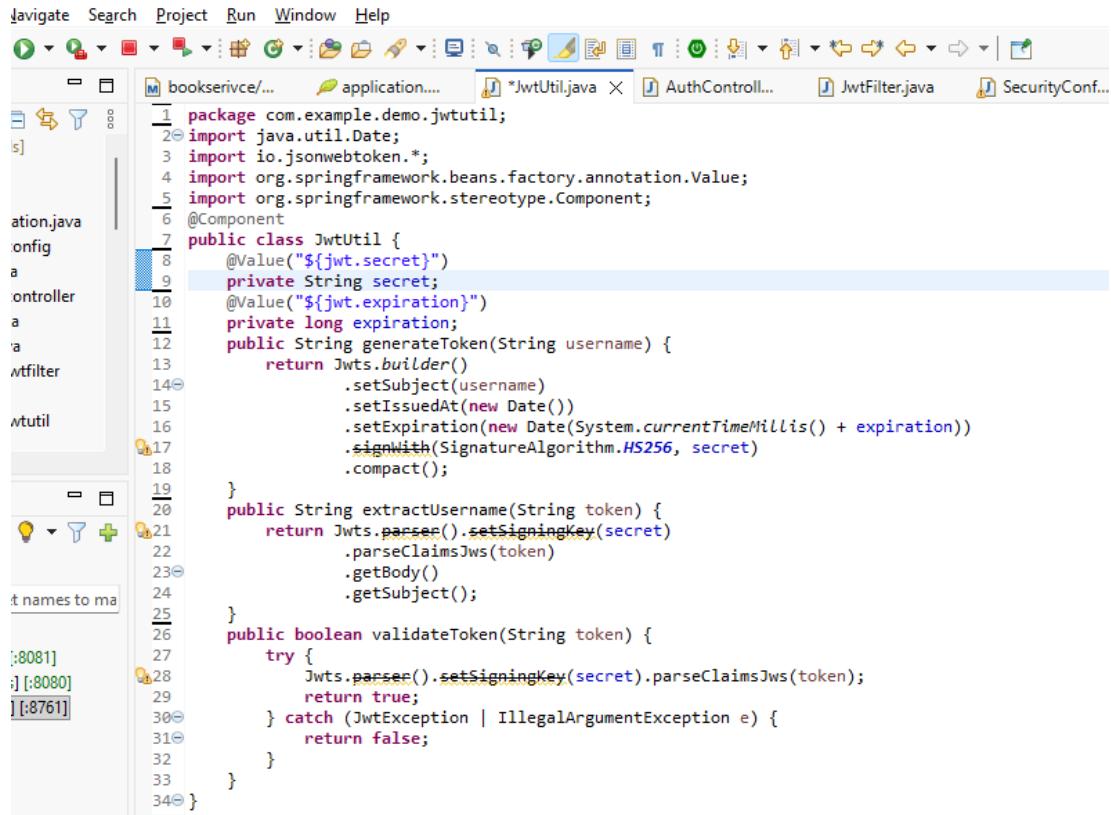


Daily task : 10/10/2025

Jwtutil.java



Screenshot of the IntelliJ IDEA IDE showing the JwtUtil.java file. The code implements a JWT utility class using the JJWT library. It includes methods for generating tokens, extracting user names from tokens, and validating tokens.

```
1 package com.example.demo.jwtutil;
2 import java.util.Date;
3 import io.jsonwebtoken.*;
4 import org.springframework.beans.factory.annotation.Value;
5 import org.springframework.stereotype.Component;
6 @Component
7 public class JwtUtil {
8     @Value("${jwt.secret}")
9     private String secret;
10    @Value("${jwt.expiration}")
11    private long expiration;
12    public String generateToken(String username) {
13        return Jwts.builder()
14            .setSubject(username)
15            .setIssuedAt(new Date())
16            .setExpiration(new Date(System.currentTimeMillis() + expiration))
17            .signWith(SignatureAlgorithm.HS256, secret)
18            .compact();
19    }
20    public String extractUsername(String token) {
21        return Jwts.parser().setSigningKey(secret)
22            .parseClaimsJws(token)
23            .getBody()
24            .getSubject();
25    }
26    public boolean validateToken(String token) {
27        try {
28            Jwts.parser().setSigningKey(secret).parseClaimsJws(token);
29            return true;
30        } catch (JwtException | IllegalArgumentException e) {
31            return false;
32        }
33    }
34}
```

