JWT Authentication in Java Spring Boot 3.4.3

1. Dependencies

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.11.5</version>
</dependency>

<dependency>
                        <groupId>io.jsonwebtoken</groupId>
                        <artifactId>jjwt-impl</artifactId>
                        <version>0.11.5</version>
                        <scope>runtime</scope>
                </dependency>
                <dependency>
                        <groupId>io.jsonwebtoken</groupId>
                        <artifactId>jjwt-jackson</artifactId>
                        <version>0.11.5</version>
                        <scope>runtime</scope>
                </dependency>
                <dependency>
                        <groupId>io.jsonwebtoken</groupId>
                        <artifactId>jjwt-gson</artifactId>
                        <version>0.11.5</version>
                </dependency>
```

2. Create JwtUtil to Generate & Validate JWT

This class is responsible for generating and validating JWT tokens.

```
import io.jsonwebtoken.*;
import io.jsonwebtoken.security.Keys;
import org.springframework.stereotype.Component;
import java.security.Key;
import java.util.Date;
```

```
@Component
public class JwtUtil {
    private static final String SECRET_KEY = "your-secret-key-your-secret-key-your-secret-key";
    private static final long EXPIRATION_TIME = 1000 * 60 * 60; // 1 Hour

    private final Key key = Keys.hmacShaKeyFor(SECRET_KEY.getBytes());

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

    public String extractUsername(String token) {
        return Jwts.parserBuilder()
                .setSigningKey(key)
                .build()
                .parseClaimsJws(token)
                .getBody()
                .getSubject();
    }

    public boolean validateToken(String token) {
        try {
            Jwts.parserBuilder().setSigningKey(key).build().parseClaimsJws(token);
            return true;
        } catch (JwtException e) {
            return false;
        }
    }
}
```

3. Create AuthController for Login

This controller exposes a login endpoint to authenticate users and return a JWT.

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

import java.util.Map;

@RestController
@RequestMapping("/auth")
```

```
public class AuthController {
  private final JwtUtil jwtUtil;

  public AuthController(JwtUtil jwtUtil) {
    this.jwtUtil = jwtUtil;
  }

  @PostMapping("/login")
  public ResponseEntity<Map<String, String>> login(@RequestBody Map<String, String> request) {
    String username = request.get("username");
    String password = request.get("password");

    // Hardcoded username & password for demo purposes
    if ("admin".equals(username) && "password".equals(password)) {
      String token = jwtUtil.generateToken(username);
      return ResponseEntity.ok(Map.of("token", token));
    } else {
      return ResponseEntity.status(401).body(Map.of("error", "Invalid credentials"));
    }
  }
}
```

4. Secure API with JWT Authentication

Create JwtFilter to Check Requests
This filter extracts and validates JWT tokens from incoming requests.

```
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import java.io.IOException;

@Component
public class JwtFilter extends OncePerRequestFilter {
  private final JwtUtil jwtUtil;

  public JwtFilter(JwtUtil jwtUtil) {
    this.jwtUtil = jwtUtil;
  }
```

```
    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
FilterChain filterChain)
            throws ServletException, IOException {

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

        if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {
            String token = authorizationHeader.substring(7);
            if (jwtUtil.validateToken(token)) {
                String username = jwtUtil.extractUsername(token);
                UserDetails userDetails = User.withUsername(username).password("").roles("USER").build();
                UsernamePasswordAuthenticationToken authToken =
                        new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }

        filterChain.doFilter(request, response);
    }
}
```

Configure Spring Security

We configure Spring Security to allow public access to /auth/login but protect all other routes.

```
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.config.http.SessionCreationPolicy;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
public class SecurityConfig {
    private final JwtFilter jwtFilter;

    public SecurityConfig(JwtFilter jwtFilter) {
        this.jwtFilter = jwtFilter;
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        return http
                .csrf(csrf -> csrf.disable())
```

```
            .sessionManagement(session ->
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                  .requestMatchers("/auth/login").permitAll()
                  .anyRequest().authenticated())
            .addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class)
            .build();
   }
}
```

5. Create a Protected API Endpoint

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.Map;

@RestController
@RequestMapping("/secure")
public class SecureController {

   @GetMapping("/message")
   public Map<String, String> secureMessage() {
      return Map.of("message", "You have accessed a protected API!");
   }
}
```

6. Test the JWT Authentication

Get JWT Token
Make a POST request to login:

```
curl -X POST http://localhost:8080/auth/login -H "Content-Type: application/json" -d
'{"username":"admin", "password":"password"}'
```

Response:

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

Access Secure Endpoint
Use the token from the previous step:

```
curl -X GET http://localhost:8080/secure/message -H "Authorization: Bearer YOUR_JWT_TOKEN"
```

Response:

```
{
  "message": "You have accessed a protected API!"
}
```