1. Add dependencies in `pom.xml`

To begin with, ensure that the required dependencies for Spring Security and JWT are included in your `pom.xml`:

```xml
<dependencies>
    <!-- Spring Security -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>

    <!-- JWT Library -->
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt</artifactId>
        <version>0.11.5</version>
    </dependency>

    <!-- Spring Boot Starter Web (For REST API support) -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Spring Boot Starter Data JPA (If required for database connectic
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- Spring Boot Starter Validation -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
</dependencies>
```

---

2. `JWTConfig` Class

This class will contain the necessary configuration for JWT token parsing and security setup.

```java
package com.example.security.config;

import io.jsonwebtoken.*;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSe
import org.springframework.security.config.annotation.web.configuration.E
import org.springframework.security.config.annotation.web.configuration.W
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.UsernamePasswordAu
import com.example.security.filters.JwtAuthenticationFilter;

@Configuration
@EnableWebSecurity
public class JWTConfig extends WebSecurityConfigurerAdapter {

    private final String SECRET_KEY = "your-secret-key"; // Use a strong

    // Bean for Password Encoder (used in authentication)
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    // Bean for JWT Authentication Filter
    @Bean
    public JwtAuthenticationFilter jwtAuthenticationFilter() {
        return new JwtAuthenticationFilter();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .csrf().disable()
            .authorizeRequests()
            .antMatchers("/auth/**").permitAll()  // Allow public access
            .anyRequest().authenticated()       // All other routes requi
            .and()
            .addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordA
    }

    // Utility method to create JWT token
    public String generateToken(String username) {
        return Jwts.builder()
            .setSubject(username)
            .setIssuedAt(new java.util.Date())
            .setExpiration(new java.util.Date(System.currentTimeMillis()
            .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
            .compact();
    }

    // Utility method to validate JWT token
    public boolean validateToken(String token) {
        try {
            Jwts.parser()
                .setSigningKey(SECRET_KEY)
                .parseClaimsJws(token);
            return true;
        } catch (JwtException e) {
            return false;
        }
    }

    // Utility method to get username from JWT token
    public String getUsernameFromToken(String token) {
        return Jwts.parser()
            .setSigningKey(SECRET_KEY)
            .parseClaimsJws(token)
            .getBody()
            .getSubject();
    }
}
```

---

3. `JwtAuthenticationFilter` Class

This filter will intercept the HTTP request and authenticate the JWT token.

```
package com.example.security.filters;

import com.example.security.config.JWTConfig;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.filter.OncePerRequestFilter;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import java.io.IOException;

public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final JWTConfig jwtConfig = new JWTConfig();

    @Override
    protected void doFilterInternal(HttpServletRequest request, javax.ser
        String token = request.getHeader("Authorization");

        if (token != null && token.startsWith("Bearer ")) {
            token = token.substring(7);  // Remove 'Bearer ' prefix

            if (jwtConfig.validateToken(token)) {
                String username = jwtConfig.getUsernameFromToken(token);
                SecurityContextHolder.getContext().setAuthentication(new
            }
        }
        filterChain.doFilter(request, null);
    }
}
```

4. `AuthController` for Authentication

Here, we implement an endpoint to authenticate and generate the JWT token.

```
package com.example.security.controllers;

import com.example.security.config.JWTConfig;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/auth")
public class AuthController {

    @Autowired
    private JWTConfig jwtConfig;
    // Login endpoint to generate JWT token
    @PostMapping("/login")
    public String login(@RequestParam String username, @RequestParam Stri
        // Authenticate user (basic check, in real case use service)
        if ("user".equals(username) && "password".equals(password)) {
            return jwtConfig.generateToken(username);  // Return generate
        }
        return "Invalid Credentials";
    }
}
```

---

5. Security Configuration

Ensure that your **Spring Security** is configured to use the JWT token correctly. This is done through the `JWTConfig` class as shown above.

- **JWT Authentication Filter** is added before the `UsernamePasswordAuthenticationFilter`.
- You can customize which endpoints should be public (`/auth/**` in this case) and which require authentication.

---

6. `application.properties` Configuration

If necessary, add configurations for Spring Security and JWT in your `application.properties`:

```
# JWT Configuration (Use a stronger secret key in production)
jwt.secret.key=your-secret-key
jwt.expiration.time=86400000  # 1 day
```
