HANDSON

# **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.

## CODE:

AuthenticationController.java

package com.example.controller;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

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

import java.util.\*;

import java.util.Base64;

@RestController

public class AunthenticationController {

    @GetMapping("/authenticate")

    public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {

        Map<String, String> map = new HashMap<>();

        String user = getUser(authHeader);

        String token = generateJwt(user);

        map.put("token", token);

        return map;

    }

    private String getUser(String authHeader) {

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

        byte[] decodedBytes = Base64.getDecoder().decode(encoded);

        String decoded = new String(decodedBytes);

        return decoded.split(":")[0];

    }

    private String generateJwt(String user) {

        return Jwts.builder()

                .setSubject(user)

                .setIssuedAt(new Date())

                .setExpiration(new Date(System.currentTimeMillis() + 1200000)) *// 20 mins*

                .signWith(SignatureAlgorithm.HS256, "secretkey")

                .compact();

    }

}

# **JWT AUTHORIZATION FILTER.JAVA**

package com.example.security;

import io.jsonwebtoken.\*;

import jakarta.servlet.\*;

import jakarta.servlet.http.\*;

import org.springframework.security.authentication.\*;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.web.authentication.www.BasicAuthenticationFilter;

import java.io.IOException;

import java.util.ArrayList;

public class JwtAuthorizationFilter extends BasicAuthenticationFilter {

    public JwtAuthorizationFilter(AuthenticationManager authManager) {

        super(authManager);

    }

    @Override

    protected void doFilterInternal(HttpServletRequest request,

                                    HttpServletResponse response,

                                    FilterChain chain)

            throws IOException, ServletException {

        String header = request.getHeader("Authorization");

        if (header == null || !header.startsWith("Bearer ")) {

            chain.doFilter(request, response);

            return;

        }

        UsernamePasswordAuthenticationToken auth = getAuthentication(request);

        SecurityContextHolder.getContext().setAuthentication(auth);

        chain.doFilter(request, response);

    }

    private UsernamePasswordAuthenticationToken getAuthentication(HttpServletRequest request) {

        String token = request.getHeader("Authorization");

        if (token != null) {

            try {

                String user = Jwts.parser()

                        .setSigningKey("secretkey")

                        .parseClaimsJws(token.replace("Bearer ", ""))

                        .getBody()

                        .getSubject();

                if (user != null) {

                    return new UsernamePasswordAuthenticationToken(user, null, new ArrayList<>());

                }

            } catch (JwtException e) {

                return null;

            }

        }

        return null;

    }

}

# **SECURITY CONFIG.JAVA**

package com.example.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.builders.AuthenticationManagerBuilder;

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

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

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

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

import org.springframework.security.web.SecurityFilterChain;

import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;

import com.example.security.JwtAuthorizationFilter;

@Configuration

@EnableWebSecurity

public class SecurityConfig {

    @Bean

    public AuthenticationManager authenticationManager(AuthenticationConfiguration authConfig) throws Exception {

        return authConfig.getAuthenticationManager();

    }

    @Bean

    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

        http.csrf().disable()

            .httpBasic()

            .and()

            .authorizeHttpRequests()

            .requestMatchers("/authenticate").hasAnyRole("USER", "ADMIN")

            .anyRequest().authenticated()

            .and()

            .addFilter(new JwtAuthorizationFilter(authenticationManager(http.getSharedObject(AuthenticationConfiguration.class))));

        return http.build();

    }

    @Bean

    public PasswordEncoder passwordEncoder() {

        return new BCryptPasswordEncoder();

    }

    @Bean

    public org.springframework.security.core.userdetails.UserDetailsService users() {

        return username -> {

            if (username.equals("user")) {

                return org.springframework.security.core.userdetails.User.withUsername("user")

                        .password(passwordEncoder().encode("pwd"))

                        .roles("USER").build();

            } else if (username.equals("admin")) {

                return org.springframework.security.core.userdetails.User.withUsername("admin")

                        .password(passwordEncoder().encode("pwd"))

                        .roles("ADMIN").build();

            }

            throw new RuntimeException("User not found");

        };

    }

}

# **SPRING JWT DEMO APPLICATION.JAVA**

package com.example;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringJwtDemoApplication {

    public static void main(String[] args) {

        SpringApplication.run(SpringJwtDemoApplication.class, args);

    }

}

# **APPLICATION.PROPERTIES**

server.port=8090

pom.xml

<project *xmlns*="http://maven.apache.org/POM/4.0.0"

*xmlns:xsi*="http://www.w3.org/2001/XMLSchema-instance"

*xsi:schemaLocation*="http://maven.apache.org/POM/4.0.0

                             http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>

    <artifactId>spring-jwt-demo</artifactId>

    <version>0.0.1-SNAPSHOT</version>

    <packaging>jar</packaging>

    <name>spring-jwt-demo</name>

    <description>Spring Boot JWT Authentication Demo</description>

    <properties>

        <java.version>17</java.version>

        <spring.boot.version>3.2.4</spring.boot.version>

    </properties>

    <dependencies>

    <dependency>

    <groupId>javax.xml.bind</groupId>

    <artifactId>jaxb-api</artifactId>

    <version>2.3.1</version>

</dependency>

<dependency>

    <groupId>org.glassfish.jaxb</groupId>

    <artifactId>jaxb-runtime</artifactId>

    <version>2.3.1</version>

</dependency>

*<!-- ✅ Spring Boot Web -->*

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-web</artifactId>

            <version>${spring.boot.version}</version>

        </dependency>

*<!-- ✅ Spring Boot Security -->*

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-security</artifactId>

            <version>${spring.boot.version}</version>

        </dependency>

*<!-- ✅ JWT Token Library -->*

        <dependency>

            <groupId>io.jsonwebtoken</groupId>

            <artifactId>jjwt</artifactId>

            <version>0.9.0</version>

        </dependency>

*<!-- ✅ Jakarta Servlet API (required for Spring Boot 3+ filters) -->*

        <dependency>

            <groupId>jakarta.servlet</groupId>

            <artifactId>jakarta.servlet-api</artifactId>

            <version>6.0.0</version>

            <scope>provided</scope>

        </dependency>

*<!-- Optional: Lombok (if you want cleaner model code) -->*

        <dependency>

            <groupId>org.projectlombok</groupId>

            <artifactId>lombok</artifactId>

            <version>1.18.30</version>

           <scope>provided</scope>

        </dependency>

*<!-- ✅ Spring Boot Test -->*

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-test</artifactId>

            <version>${spring.boot.version}</version>

            <scope>test</scope>

        </dependency>

        <dependency>

  <groupId>com.fasterxml.jackson.core</groupId>

  <artifactId>jackson-databind</artifactId>

  <version>2.17.1</version> *<!-- ✅ Latest as of mid-2025 -->*

</dependency>

    </dependencies>

    <build>

        <plugins>

*<!-- ✅ Spring Boot Maven Plugin -->*

            <plugin>

                <groupId>org.springframework.boot</groupId>

                <artifactId>spring-boot-maven-plugin</artifactId>

                <version>${spring.boot.version}</version>

            </plugin>

        </plugins>

    </build>

</project>

# **OUTPUT:**

