

Spring Security

Spring Security

- ▶ Radni okvir za podršku bezbednosti u Spring aplikacijama
- ▶ Zasniva se na tehnikama deklarativnog programiranja
- ▶ Pokriva autentifikaciju i autorizaciju na nivou HTTP zahteva, ali i poziva pojedinačnih metoda

Moduli

▶ Modul za konfiguraciju

- ▶ podrška za konfigurisanje bezbednosti u aplikaciji kroz XML ili Java anotacije

▶ Jezgro (*core*)

- ▶ biblioteka sa ključnim funkcijama za bezbednost

▶ Kriptografija

- ▶ podrška za kriptovanje šifara

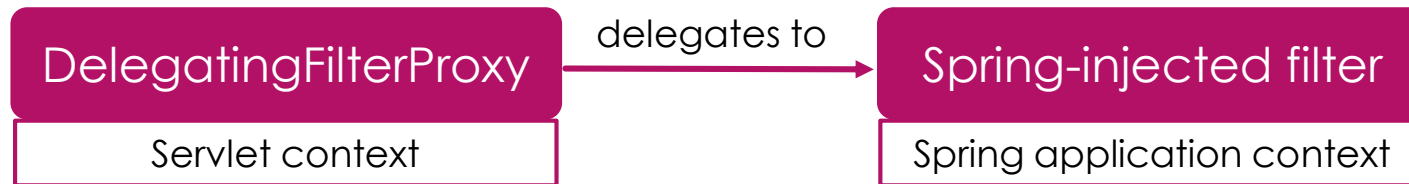
▶ Veb

- ▶ podrška za bezbednost veb aplikacije kroz filtere za procesiranje zahteva

- spring-security-core.jar
- spring-security-remoting.jar
- spring-security-web.jar
- spring-security-config.jar
- spring-security-ldap.jar
- spring-security-oauth2-core.jar
- spring-security-oauth2-client.jar
- spring-security-oauth2-jose.jar
- spring-security-acl.jar
- spring-security-cas.jar
- spring-security-openid.jar
- spring-security-test.jar

Filtriranje veb zahteva

- ▶ Spring sadrži generički filter `DelegatingFilterProxy` koji može da delegira posao specifičnom filteru



- ▶ Spring Security tu može da injektuje svoj specifičan filter `springSecurityFilterChain`
- ▶ Spring Boot automatski injektuje ovaj *bean* kroz anotaciju `@EnableWebSecurity`

Konfiguracija bezbednosti

- ▶ U klasi koja nasleđuje `WebSecurityConfigurerAdapter` klasu
- ▶ Redefinišu se sledeće metode:
 - ▶ `configureAuthentication (AuthenticationManagerBuilder)`
 - ▶ definiše način utvrđivanja identiteta korisnika pri autentikaciji
 - ▶ `configure (HttpSecurity)`
 - ▶ definiše prava pristupa za zahteve ka određenim URL
 - ▶ `configure (WebSecurity)`
 - ▶ generalna bezbednost veb aplikacije
 - ▶ ignorisanje resursa, *firewall*, ...

Autentifikacija korisnika

- ▶ Za autentifikaciju korisnika potrebno je da postoje uskladišteni podaci o korisnicima aplikacije i njihovim ulogama u sistemu
- ▶ Spring ima ugrađenu podršku za različite standardne pristupe skladištenju
 - ▶ *in-memory*
 - ▶ jdbc
 - ▶ LDAP
- ▶ Može se realizovati i nestandardna podrška za utvrđivanje identiteta korisnika

Reprezentacija korisnika

- ▶ Korisnik je predstavljen interfejsom `UserDetails`
- ▶ Za korisnika se minimalno evidentira:
 - ▶ **korisničko ime**
 - ▶ **lozinka**
 - ▶ **lista uloga** (uloga predstavljena interfejsom `GrantedAuthority`)
- ▶ Način pristupa je predstavljen interfejsom `UserDetailsService`

Nestandardna autentifikacija korisnika

- ▶ Interfejs `UserDetailsService` ima samo metodu `loadUserByUsername(String username)`
- ▶ Potrebno je implementirati interfejs i redefinisati ovu metodu na proizvoljan način
 - ▶ npr. korišćenjem JPA repozitorijuma za pristup bazi korisnika
- ▶ Pri konfiguraciji autentifikacije u metodi `configure` je potrebno navesti koja klasa je zadužena za autentifikaciju

Presretanje HTTP zahteva

- ▶ Spring security presreće HTTP zahteve posebnim filterom
- ▶ Može se definisati da li korisnik ima pravo da dobije odgovor na zahtev koji je poslat na određenu putanju
- ▶ Putanje se definišu u stilu koji koristi Ant alat za izgradnju softvera
 - ▶ Definisanje se vrši u metodi `configure(HttpSecurity)`
- ▶ Definiše se ko ima pravo da pristupi putanji:
 - ▶ ulogovani korisnici
 - ▶ svi korisnici
 - ▶ oni koji imaju određenu ulogu u sistemu
 - ▶ oni koji upućuju zahtev sa određene IP adrese
 - ▶ ...

