Wprowadzenie #1

Spring Security to potężny i wysoce konfigurowalny framework uwierzytelniania i kontroli dostępu. Jest to standard de-facto dla zabezpieczania aplikacji opartych na Springu.

Spring Security to framework, który koncentruje się na dostarczaniu zarówno uwierzytelniania, jak i autoryzacji do aplikacji Java. Jak wszystkie projekty Spring, prawdziwa siła Spring Security tkwi w tym, jak łatwo można go rozszerzyć, aby spełnić niestandardowe wymagania.

Główne cechy:

- ➢ Kompleksowe i rozszerzalne wsparcie dla uwierzytelniania i autoryzacji
- ➢ Ochrona przed atakami typu session fixation, clickjacking, cross site request forgery itp.
- ➢ Integracja z API serwletów
- ➢ Opcjonalna integracja z Spring Web MVC
- ➢ Wsparcie dla konfiguracji w języku Java

Wprowadzenie #2

Podprojekt Spring Security był pierwotnie oddzielnym projektem rozpoczętym w 2003 roku o nazwie Acegi Security System. Po 3 latach rozwoju oddzielnie od samego Springa, został oficjalnie przyjęty w 2007 roku przez Spring i przemianowany na Spring Security. Dzisiaj podprojekt jest standardem uwierzytelniania i kontroli dostępu dla Spring.

Podobnie jak wiele innych projektów Spring (w tym Spring Data), podprojekt Spring Security jest dalej podzielony na wiele innych podprojektów opartych na konkretnych cechach. Najbardziej podstawowe są: spring-security-core, spring-security-config oraz spring- security-web.

Jest wiele innych bibliotek, o których warto wspomnieć, jak OAuth (która ma własną grupę artefaktów, org.springframework.security.oauth, z wieloma artefaktami zapewniającymi specyficzne cechy) i JOSE. Istnieje również wsparcie dla CAS (system pojedynczego logowania (SSO); CAS oznacza "Central Authentication Service") i OpenID (szeroka specyfikacja uwierzytelniania w Internecie). Istnieją biblioteki, które skupiają się tylko na aspekcie uwierzytelniania i te, które integrują różne pomysły otaczające autoryzację.

Funkcjami Spring Security są:

- Authorization
- Single sign-on
- Software Localization
- Remember-me
- LDAP (Lightweight Directory Access Protocol)
- JAAS (Java Authentication and Authorization Service) LoginModule
- Web Form Authentication
- Digest Access Authentication
- HTTP Authorization
- Basic Access Authentication

Wprowadzenie #3

Jeśli uruchamiasz typową aplikację (internetową), potrzebujesz uwierzytelnić swoich użytkowników. Oznacza to, że twoja aplikacja musi zweryfikować, czy użytkownik jest tym, za kogo się podaje, zazwyczaj odbywa się to za pomocą sprawdzania nazwy użytkownika i hasła.
Kiedy mówimy o tym w odniesieniu do uwierzytelniania opartego na sieci, mówimy ogólnie o wielu różnych metodach, z najprostszym dostępem HTTP BASIC, aż do bardziej zaangażowanych mechanizmów, takich jak OpenID lub CAS.

W prostszych aplikacjach uwierzytelnianie może być wystarczające: gdy tylko użytkownik uwierzytelni się, może uzyskać dostęp do każdej części aplikacji.
Jednak większość aplikacji ma koncepcję uprawnień (lub ról).
Wyobraźmy sobie: klientów, którzy mają dostęp do publicznego frontu jakiegoś sklepu internetowego oraz administratorów, którzy mają dostęp do oddzielnego obszaru administracyjnego.
Oba typy użytkowników muszą się zalogować, ale sam fakt uwierzytelnienia nie mówi nic o tym, co mogą robić w danym systemie. W tym celu należy również sprawdzić uprawnienia uwierzytelnionego użytkownika, czyli trzeba go autoryzować.