AuthController.java



Screenshot of the IntelliJ IDEA IDE showing the AuthController.java file. The controller handles authentication requests, specifically logging in users by generating JWT tokens.

```
1 package com.example.demo.controller;
2 import java.util.Map;
3
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.http.ResponseEntity;
6 import org.springframework.web.bind.annotation.PostMapping;
7 import org.springframework.web.bind.annotation.RequestBody;
8 import org.springframework.web.bind.annotation.RequestMapping;
9 import org.springframework.web.bind.annotation.RestController;
10
11 import com.example.demo.jwtutil.JwtUtil;
12
13 @RestController
14 @RequestMapping("/auth")
15 public class AuthController {
16
17     @Autowired
18     private JwtUtil jwtUtil;
19
20     @PostMapping("/login")
21     public ResponseEntity<Map<String, String>> login(@RequestBody Map<String, String> body) {
22         String username = body.get("username");
23         String password = body.get("password");
24
25         if ("admin".equals(username) && "admin123".equals(password)) {
26             String token = jwtUtil.generateToken(username);
27             return ResponseEntity.ok(Map.of("token", token));
28         } else {
29             return ResponseEntity.status(401).body(Map.of("error", "Invalid username or password"));
30         }
31     }
32 }
33 }
```

JwtFilter.java

The screenshot shows the Eclipse IDE interface with theJwtFilter.java file open in the editor. The code implements a OncePerRequestFilter that checks for a JWT token in the Authorization header and delegates to another filter if it's present. It also handles unauthorized requests by setting a status code.

```
import org.springframework.web.filter.OncePerRequestFilter;
import java.io.IOException;
@Component
public class JwtFilter extends OncePerRequestFilter {
    @Autowired
    private JwtUtil jwtutil;
    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain)
        throws ServletException, IOException {
        String path = request.getRequestURI();
        if (path.startsWith("/auth")) {
            filterChain.doFilter(request, response);
            return;
        }
        String authHeader = request.getHeader("Authorization");
        if (authHeader != null && authHeader.startsWith("Bearer ")) {
            String token = authHeader.substring(7);
            if (!jwtutil.validateToken(token)) {
                response.setStatus(HttpStatus.SC_UNAUTHORIZED);
                return;
            }
        } else {
            response.setStatus(HttpStatus.SC_UNAUTHORIZED);
            return;
        }
        filterChain.doFilter(request, response);
    }
}
```

SecurityConfig.java

The screenshot shows the Eclipse IDE interface with theSecurityConfig.java code. It configures a security filter chain with CSRF protection disabled, basic authentication disabled, and a custom JWT filter added before the password authentication filter.

```
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
@Configuration
public class SecurityConfig {
    @Autowired
    private JwtFilter jwtfilter;
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/auth/**").permitAll()
                .anyRequest().authenticated()
            )
            .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .httpBasic().disable();
        http.addFilterBefore(jwtfilter, UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }
    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {
        return config.getAuthenticationManager();
    }
}
```

Postman

POST | http://localhost:8081/auth/login

Params Authorization Headers (8) **Body** Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```
1  {
2    "username": "admin",
3    "password": "password"
4 }
```

Body Cookies Headers (11) Test Results

200 OK • 684 ms • 466 B

Raw Preview Visualize

```
1 eyJhbGciOiJIUzI1NiJ9eyJzdWIiOiJhZG1pbkiIsImlhdcI6MTc2MDEwNDU0OSwiZXhwIjoxNzYwMTA4MT05fQ.aPvIkUULmtCQRc8meF6ixGUqpub5HANe7A4n2
```

POST | http://localhost:8081/books

Params **Authorization** * Headers (9) Body Scripts Settings

Auth Type

Token

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

Body Cookies Headers (11) Test Results

201 Created • 477 ms • 404 B

JSON Preview Visualize

```
1  {
2    "isbn": "101",
3    "title": "Spring Boot",
4    "author": "Oak",
5    "quantity": 10
6 }
```

GET | http://localhost:8081/books

Params Authorization Headers (7) Body Scripts Settings

Auth Type: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

Token: [Copy](#) [Reset](#)

Body Cookies Headers (11) Test Results [?](#) 200 OK | 325 ms | 401 B | [Raw](#)

{ } JSON ▾ | Preview | Visualize | ▾

```
1 [  
2 {  
3   "isbn": "101",  
4   "title": "Spring Boot",  
5   "author": "Oak",  
6   "quantity": 10  
7 }
```