Spring Security i JWT

Cześć

Maciej Koziara

Plan spotkania

1. Spring Security
2. JWT
3. Live coding

Q & A - po każdym module

1. Spring Security

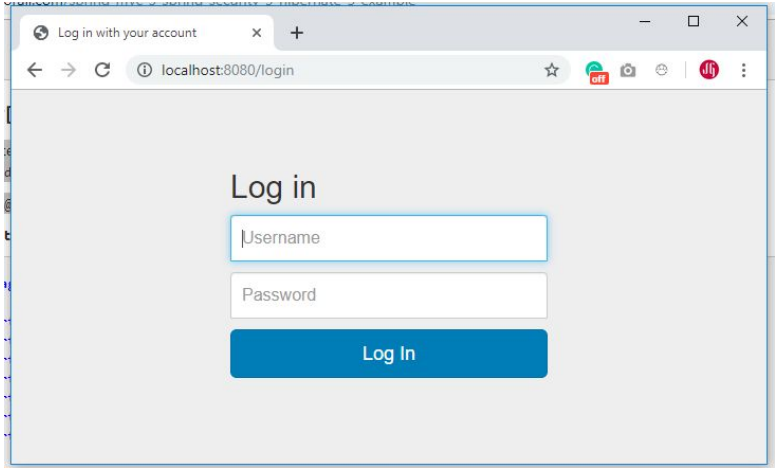
Czyli jak ta magia w ogóle działa?

Czym jest Spring Security?

1. Moduł frameworka Spring pozwalający na łatwe zarządzanie wszelkimi aspektami szeroko pojętego Security
2. Pozwala na prostą konfigurację uwierzytelniania i autoryzacji użytkownika.
3. Integruje się z wieloma systemami i standardami służącymi do bezpiecznego uwierzytelnienia użytkownika.

Przykład konfiguracji

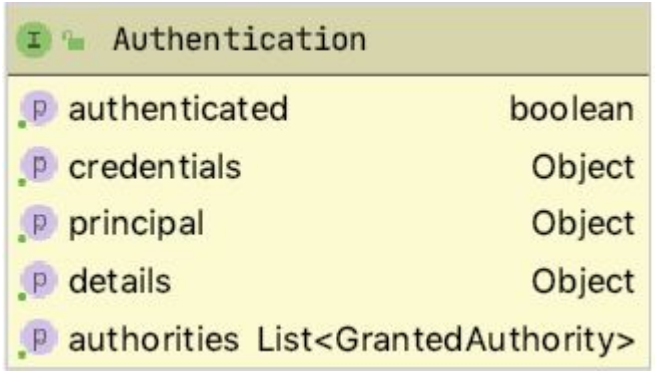
```java
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.formLogin();

        http.authorizeRequests()
                .antMatchers( ...antPatterns: "/api/public/**").permitAll()
                .antMatchers( ...antPatterns: "/login").permitAll()
                .anyRequest().authenticated();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth
                .inMemoryAuthentication()
                .withUser( username: "admin").password("{noop}admin").roles("ADMIN");
    }
}
```



Kluczowe pojęcia

1. **Authentication (uwierzytelnianie)** - potwierdzanie tożsamości użytkownika poprzez np. walidację nazwy użytkownika i hasła.
2. **Authorization (autoryzacja)** - sprawdzenie czy uwierzytelniony ma dostęp do danego zasobu.

