Implémentation d'un JWT Token avec Spring Boot

🧩 1. Dépendances nécessaires (Maven)

```
<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-api</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>
</dependencies>
```

🔑 2. Classe `JwtUtil`

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

@Component
public class JwtUtil {
    private final String SECRET =
"maCleSuperSecretePourJWTDe32CaracteresMinimum";
    private final long EXPIRATION_TIME = 86400000; // 24h
```

```
    private Key getSigningKey() {
        return Keys.hmacShaKeyFor(SECRET.getBytes());
    }

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

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

    public boolean validateToken(String token) {
        try {

Jwts.parserBuilder().setSigningKey(getSigningKey()).build().parseClaimsJws
(token);
            return true;
        } catch (JwtException e) {
            return false;
        }
    }
}
```

🧍 3. Filtre `JwtAuthenticationFilter`

```
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import
org.springframework.security.authentication.UsernamePasswordAuthentication
Token;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
```

```
import java.io.IOException;

@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final JwtUtil jwtUtil;
    private final UserDetailsService userDetailsService;

    public JwtAuthenticationFilter(JwtUtil jwtUtil, UserDetailsService
userDetailsService) {
        this.jwtUtil = jwtUtil;
        this.userDetailsService = userDetailsService;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response,
                                    FilterChain filterChain) throws
ServletException, IOException {
        String authHeader = request.getHeader("Authorization");

        if (authHeader != null && authHeader.startsWith("Bearer ")) {
            String token = authHeader.substring(7);
            String username = jwtUtil.extractUsername(token);

            if (username != null &&
SecurityContextHolder.getContext().getAuthentication() == null) {
                UserDetails userDetails =
userDetailsService.loadUserByUsername(username);

                if (jwtUtil.validateToken(token)) {
                    UsernamePasswordAuthenticationToken authToken =
                            new
UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());

SecurityContextHolder.getContext().setAuthentication(authToken);
                }
            }
        }

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

⚙️ 4. Configuration `SecurityConfig`

```
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.configuratio
n.AuthenticationConfiguration;
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.UsernamePasswordAuthentica
tionFilter;

@Configuration
public class SecurityConfig {

    private final JwtAuthenticationFilter jwtAuthFilter;

    public SecurityConfig(JwtAuthenticationFilter jwtAuthFilter) {
        this.jwtAuthFilter = jwtAuthFilter;
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http)
throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**").permitAll()
                .anyRequest().authenticated())
            .sessionManagement(session ->
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .addFilterBefore(jwtAuthFilter,
UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }

    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration config) throws Exception
{
        return config.getAuthenticationManager();
    }
}
```

🔐 5. Contrôleur `AuthController`

```
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthentication
Token;
```

```java
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.*;

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

    private final AuthenticationManager authenticationManager;
    private final JwtUtil jwtUtil;

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

    @PostMapping("/login")
    public String login(@RequestBody AuthRequest request) {
        Authentication authentication =
authenticationManager.authenticate(
                new
UsernamePasswordAuthenticationToken(request.getUsername(),
request.getPassword()));
        return jwtUtil.generateToken(request.getUsername());
    }
}

class AuthRequest {
    private String username;
    private String password;
    // getters/setters
}
```

🧪 6. Test avec Postman

1. **POST** `/api/auth/login`

```
{
  "username": "admin",
  "password": "admin"
}
```

→ Réponse : `eyJhbGciOiJIUzI1NiJ9...`

2. **GET** `/api/users`
   → Header :
   `Authorization: Bearer eyJhbGciOiJIUzI1NiJ9...`

✅ Résultat

- Authentification **stateless**
- JWT signé avec clé secrète
- Protection automatique des routes Spring