



Spring Boot en Hibernate

Security



Leerdoelen

- Wat is authenticatie en wat is autorisatie
- Form-based authentication
- Basic Auth
- Zelf een gebruiker toevoegen
- Password encoding
- Basic auth gebruiken in JS

Authorization vs Authenticatie

Deze twee termen worden in de volksmond vaak door elkaar gebruikt, maar maken voor ons een wezenlijk verschil.

Authenticatie

- Wie ben jij?
- Bewijs dat jij bent wie je zegt dat je bent



Authorizatie

- Okee, we weten wie je bent
- Maar mag je bij de informatie die je opvraagt?
- Heb je toestemming om een operatie uit te voeren?



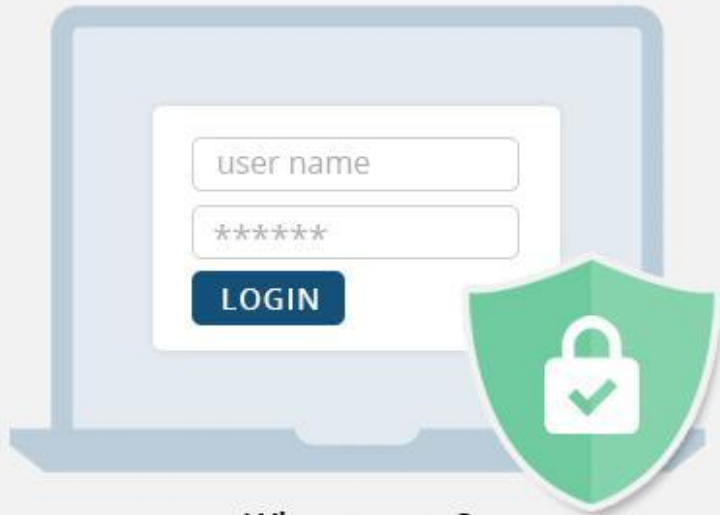
Authenticatie vs Autorizatie

Authenticatie is dus het controleren wie je bent

En autorizatie is controleren of jij toestemming hebt om iets te doen

Authenticatie vs Autorisatie

Authentication



Who are you?

Validate a system is accessing by the right person

Authorization



Are you allowed to do that?

Check users' permissions to access data

Beginnen met security in Spring

Spring heeft een dependency waarmee we simpel security kunnen toepassen

- We gaan naar start.spring.io en voegen de volgende dependencies toe:
 - Spring Security
 - Spring Web

Dit is voldoende om mee te beginnen. We kunnen straks endpoints maken en daar restricties op toepassen.

Eerst een endpoint

```
- import org.springframework.web.bind.annotation.GetMapping;  
- import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class ResourceController {  
  
    @GetMapping("/")  
    public String home() { return("<h1>Welcome</h1>"); }  
    +  
}
```

En draaien

- Als we de applicatie draaien, zien we in de console het volgende bericht verschijnen:

```
Using generated security password: 4670ebb7-10cb-46ad-967d-bb48e3af37bf
```

```
This generated password is for development use only. Your security configuration must be updated before running your application in production.
```

- De applicatie heeft een wachtwoord gegenereerd
- Verder een waarschuwing om dit niet in productie te gebruiken en alleen tijdens het ontwikkelen

Endpoint aanspreken

- Laten we een browser openen en naar localhost:8080/ gaan, waar onze applicatie draait.
- We verwachten hier ons stukje HTML uit ons endpoint te zien
- Maar...

Een loginscherm

Please sign in

Sign in

Spring Security

- Dit is een standaard inlogschermb van Spring Security
- Ziet er prima uit, maar dit is niet helemaal wat we willen
- Als we via httpie proberen dit endpoint aan te spreken, krijgen we hetzelfde scherm te zien
- Ook niet zo handig als we vanuit een frontend dit endpoint aanspreken
- Gelukkig kunnen we dit gedrag aanpassen!