2. Configuration

Konfiguracja [AP]

Aby móc korzystać ze Spring Security należy dodać odpowiednie moduły podczas budowania projektu w Gradle/Maven.
Są to między innymi moduły omówione w poprzedni podrozdziale, takie jak:
- spring-security-core
- spring-security-config
- spring-security-web

Konfiguracja #2 [AP]

Należy upewnić się, że framework Spring Security jest zarejestrowany dla każdego linku URL w naszej aplikacji.
W tym celu należy utworzyć klasę
o przykładowej nazwie „GatewaySecurityWebApplicationInitializer”, która rozszerza klasę „AbstractSecurityWebApplicationInitializer”.
Spring zajmie się obsługą tych klas resztą.

Spring Security działa w sieci Web, wykorzystując mechanizm filtrów serwletów do przechwytywania wywołań HTTP, dopasowując ich treść do mapowań zabezpieczeń. Robi to samo dla wywołań metod, tyle że z dynamicznym proxy zamiast filtra serwletów.

Konfiguracja #3 [AP]

Za pomocą metody getRootConfigClasses() zwracany jest config Spring Security.

Security w praktyce #1 [AP]
Przykład kodu Java z użyciem Spring Security, który zostanie przeanalizowany w następnych slajdach:

Security w praktyce #2 [AP]

Adnotacja @EnableWebSecurity pozwala Spring Security wiedzieć, aby użyć tej klasy do konfiguracji.

W nadpisanej metodzie userDetailsService wykorzystana została klasa InMemoryUserDetailsManager, która pozwala programiście na zarządzanie użytkownikami przy użyciu modułów bezpieczeństwa.
Aby tego dokonać należy utworzyć obiekt UserDetails używają wzorca konstrukcyjnego obiektu User

Na co warto zwrócić uwagę?:

- Przechowywanie zakodowanych haseł
- Każdy użytkownik musi mieć przypisane atrybuty GrantedAuthority i Role

a) GrantedAuthority – np. READ_AUTHORITY, WRITE_AUTHORITY zależy od programisty jak je nazwać i obsłużyć
b) Role – np. ADMIN, USER, SUPERUSER – tak samo jest to arbitralne i zależy od programisty

Moduły obiektu HttpSecurity [AP]

Za pomocą obiektu HttpSecurity możemy nadpisać konfigurację według własnych potrzeb. Obiekt ten wykorzystuje wzorzec konstruktora, który możemy użyć do skonfigurowania różnych opcji, miedzy innymi:

- And – przydatne dla łańcuchów metod, ponieważ zwraca obiekt SecurityBuilder
- authorizeRequests – ogranicza dostęp na podstawie klasy HttpServletRequest
- antMatchers – pozwala na zezwolenie lub ograniczenie funkcjonalności
- formLogin –upewnia się, że używamy logowania opartego na formularzu
- httpBasic – wykorzystuje prostą metodę uwierzytelniania HTTP Basic
- hasRole – zapewnia, że odpowiednio skonfigurowana trasa posiada określoną rolę
- permitALL – adresy URL dopasowane przez tą metodę są dozwolone przez każdego

- loginPage - określa adres URL do wysłania użytkownikom, jeśli wymagane jest logowanie
- Cors - dodaje corsFilter do dopasowanych żądań
- rememberMe – pozwala na skonfigurowanie autentykacji „Pamiętaj mnie”

Strona logowania #1 [AP]

Poniższy kod zapewni, że każde żądanie do tej strony spotka się z uruchomieniem autoryzacji Basic używając httpBasic() i wymagając dla uwierzytelnionego użytkownika roli „ADMIN”.

Strona logowania #2 [AP]

Spring Security obsługuje wszystkie opcje komunikatów i wyświetlania, co oznacza, że jeśli użytkownik wprowadzi niepoprawne dane uwierzytelniające, to otrzyma odpowiedni komunikat o błędzie.

