Manuale della Sicurezza in Spring Boot

Introduzione

Questa guida spiega in dettaglio come funziona il sistema di autenticazione basato su **JWT (JSON Web Token)** in un'applicazione **Spring Boot**. Il sistema gestisce l'autenticazione degli utenti e la protezione delle API, consentendo l'accesso solo a utenti autorizzati.

Struttura e Funzionamento della Sicurezza

1. **Registrazione e Login**
    - Un utente può registrarsi e accedere con username e password.
    - Se il login è corretto, viene generato un **token JWT** che l'utente utilizza per autenticarsi nelle API.
2. **Protezione delle API**
    - Ogni richiesta alle API deve includere il token JWT.
    - Il token viene validato e l'utente autenticato viene identificato.
3. **Ruoli e Autorizzazioni**
    - Gli utenti possono avere ruoli diversi (`USER` , `ADMIN` , `SELLER`).
    - Le autorizzazioni vengono definite in base ai ruoli.

Spiegazione dei File

1. `AppUser.java` – Modello dell'utente

📌 **Cosa fa?**
Rappresenta un utente nel database e implementa `UserDetails` di Spring Security.

🔍 **Dettagli principali:**

- Contiene **username**, **password** e **ruoli**.
- Implementa il metodo `getAuthorities()` per restituire i ruoli dell'utente.
- **Spring Security** utilizza questa classe per l'autenticazione.

2. `AppUserRepository.java` – Interfaccia per il database

📌 **Cosa fa?**
Definisce i metodi per recuperare e verificare gli utenti nel database.

🔍 **Metodi principali:**

- `findByUsername(String username)` : Cerca un utente per username.
- `existsByUsername(String username)` : Controlla se un utente esiste.

3. `AppUserService.java` – Logica di autenticazione

📌 **Cosa fa?**
Gestisce la registrazione, autenticazione e recupero degli utenti.

🔍 **Funzionalità principali:**

- `registerUser()` : Registra un nuovo utente.

- `authenticateUser()` : Verifica le credenziali e genera un token JWT.
- `loadUserByUsername()` : Cerca un utente nel database.

4. `AuthController.java` – API per login e registrazione

📌 **Cosa fa?**
Fornisce endpoint per autenticare e registrare gli utenti.

🔍 **Endpoint principali:**

- `POST /api/auth/register` : Registra un nuovo utente.
- `POST /api/auth/login` : Autentica un utente e restituisce un **token JWT**.

5. `AuthResponse.java` – Risposta del login

📌 **Cosa fa?**
Contiene il token JWT restituito dopo il login.

🔍 **Struttura del token JWT:**

```
{
  "token": "eyJhbGciOiJIUzI1..."
}
```

6. `JwtTokenUtil.java` – Generazione e gestione del token JWT

📌 Cosa fa?

Gestisce la creazione, validazione e parsing del token JWT.

🔍 Funzionalità principali:

- `generateToken()` : Genera un JWT con **username** e **ruoli**.
- `getUsernameFromToken()` : Estrae l'username dal token.
- `getRolesFromToken()` : Estrae i ruoli dal token.
- `validateToken()` : Controlla se il token è valido e non scaduto.

🔷 Come viene generato un token?

```
public String generateToken(UserDetails userDetails) {
    Collection<? extends GrantedAuthority> authorities = userDetails.getAuthorities();
    List<String> roles = authorities.stream()
                                    .map(GrantedAuthority::getAuthority)
                                    .collect(Collectors.toList());

    return Jwts.builder()
            .setSubject(userDetails.getUsername())
            .claim("roles", roles) // Aggiunge ruoli come claim
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + jwtExpirationInMs))
            .signWith(SignatureAlgorithm.HS256, secret)
            .compact();
}
```

7. `JwtRequestFilter.java` – Filtro JWT

📌 Cosa fa?

Intercepta le richieste HTTP e controlla se contengono un **token JWT valido**.

🔍 **Passaggi chiave:**

1. **Legge il token dall'header `Authorization`.**
2. **Estrae l'username dal token.**
3. **Verifica il token e autentica l'utente.**
4. **Imposta l'utente nel contesto di sicurezza di Spring.**

🔷 **Esempio di codice:**

```
String token = request.getHeader("Authorization").substring(7);
String username = jwtTokenUtil.getUsernameFromToken(token);

UserDetails userDetails = customUserDetailsService.loadUserByUsername(username);

if (jwtTokenUtil.validateToken(token, userDetails)) {
    UsernamePasswordAuthenticationToken authenticationToken =
        new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
    SecurityContextHolder.getContext().setAuthentication(authenticationToken);
}
```

8. `SecurityConfig.java` – Configurazione della sicurezza

📌 **Cosa fa?**
Configura le regole di accesso dell'applicazione.

🔍 **Configurazioni principali:**

- **Disabilita CSRF** (`csrf().disable()`)
- **Definisce la politica di sessione** (`STATELESS`)
- **Aggiunge il filtro JWT** (`addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class)`)

🔷 **Esempio di protezione:**

```
http
    .authorizeHttpRequests(auth -> auth.anyRequest().authenticated());
```

9. `JwtAuthenticationEntryPoint.java` – Gestione accessi non autorizzati

📌 **Cosa fa?**

Restituisce un errore `401 Unauthorized` quando un utente non autenticato prova ad accedere a un'API protetta.

🔍 **Risultato di una richiesta senza token:**

```
HTTP/1.1 401 Unauthorized
Accesso non autorizzato
```

10. `CorsConfig.java` – Configurazione CORS

📌 **Cosa fa?**

Permette al browser di inviare richieste API da domini diversi.

Se non configurato correttamente, il browser potrebbe **bloccare le richieste** per motivi di sicurezza.

🔍 **Soluzione:**

```
registry.addMapping("/**")
    .allowedOrigins("*") // Accetta richieste da qualsiasi dominio
    .allowedMethods("*") // Consente tutti i metodi (GET, POST, PUT, DELETE)
    .allowedHeaders("*");
```

Ruolo del Token JWT

Il **token JWT** è un identificatore unico che permette agli utenti di autenticarsi senza dover reinserire le credenziali ad ogni richiesta.
Contiene **claims**, ovvero informazioni come:

- **username**
- **ruoli**
- **data di scadenza**
- **chiave di firma** (per verificare l'autenticità del token)

🔍 **Esempio di un JWT decodificato:**

```
{
  "sub": "admin",
  "roles": ["ROLE_ADMIN"],
  "exp": 1712065312
}
```

Iniettare il Token nei Controller

Per ottenere i dettagli dell'utente autenticato all'interno di un controller, si può usare:

```
@GetMapping("/me")
public ResponseEntity<String> getAuthenticatedUser(@AuthenticationPrincipal UserDetails userDetails) {
    return ResponseEntity.ok("Logged in as: " + userDetails.getUsername());
}
```

8. `OpenApiConfig.java` – Configurazione Swagger

📌 **Cosa fa?**
Configura Swagger per documentare l'API e aggiunge un meccanismo di sicurezza con JWT.

🔍 **Effetto:**
Swagger mostrerà un campo dove inserire il token JWT per testare gli endpoint.