Spring Security

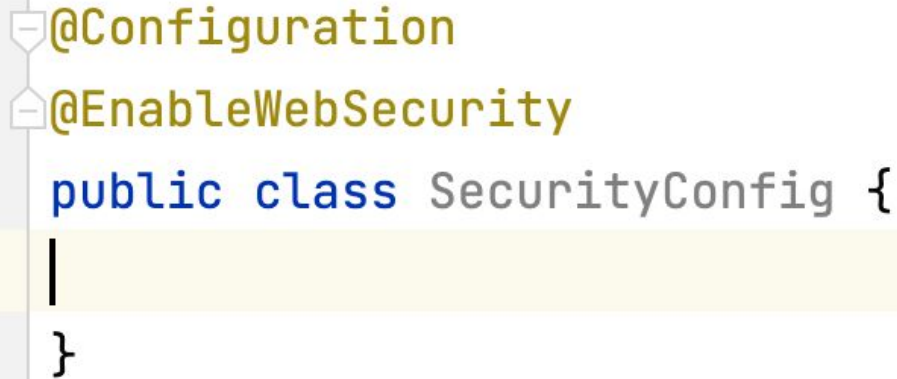
- Dit is een standaard inlogschermb van Spring Security
- Ziet er prima uit, maar dit is niet helemaal wat we willen
- Als we via httpie proberen dit endpoint aan te spreken, krijgen we hetzelfde scherm te zien
- Ook niet zo handig als we vanuit een frontend dit endpoint aanspreken
- Gelukkig kunnen we dit gedrag aanpassen!

Spring Security

- We maken een nieuwe package in ons project: configuration
- Daarin maken we een nieuwe klasse: SecurityConfig
- Boven de klasse zetten we twee annotaties:
 - @Configuration
 - @EnableWebSecurity
- Met @Configuration geven we aan dat deze klasse onderdeel is van het Spring framework
- Met @EnableWebSecurity geven we dat we in deze klasse de beveiliging van onze applicatie gaan inrichten

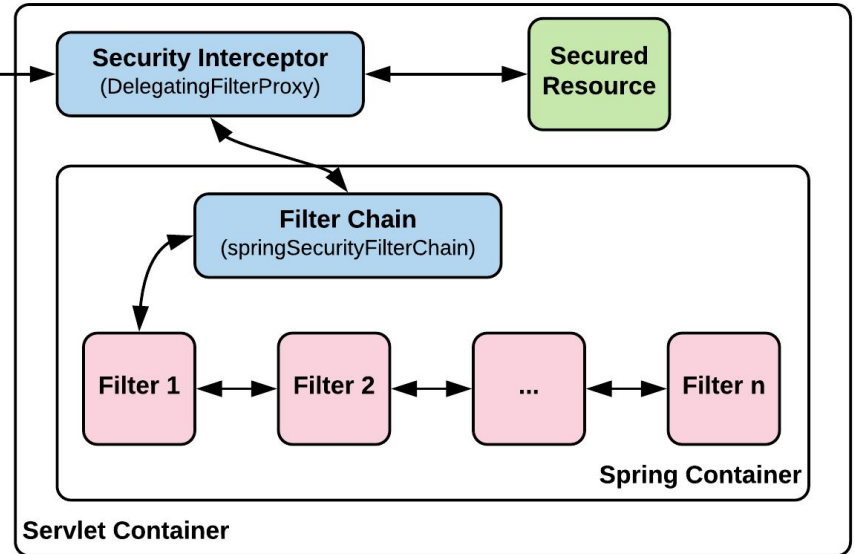
Spring Security

De basis van onze configuratie!

```
A code snippet from an IDE. The first two lines, '@Configuration' and '@EnableWebSecurity', are annotated with a shield icon. The third line, 'public class SecurityConfig {', is highlighted in blue. The fourth line, a vertical bar '|', is highlighted in yellow. The fifth line, '}', is highlighted in grey.  
@Configuration  
@EnableWebSecurity  
public class SecurityConfig {  
|  
}
```


En nu?