W naszej konfiguracji określiliśmy, że użytkownik musi posiadać rolę ADMIN. Sensownym rozwiązaniem jest zdefiniowanie prostej strony, aby pokazać ją użytkownikowi po pomyślnym zalogowaniu. Możemy do tego wykorzystać prosty kontroler

Własny formularz logowania |Slajd x

Wykorzystując Spring Security mamy do dyspozycji domyślną stronę logowania (wygląda ona jak na screenie przedstawionym na slajdzie) która jest wbudowana w framework. Jeśli nie dokonamy dodatkowej konfiguracji będzie ona wykorzystywana w aplikacji. W większości aplikacji oczywiście programiści nie wykorzystują tej możliwości, dopasowując stronę logowania, tak aby była dopasowana do wytwarzanej aplikacji. Spring Security umożliwia elastyczną opcję konfiguracji strony logowania, co pokażemy na kilku kolejnych slajdach.

Własny formularz logowania - metoda konfiguracyjna |Slajd x

Największą zmianą która tak naprawdę musimy zrobić w celu stworzenia własnej strony logowania jest modyfikacja metody konfiguracyjnej. Do prezentacji wykorzystałem kod autorskiej aplikacji. Jak widzimy zostały już stworzone metody przyznające uprawnienia dla poszczególnych grup użytkowników, natomiast jeśli chodzi o zmiany które musimy dokonać w przypadku customizacji strony logowania to do linii .formLogin() musimy dopisać kod .permitAll() co po prostu oznacza, że z funkcji logowania będą mogli korzystać wszystko użytkownicy aplikacji również goście co jest oczywiście zrozumiałe. Następnie definiujemy odnośnik pod którym ma znajdować się nasza strona logowania - .loginPage(“/tutaj dowolny odnośnik w naszym przypadku /login”), następnie definiujemy domyślną stronę która pojawi się po zalogowaniu .defaultSuccessUrl(“/”), i to wszystkie zmiany które musimy dokonać.

Własny formularz logowania - kontroler |Slajd x

Następnym krokiem jest stworzenie w klasie kontrolera odpowiedniego mapowania - nazwa kontrolera nie ma znaczenia, w swoim kodzie przykładowo wykorzystałem nazwę

IndexController. Jak widzimy metoda login() zwróci wartość “login” co tak naprawdę odnosi się do pliku login.html w którym będzie formularz logowania.

Własny formularz logowania - widok |Slajd x

Na samym końcu nie pozostaje nic innego jak stworzyć widok html, który będzie udostępniony użytkownikowi aplikacji do logowania. Na slajdzie przykładowy ekran logowania który wykorzystywałem w swojej aplikacji.

Zabezpieczenie aplikacji REST

Aby dodać Spring Security do projektu w Spring Boocie będziemy potrzebować trzech składników:

1. odpowiednie zależności w projekcie,
2. konfiguracja endpointów,
3. konfiguracja użytkowników.

Dodanie zależności do projektu

Dodawanie zależności jest bardzo proste, w przypadku gdy nasz projekt korzysta z Gradle, wystarczy dodać odpowiednią linijkę kodu w build.gradle. W Mavenie odpowiednikiem jest plik pom.xml.

W tym momencie cała aplikacja jest zabezpieczona przed nieupoważnionym dostępem, a jedyny sposób by się do niej dostać to skorzystanie z uwierzytelnienia Basic Auth i dwóch parametrów:

- nazwa użytkownika - domyślnie *user*
- hasło - wygenerowane w formacie UUID

Konfiguracja endpointów

Aby zabezpieczyć tylko niektóre endpointy należy nadpisać klasę WebSecurityConfigurerAdapter, dodając ją do konteksty Springa (za pomocą adnotacji @Configuration) i zdefiniować własne reguły autoryzacji.