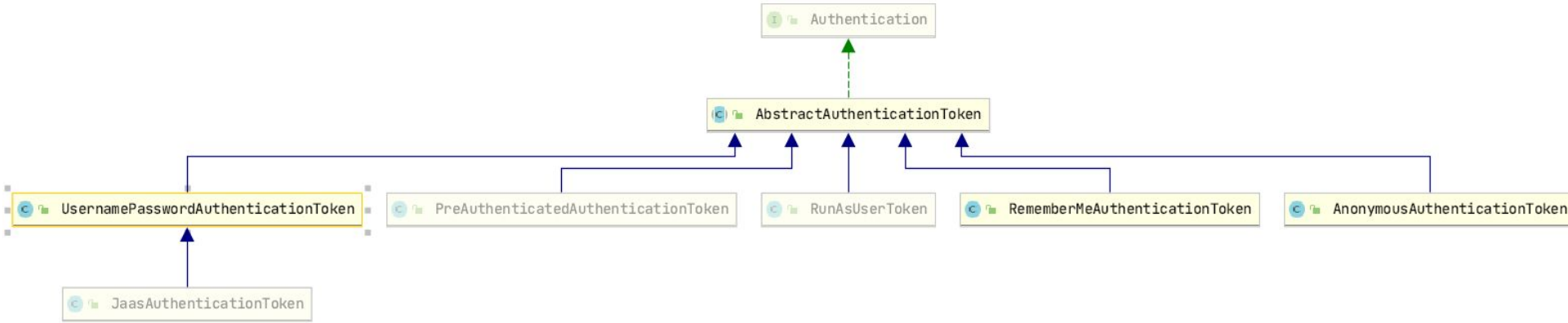
Interfejs Authentication



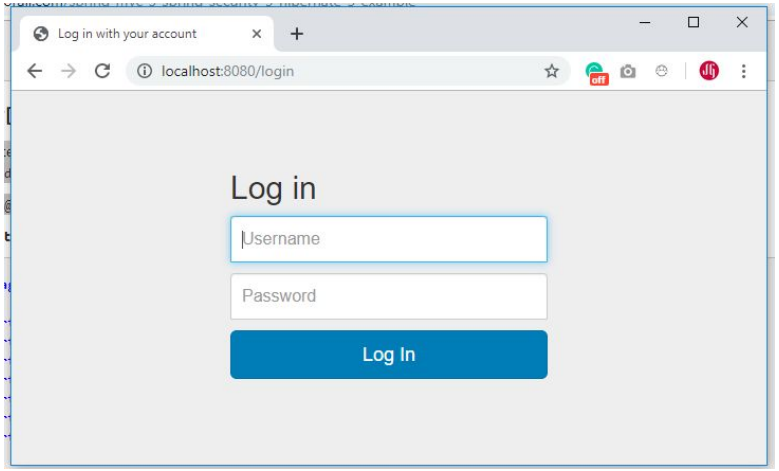
- Centralny interfejs reprezentujący prośbę o uwierzytelnienie przez użytkownika
- **principal** - obiekt pozwalający na określenie jakiego użytkownika dotyczy weryfikacja, np. **username**
- **credentials** - obiekt potwierdzający, że **principal** jest poprawny, np. Hasło
- **authenticated** - określa czy obiekt został już uwierzytelniony i nie jest potrzebna powtórna weryfikacja
- **authorities** - lista uprawnień, która została przyznana użytkownikowi, np. rola

Interfejs Authentication

1. We frameworku reprezentowany przez tokeny
2. Principal zmienia się w czasie

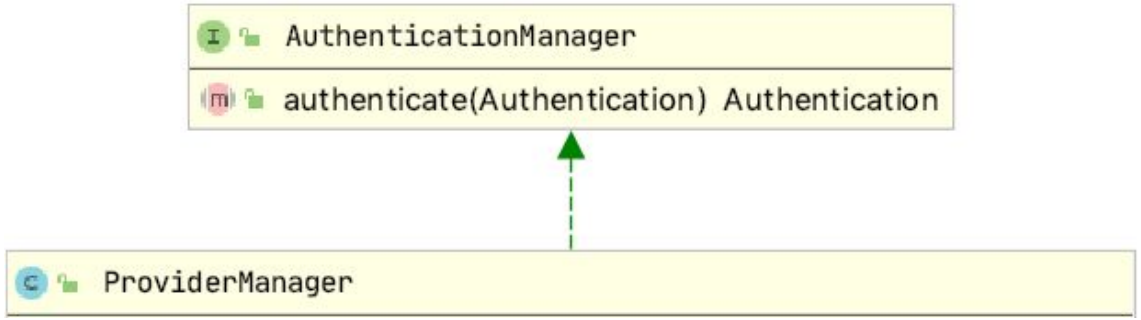


UsernamePasswordAuthenticationToken



1. Reprezentuje próbę logowania użytkownika poprzez formularz logowania
2. Wypełniony na podstawie danych pobranych z requesta
3. **principal** - na początku nazwa użytkownika
4. **credentials** - wpisane hasło
5. **principal** - po poprawnym uwierzytelnieniu - wszystkie informacje o użytkowniku