- We kunnen beginnen met configureren, maar waar beginnen we?
- We gebruiken een filter om toegang to endpoints te beheren
- Deze filter komt tussen de gebruiker en onze endpoints in te staan
- Laten we eerst een tweede endpoint maken
- Dit endpoint gaan we zo beveiligen



Tweede endpoint

```
5 @RestController
7 public class ResourceController {
8
9     @GetMapping("/")
10    public String home() { return("<h1>Welcome</h1>"); }
11
12    @GetMapping("/user")
13    public String user() {
14        return("<h1>User</h1>");
15    }
16 }
```

Security Filter

- We kunnen nu onze filter op gaan bouwen
- We maken in de SecurityConfiguration klasse een nieuwe methode, filterChain
- Deze methode
 - neemt een HttpSecurity object
 - throwsd een Exception
 - returned een SecurityFilterChain
 - Is een @Bean

Security Filter

@Configuration

@EnableWebSecurity

```
public class SecurityConfig {
```

@Bean

```
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
```

```
}
```

```
}
```

Bonen?

- We hebben de annotatie `@Bean` gebruikt voor de securityfilter
- Maar wat hebben bonen te maken met methodes?
- Beans zijn objecten en methodes die door het Spring framework gemanaged worden
- Doordat we de `SecurityConfiguration` klasse met `@Configuration` gemarkeerd hebben, gaat Spring op zoek naar `@Bean` 's in die klasse
- Spring zorgt ervoor dat deze methodes beschikbaar zijn voor de applicatie

Filters maken

- We kunnen nu in de `filterChain` methode regels gaan maken waar het verkeer van onze applicatie zo aan moet voldoen.
- We gaan matchen op urls van onze endpoints en kunnen gebruikers met een rol (Authorizatie) toegang geven tot deze endpoints
- Dit doen we met behulp van method chaining
- We kijken eerst even naar een voorbeeld

Filters maken

@Bean

```
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
    http  
        .authorizeHttpRequests() AuthorizeHttpRequestsConfigurer<...>.AuthorizationManagerRequestMatcherSecurityExpressionFactory  
        .requestMatchers( ...patterns: "/user") AuthorizeHttpRequestsConfigurer<...>.AuthorizedUrl  
        .hasRole("USER");  
    return http.build();  
}
```

Filters maken

- We pakken het `HttpSecurity` object wat binnenkomt
- We geven eerst aan dat http requests geauthoriseerd moeten worden.
- Vervolgens matchen we op een url met `requestMatchers(url)`
- Wat opvalt is dat deze methode direct op een andere methode wordt aangeroepen. Dit heet method chaining.
- Daarna geven we aan welke rol toegang heeft tot de url, namelijk de `USER` rol.
- Als laatste bouwen we het resultaat van onze method chain en returnen dit.

Terug naar de applicatie

- Als we nu proberen een endpoint aan te spreken in onze browser, krijgen we een 403 error.
- We zijn unauthorized om een endpoint aan te spreken
- Zelfs de root van onze applicatie ("/") kunnen we niet aanspreken. We hebben hier nog geen regels voor ingesteld, en dus is deze op slot gedaan
- Daarnaast is onze inlogformulier verdwenen

Eerst het formulier

- We kunnen het inlogformulier terug krijgen door een extra regel aan onze filterchain toe te voegen
- Door achteraan onze chain `.formLogin()` toe te voegen, krijgen we het formulier terug.
- Als we dit neerzetten, word `formLogin()` rood.
- We moeten eerst het blokje van matchers afsluiten voor we aan de volgende beginnen. Dit doen we met `.and()`