Fragment kodu znajdujący się na slajdzie oznacza:

- requesty GET pod adres `/catalog` i jego podścieżki mają być dostępne dla wszystkich użytkowników - `mvcMatchers(HttpMethod.GET, "/catalog/**").permitAll()`
- pozostałe żądania - `anyRequest().authenticated()` wymagają bycia uwierzytelnionym
- dodatkowo pozwalamy na dostęp za pomocą Basic Auth - `.and().httpBasic()`

Konfiguracja użytkowników

Użytkowników można konfigurować na trzy sposoby:

1. w pamięci aplikacji,
2. w bazie danych aplikacji,
3. z zewnętrznych dostawców

Podczas prezentacji skupimy się na pierwszym sposobie.

Aby skonfigurować użytkowników w pamięci należy nadpisać kolejną metodę w klasie SecurityConfiguration. Za pomocą obiektu AuthenticationManagerBuilder możemy zdefiniować jakich użytkowników z jakimi rolami chcemy mieć w swoim systemie. W tym przypadku dodano dwóch użytkowników:

- john@example.org z hasłem xxx i rolą USER,
- admin z hasłem xxx i rolą ADMIN.

Zapis .password("{noop}xxx") oznacza, że nie korzystamy z żadnego (noop => no-operation) algorytmu szyfrującego hasła. Stąd przy próbie dostępu do endpointów w Basic Auth xxx przekazywane jest jako hasło.

Zabezpieczenie po roli

Mając już użytkowników z konkretnymi rolami, można je wykorzystać do zabezpieczenia dostępu do konkretnych endpointów w aplikacji. Możemy to osiągnąć korzystając z adnotacji `@Secured`. Do endpointu `/admin` będzie miał dostęp tylko administrator, a do endpointu `/orders` zarówno administrator jak i zwykły, uwierzytelniony użytkownik. Osoba anonimowa nie będzie mogła uzyskać odpowiedzi z systemu.

Aby adnotacja `@Secured` zadziałała, trzeba ją explicite włączyć w aplikacji za pomocą `@EnableGlobalMethodSecurity(securedEnabled = true)` w nadpisanej wcześniej klasie `WebSecurityConfigurerAdapter`.

OAuth 2.0

Framework pozwalający aplikacjom zewnętrznym na otrzymanie określonych zasobów na określony czas. Przykładem takiego udostępnienia zasobów może nadanie praw innym aplikacjom do odczytu danych z konta google, facebook lub innych platform obsługujących technologię OAuth. Przy nadaniu praw dostępu do zasobów dla użytkownika generowany jest token, dzięki któremu może uzyskać dostęp do zasobów.

Zasada działania OAuth 2.0

Najpopularniejsza metoda działania to Authorization Code Flow, zaangażowani w tym są klient, właściciel zasobu oraz serwer autoryzacyjny.

- Rejestracja klienta na serwerze autoryzacyjnym - generuje się para danych client id oraz client secret, co jest potrzebne później przy autoryzacji.
- Wysłanie zapytania o dostęp do zasobów - przesyłana jest prośba o przyznanie

dostępu do zasobów, np. próba dostępu do zasobów na dysku Google
- Nadanie klientowi odpowiednich praw do zasobów - właściciel zasobów nadaje odpowiednie uprawnienia
- Odesłanie do klienta kodu autoryzacji - dzięki temu oraz wcześniej wygenerowanemu client id orac client secret, dany użytkownik może dokonać autoryzacji.
- Przesłanie do serwera autoryzacji informacji o kodzie autoryzacji oraz identyfikator i klucz klienta
- Jeżeli autoryzacja się powiodła klient otrzymuje token
- Zapytanie o otrzymanie chronionych danych - jeżeli wszystko się powiodło to klient otrzymuje określone dane

Zasada działania OAuth 2.0

Jeżeli korzystamy z Spring Boot to dociągniecie zależności jest proste, wystarczy przy tworzeniu projektu dodać moduł odpowiedzialny za OAuth 2.0, lub w trakcie projektu samemu dodać odpowiednie dependency.

W przypadku braku Spring Boot konieczne jest dociągnięcie dwóch zależności *spring-security-oauth2-client* oraz *spring-security-oauth2-jose* .