Interfejs AuthenticationManager

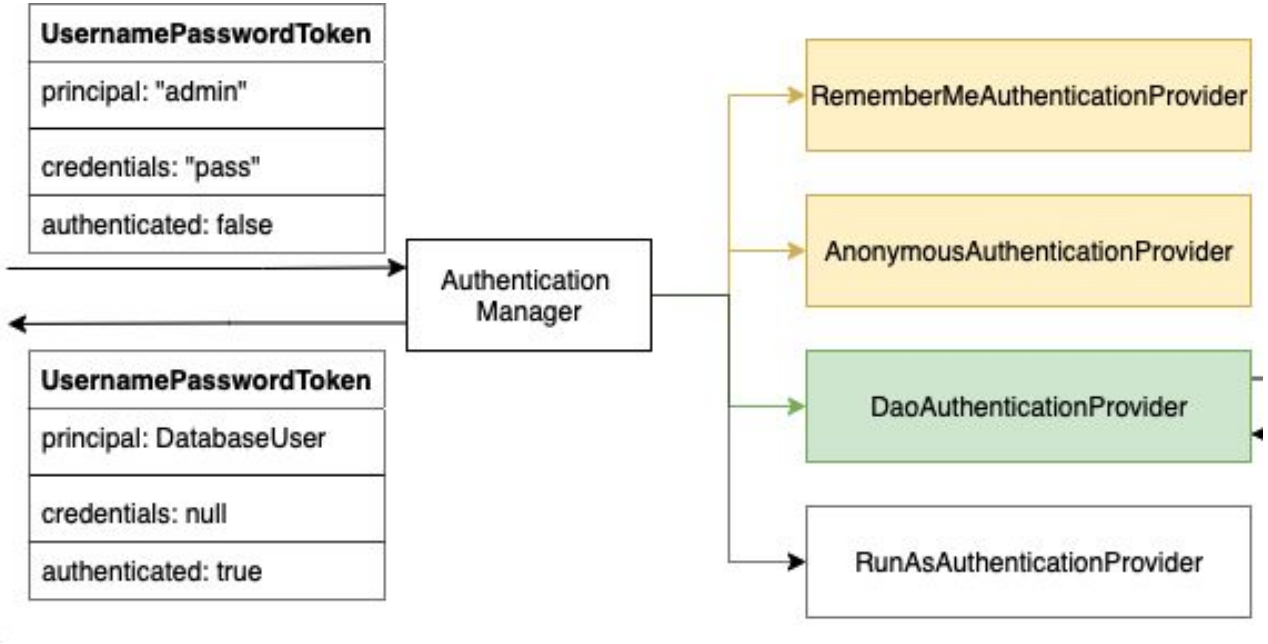


1. Jedno zadanie - uwierzytelnienie tokenu.
2. Metoda authenticate - przyjmuje nieuwierzytelniony token i ma dwie możliwości:
   a. Zwrócić token uwierzytelniony
   b. Rzucić wyjątek oznaczający, że uwierzytelnienie nie powiodło się