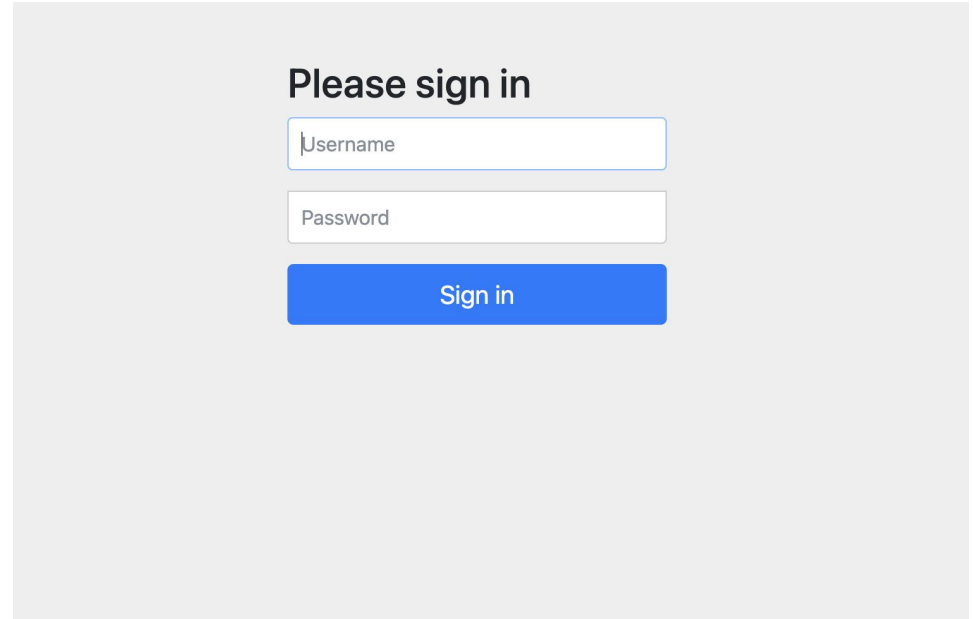
Eerst het formulier

@Bean

```
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
    http  
        .authorizeHttpRequests() AuthorizeHttpRequestsConfigurer<...>.AuthorizationManagerRequestMatcher  
        .requestMatchers( ...patterns: "/user" AuthorizeHttpRequestsConfigurer<...>.AuthorizedUrl  
        .hasRole("USER") AuthorizeHttpRequestsConfigurer<...>.AuthorizationManagerRequestMatcher  
        .and() HttpSecurity  
        .formLogin();  
    return http.build();  
}
```

Inlog

- Ons inlogscherf is weer terug!
- Als we naar /user gaan, en inloggen krijgen we de header te zien
- Maar, als we naar / gaan, mogen we er nog steeds niet in



Please sign in

Username

Password

Sign in

Tijd voor een nieuwe regel

- We willen een wildcard hebben, zodat we op al onze endpoints die niet /user zijn iedereen toelaten
- Gelukkig kunnen we dit maken met de requestMatchers() methode door “/**” als argument mee te geven.
- Vervolgens geven we met permitAll() aan dat iedereen hierbij mag
- De volgorde van filters is belangrijk! Daar komen we zo op terug

Tijd voor een nieuwe regel

@Bean

```
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
    http  
        .authorizeHttpRequests() AuthorizeHttpRequestsConfigurer<...>.AuthorizationManagerReque  
        .requestMatchers( ...patterns: "/user") AuthorizeHttpRequestsConfigurer<...>.AuthorizedUrl  
        .hasRole("USER") AuthorizeHttpRequestsConfigurer<...>.AuthorizationManagerRequestMatcherI  
  
        .requestMatchers( ...patterns: "/*") AuthorizeHttpRequestsConfigurer<...>.AuthorizedUrl  
        .permitAll() AuthorizeHttpRequestsConfigurer<...>.AuthorizationManagerRequestMatcherRegis  
        .and() HttpSecurity  
  
        .formLogin();  
    return http.build();  
}
```

We hebben weer toegang

- We kunnen weer bij onze root pagina ("/")
- We hebben aangegeven dat alleen een gebruiker met rol "USER" bij /user kan
- Maar waar komt deze rol vandaag?
- Dit is een ingebouwde rol van Spring Security
- Er is ook nog een "Admin" rol
- Laten we een extra endpoint maken en alleen een admin toegang geven

Admin Endpoint

```
@RestController
public class ResourceController {

    @GetMapping("/")
    public String home() { return ("<h1>Welcome</h1>"); }

    @GetMapping("/user")
    public String user() {
        return ("<h1>User</h1>");
    }

    @GetMapping("/admin")
    public String admin() {
        return ("<h1>Admin</h1>");
    }
}
```


Vragen?

- E-mail mij op voornaam.achternaam@code-cafe.nl!
- Join de Code-Café community op discord!