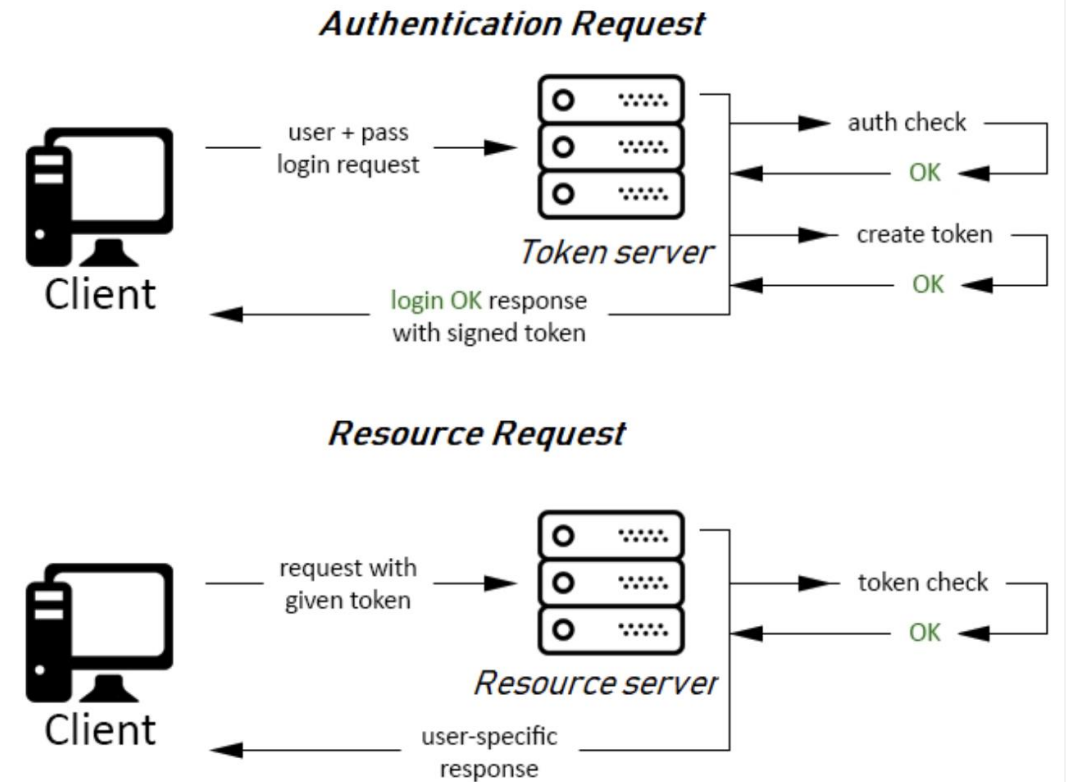
Šifrovanje lozinki

- ▶ Spring security ima ugrađenu podršku za skladištenje lozinki u šifrovanom obliku
- ▶ Dovoljno je instancirati odgovarajuću klasu za šifrovanje lozinki i primeniti je pri konfigurisanju načina identifikacije korisnika

```
@Override  
public void configure(AuthenticationManagerBuilder auth) throws Exception {  
    auth.userDetailsService(userDetailsService).passwordEncoder(new BCryptPasswordEncoder());  
}
```

Autentifikacija bazirana na tokenima

- ▶ HTTP protokol je *stateless* pa je potreban mehanizam identifikacije klijenta koji šalje višestruke zahteve
- ▶ Nakon uspešne autentifikacije, klijent dobija token koji je server generisao
- ▶ Sa svakim sledećim zahtevom, klijent šalje token kao informaciju o svom identitetu



JSON veb token (JWT)

- ▶ Trenutno dominantan format u kojem se tokeni za autentikaciju reprezentuju
- ▶ <https://jwt.io/>
- ▶ To je string koji se sastoji iz tri dela:
 1. **zaglavlje** (*header*)
 - ▶ tip tokena i algoritam kriptovanja
 2. **glavni sadržaj** (*payload*)
 - ▶ podaci o korisniku na kojeg se token odnosi, rok trajanja tokena, ...
 3. **potpis** (*signature*)
 - ▶ string generisan kriptovanjem zaglavlja, sadržaja i serverove tajne reči

JSON web token (JWT)

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

PAYLOAD: DATA

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "iat": 1516239022  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
) ☐ secret base64 encoded
```

Realizacija JWT autentikacije kroz Spring Security

- ▶ Potrebno je nakon uspešnog logovanja generisati JWT token i poslati ga klijentu
- ▶ Svaki zahtev je potrebno presresti kako bi se proverilo da li sadrži validan JWT token
 - ▶ presretanje se vrši dodavanjem novog filtera
 - ▶ filter će postaviti odgovarajuću autentifikaciju ako je token validan
 - ▶ dalje Spring security dozvoljava pristup putanji zavisno od uspešnosti autentifikacije i ranije konfigurisanih prava pristupa putanji