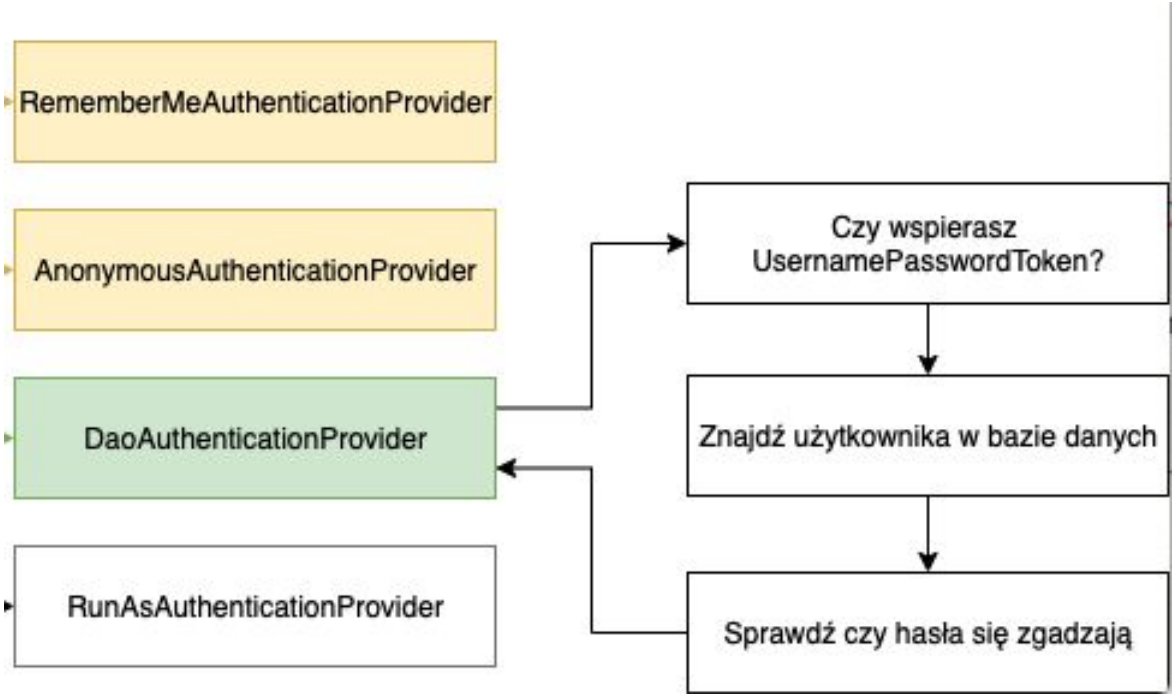
Jak AuthenticationManager uwierzytelnia token?

1. Oddelegowuje sprawdzenie do klas **AuthenticationProvider.**
2. Każdy **AuthenticationProvider** jest wyspecjalizowany w jednym typie uwierzytelnienia, np.
    a. **DaoAuthenticationProvider** - obsługuje username / password
    b. **BearerTokenAuthenticationProvider** - obsługuje tokeny
3. Każdy **AuthenticationProvider** ma trzy opcje:
    a. Zaakceptować token
    b. Odrzucić token rzucając wyjątek **AuthenticationException**
    c. Powiedzieć, że nie wspiera danego tokenu

Klasyczny proces uwierzytelnienia użytkownika



Co się dzieje w DaoAuthenticationProvider?