Zasada działania OAuth 2.0

Podstawowa konfiguracja jest bardzo prosta, zakłada że w celu uzyskania danych przy każdym request'e użytkownik musi być autoryzowany, do autoryzacji jeśt wykorzystywany domyślny moduł logowania OAuth. Klasa konfiguracyjna musi rozszerzać klasę WebSecurityConfigurerAdaoter oraz musi być oznaczona adnotacją "@Confguration". Musi się tam znajdować podstawowa metoda configurer(HttpSecurity http). Oauth2Login() odpowiada za autoryzacje, podobne jest do httpBasic() oraz formLogin().

Własna konfiguracja OAuth 2.0

Własna konfiguracja nie różni się specjalnie od konfiguracji podstawowego Spring Security. Tak samo można użyć wcześniej już omówionych metod, przykłady:

- loginPage()
- defaultSuccessUrl()
- failureUrl()
- successHandler()
- failureHandler()

Korzyści oraz wady OAuth 2.0

Zalety:

- Klienci nie mają styczności z danymi uwierzytelniającymi użytkowników
- Wykorzystanie jednego serwera autoryzacyjnego w celu chronienia różnych zasobów
- Ograniczenie ilości kont w różnych aplikacjach, autoryzacja jednym kontem.
- Popularność w innych serwisach.

Wady:

- Bez zadbania o dobre szyfrowanie system jest podatny na wyciek danych autoryzacyjnych
- Dodatkowa ilość zapytań w celu uzyskania informacji o użytkowniku
- Każdy projekt potrzebuje własnej implementacji, nie ma wspólnego formatu.
- Po kradzieży tokenu, atakujący dostaje dostęp do chronionych zasobów.

Reactive Support

Systemy reaktywne mają pewne cechy, które czynią je idealnymi dla obciążeń roboczych o małych opóźnieniach i dużej przepustowości. Project Reactor i Spring portfolio współpracują ze sobą, aby umożliwić programistom tworzenie reaktywnych systemów klasy korporacyjnej, które są responsywne, odporne, elastyczne i sterowane komunikatami.

Czym jest przetwarzanie reaktywne i dlaczego warto je stosować?

Przetwarzanie reaktywne to paradygmat, który umożliwia programistom tworzenie asynchronicznych aplikacji, które mogą obsługiwać kontrolę przepływu (flow control).

Systemy reaktywne lepiej wykorzystują nowoczesne procesory. Ponadto uwzględnienie przeciwciśnienia (back-pressure) w programowaniu reaktywnym zapewnia lepszą elastyczność między osobnymi komponentami.

Przykład Reactive Support

Na widocznym przykładzie można zobaczyć, że podobnie jak w innych przypadkach, które nie były reaktywne, konfigurujemy określone odwzorowania za pomocą metody springSecurityFilterChain. Powyższy kod można dodać do swojej konfiguracji dla Spring Security, dokładnie w ten sam sposób jak dla stosu serwletów. Pełny przykład tego, jak zabezpieczyć reaktywną aplikację WebFlux, można zobaczyć na oficjalnym przykładzie Spring security dostępnym na GitHub.

Reaktywne mikroserwisy z Spring Boot

Portfel Spring zapewnia dwa równoległe stosy. Jeden oparty jest na Servlet API z konstrukcjami Spring MVC i Spring Data. Drugi to w pełni reaktywny stos, który wykorzystuje reaktywne repozytoria Spring WebFlux i Spring Data. W obu przypadkach Spring Security zapewnia natywną obsługę obu stosów.

Spring WebFlux to nieblokująca się platforma internetowa zbudowana od podstaw w celu wykorzystania wielordzeniowych procesorów nowej generacji i obsługi ogromnej liczby jednoczesnych połączeń.

Spring MVC jest zbudowany na Servlet API i wykorzystuje architekturę synchronicznego blokowania I/O z modelem jednego żądania na wątek