthAp
2025-07-12T22:39:13.142+05:30  INFO 2788 --- [jwt-auth] [  restartedMain] c.c.jwt.jwt_auth.JwtAuthApplication      : No active profile
2025-07-12T22:39:13.211+05:30  INFO 2788 --- [jwt-auth] [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property
2025-07-12T22:39:13.211+05:30  INFO 2788 --- [jwt-auth] [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional we
2025-07-12T22:39:14.116+05:30  INFO 2788 --- [jwt-auth] [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialize
2025-07-12T22:39:14.132+05:30  INFO 2788 --- [jwt-auth] [  restartedMain] o.apache.catalina.core.StandardService   : Starting service
2025-07-12T22:39:14.133+05:30  INFO 2788 --- [jwt-auth] [  restartedMain] o.apache.catalina.core.StandardEngine    : Starting Servlet
2025-07-12T22:39:14.184+05:30  INFO 2788 --- [jwt-auth] [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Sprin
2025-07-12T22:39:14.185+05:30  INFO 2788 --- [jwt-auth] [  restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicati
2025-07-12T22:39:14.310+05:30  WARN 2788 --- [jwt-auth] [  restartedMain] .s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: fccd276e-4fe0-48c1-b364-0ed1a531397f

This generated password is for development use only. Your security configuration must be updated before running your application in pr

2025-07-12T22:39:14.316+05:30  INFO 2788 --- [jwt-auth] [  restartedMain] r$InitializeUserDetailsManagerConfigurer : Global Authentica
2025-07-12T22:39:14.629+05:30  INFO 2788 --- [jwt-auth] [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server
2025-07-12T22:39:14.654+05:30  INFO 2788 --- [jwt-auth] [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on
2025-07-12T22:39:14.660+05:30  INFO 2788 --- [jwt-auth] [  restartedMain] c.c.jwt.jwt_auth.JwtAuthApplication      : Started JwtAuthAp
```

Conclusion:

This hands-on project demonstrates how to implement JWT-based stateless authentication in a Spring Boot application. It offers a solid foundation for securing REST APIs by issuing and verifying tokens, ensuring that only authenticated users can access protected resources. This approach is scalable, lightweight, and widely adopted in modern microservices and web applications.