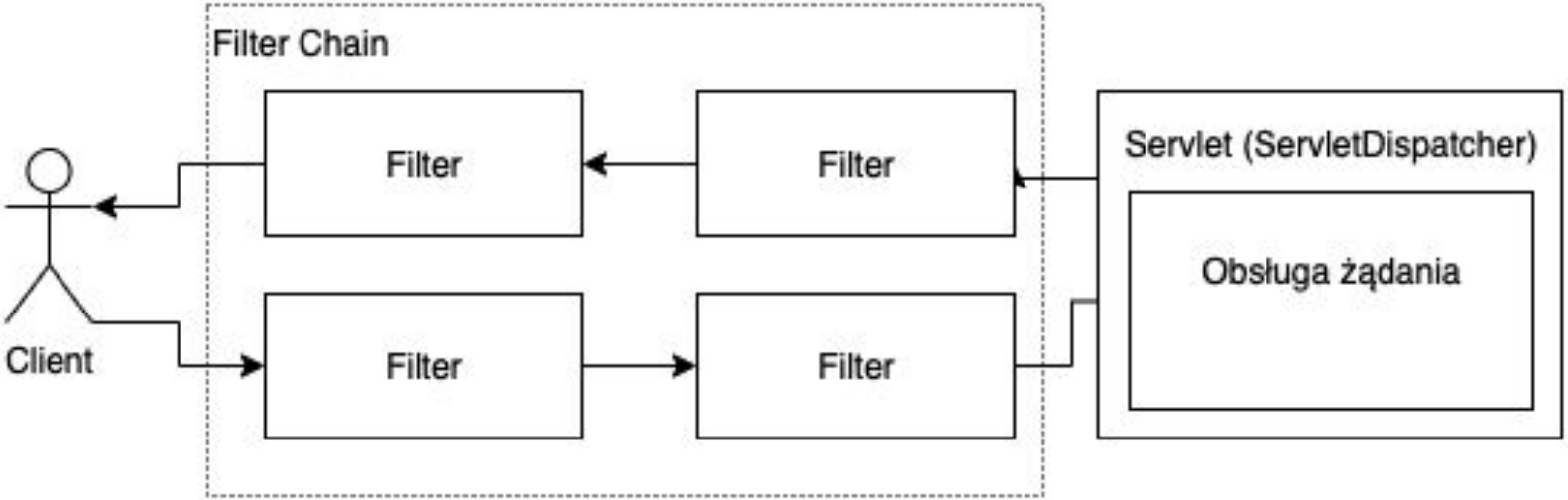


Q&A I

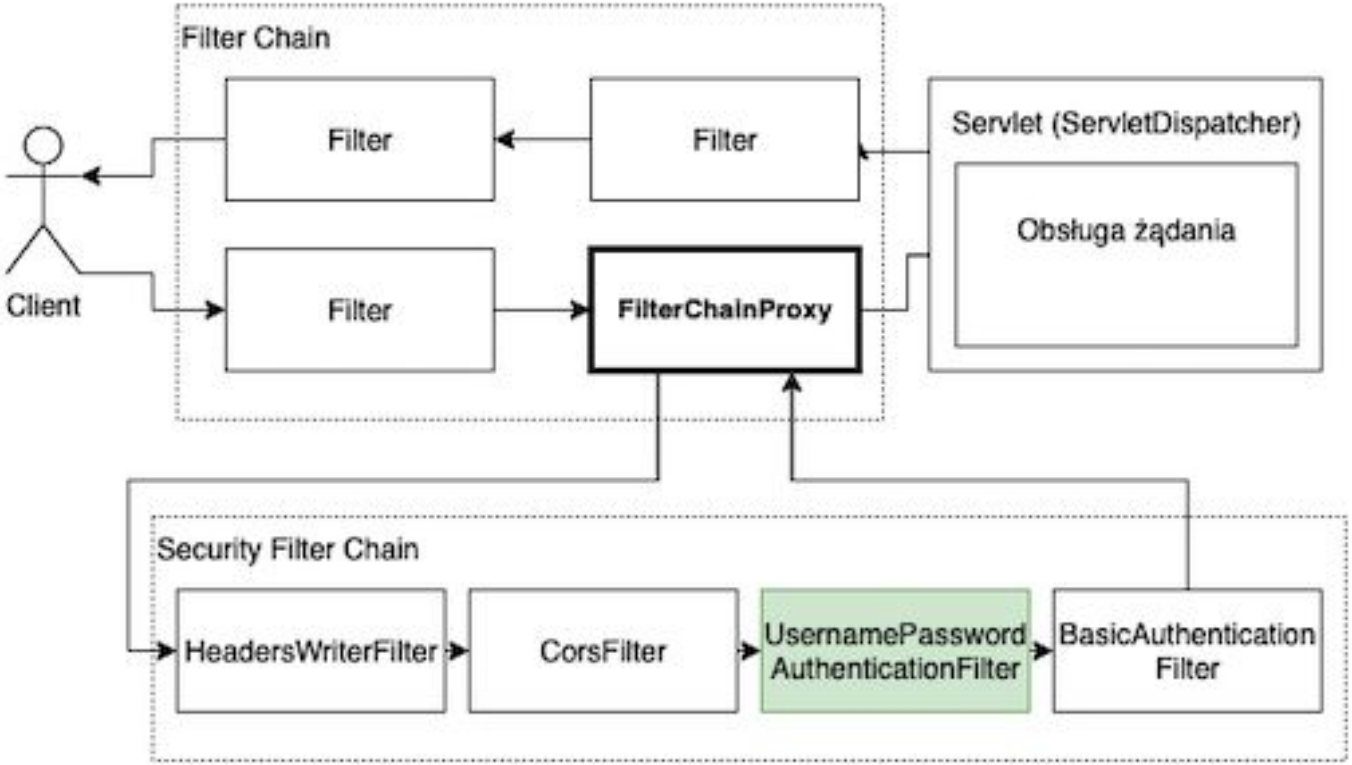
Skąd bierze się wypełniony token?

1. Aplikacja Springowa uruchamia się na serwerach, które wspierają **Java Servlet Specification**, np. **Tomcat**.
2. Częścią specyfikacji **Servlet**, są tak zwane **Servlet Filter**
3. **Servlet Filter** - obiekt pozwalający na przechwycenie i modyfikację żądania przed lub po jego obsłudze

Ścieżka żądania w środowisku servletowym

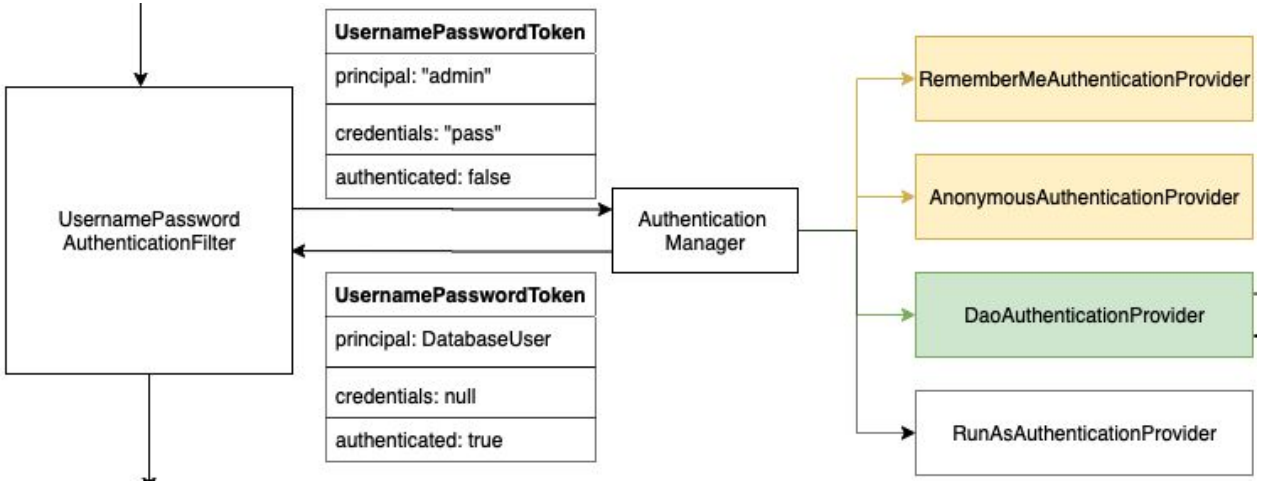


Ścieżka żądania ze Spring Security



UsernamePasswordAuthenticationFilter

1. Pobiera z żądania parametry username i password.
2. Tworzy **UsernamePasswordAuthenticationToken**.
3. Uwierzytelnia token poprzez AuthenticationManager.
4. Kontynuuje żądanie przez kolejne filtry.



Kilka dodatkowych pojęć

1. **SecurityContext** - po udanym uwierzytelnieniu przechowuje obiekt Authentication.
2. **SecurityContextHolder** - przechowuje SecurityContext, dzięki czemu jest on łatwo dostępny w wątku obsługującym żądanie.
3. **AuthenticationSuccessHandler** - kod wywołany w momencie gdy uwierzytelnienie się powiedzie, np. przekierowanie na poprzednią stronę.
4. **AuthenticationFailureHandler** - kod wywołany w momencie gdy uwierzytelnienie się nie powiedzie, np. po którejś nieudanej próbie logowania poinformowanie klienta, że należy wyświetlić captcha.

Q&A II

2. JWT

Czyli dlaczego miałbym tego użyć?

Czym są Json Web Tokens

- Otwarty standard definiujący kompaktowy i niezależny sposób bezpiecznego przesyłania informacji między stronami jako obiekt JSON.
- Przekazywane informacje są bezpieczne ponieważ zostały podpisane cyfrowo i nie mogą ulec zmianie bez znajomości klucza użytego do podpisu

Jak wygląda JWT?

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
  "sub": "1234567890",
  "iat": 1516239022,
  "exp": 1516239066,
  "name": "Maciek",
  "app_role": "ADMIN"
}
```

VERIFY SIGNATURE

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

Header

Zawiera podstawowe informacje o tokenie, przede wszystkim algorytm, który został użyty do wygenerowania sygnatury.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
```

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

Payload

Przechowuje dane (tzw. claims), dotyczące uwierzytelnionej jednostki (zwykle użytkownik).

Dane mogą:

- pochodzić ze standardu (np. exp)
- być dodane przez nas (np. app_role)


```
.eyJzdWIiOiIxMjM0NTY3ODkwIiwiaWF0IjoxNTE
2MjM5MDIyLCJleHAiOjE1MTYyMzkwNjYsIm5hbWU
iOiJNYWNpZWsiLCJhcHBfcm9sZSI6IkFETUlOIn0
```


```
{
  "sub": "1234567890",
  "iat": 1516239022,
  "exp": 1516239066,
  "name": "Maciek",
  "app_role": "ADMIN"
}
```

Signature

Pozwala na weryfikację tego iż wiadomość nie została zmieniona po drodze.

Secret użyty do podpisu powinien być długi i skomplikowany, gdyż jego odgadnięcie pozwoli atakującemu podszyć się pod dowolnego użytkownika!

```
.g_9J_Xvzs2P7Rk6UShMImHaxTfn-
VNxweD2wPXEwvXI
```

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

JWT - krótkie FAQ

P: Kto może odczytać zawartość JWT?

O: Każdy może odczytać jego zawartość dlatego w claimach nie należy przechowywać danych poufnych.

P: Czy to jest bezpieczne?

O: Tak, ponieważ bez znajomości klucza (secret) nie jesteśmy w stanie zmienić zawartości tokena.

P: Gdzie przechowywać na frontendzie token?

O: Nie ma reguły. U mnie sprawdził się local storage.

JWT - krótkie FAQ

P: W jaki sposób wysłać token na serwer?

O: Robi się to korzystając z nagłówka Authorization, poprzedzając token przedrostkiem Bearer (tzw. Bearer token).

P: Kiedy warto pomyśleć nad użyciem tego w swoim projekcie?

O: 1) Kiedy z jakiegoś powodu zależy nam na bezstanowym backendzie

2) Kiedy problematyczne jest użycie ciasteczek do trzymania sesji (np. tworzymy backend pod aplikacje mobilne)

3) W środowisku mikroserwisowym w momencie gdy potrzebujemy wysyłać kontekst użytkownika pomiędzy serwisami

Q&A III

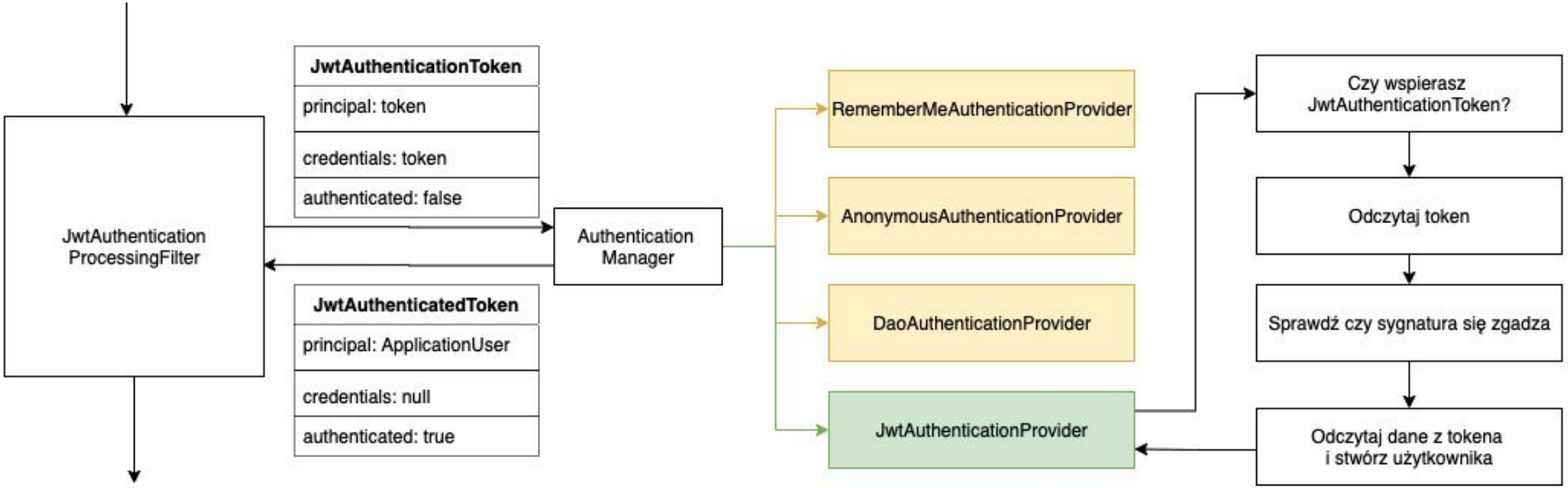
3. Live coding

Czyli Spring Security w praktyce.

Co zbudujemy?

- Rozszerzymy Spring Security o możliwość uwierzytelnienia przy pomocy JWT, która nie przychodzi w standardzie.
- Czego będziemy potrzebować:
  - Implementacja **Filter** - JwtAuthenticationProcessingFilter
  - Dwie implementacje **Authentication:**
    - JwtAuthenticationToken - przed uwierzytelnieniem
    - JwtAuthenticatedUser - po uwierzytelnieniu
  - Implementacja **AuthenticationProvider** - JwtAuthenticationProvider
  - Implementacja **AuthenticationSuccessHandler**
    - JwtLoginSuccessHandler

Diagram rozwiązania



O czym nie powiedziałem?

- W jaki sposób działa autoryzacja, czyli okreslanie czy użytkownik ma dostęp do danego zasobu
- W jaki sposób obsługiwane są poszczególne wyjątki rzucane przez Spring Security
- W jaki sposób chronić poszczególne endpointy, metody i obiekty przed nieautoryzowanym dostępem
- Jakie możliwości konfiguracyjne dostarcza sam Spring Security
- Jakie dostosować autoryzację przy użyciu bazy danych do naszych wymagań
- Z jakimi systemami integruje się Spring Security

Q&A

Zapytaj o szkolenie dla Twojego zespołu

DOMINIKA KALINA

dominika.kalina@infoshareacademy.com

tel. 730-830-801

szkoleniazdalne.infoshareacademy.com

Każdy moment jest dobry na rozwój kompetencji!

infoShare ACADEMY

Dzieki

email: maciej.koziara@yahoo.pl