OPTIMUS
Przedsiębiorstwo Optimus Sp. z o.o.
ul. Jana Pawła II 84K
98-200 Sieradz
+48 43 827 10 07
+48 501 538 149
optsier@onet.pl
NIP 8270018155
REGON 730173184
KRS 0000203770

AIS DT
Zabezpieczanie aplikacji i usług
Centrum monitorowania AIS.ID2

2020.09.30

Spis treści

Fundusze Europejskie Program Regionalny
promuje łódzkie
Unia Europejska Europejski Fundusz Rozwoju Regionalnego

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

1. Wprowadzenie

Serwer zarządzania tożsamością i dostępem *Centrum monitorowania* obsługuje zarówno *OpenID Connect* (rozszerzenie *OAuth 2.0*), jak i *SAML 2.0*. Podczas zabezpieczania klientów i usług pierwszą rzeczą, którą musisz zdecydować, jest to, z którego z nich będziesz korzystać. Jeśli chcesz, możesz także zabezpieczyć niektóre za pomocą OpenID Connect, a inne za pomocą SAML.

Aby zabezpieczyć klientów i usługi, będziesz także potrzebować adaptera lub biblioteki dla wybranego protokołu. Centrum monitorowania posiada własne adaptery dla wybranych platform, ale możliwe jest również użycie ogólnych bibliotek *OpenID Connect Relying Party* i bibliotek *SAML Service Provider*.

1.1. Co to są adaptery klienta?

Adaptery klienckie to biblioteki, dzięki którym bardzo łatwo jest zabezpieczyć aplikacje i usługi. Nazywamy je raczej adapterami niż bibliotekami, ponieważ zapewniają ścisłą integrację z platformą i frameworkiem. To sprawia, że nasze adaptery są łatwe w użyciu i wymagają mniej kodu niż jest to zwykle wymagane przez uniwersalną bibliotekę.

1.2. Obsługiwane platformy

1.2.1. OpenID Connect

1) Java
   a) JBoss EAP
   b) WildFly
   c) Fuse
   d) Tomcat
   e) Jetty 9
   f) Servlet Filter
   g) Spring Boot
   h) Spring Security
2) JavaScript (client-side)
   a) JavaScript
3) Node.js (server-side)
   a) Node.js
4) C#
   a) OWIN (community)
5) Python
   a) oidc (generic)
6) Android
   a) AppAuth (generic)
7) iOS
   a) AppAuth (generic)
8) Apache HTTP Server
   a) mod_auth_openidc

1.2.2. SAML

1) Java
   a) JBoss EAP
   b) WildFly
   c) Tomcat
   d) Jetty
2) Apache HTTP Server

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

a) mod_auth_mellon

1.3. Obsługiwane protokoły

1.3.1. OpenID Connect

OpenID Connect (OIDC) to protokół uwierzytelniania będący rozszerzeniem OAuth 2.0. Podczas gdy OAuth 2.0 stanowi jedynie strukturę do budowania protokołów autoryzacji i jest niekompletny, OIDC jest pełnoprawnym protokołem uwierzytelniania i autoryzacji. OIDC również intensywnie wykorzystuje zestaw standardów Json Web Token (JWT). Standardy te definiują format JSON tokena tożsamości oraz sposoby cyfrowego podpisywania i szyfrowania tych danych w kompaktowy i przyjazny dla sieci oraz aplikacji webowych sposób.

Istnieją dwa typy przypadków użycia podczas korzystania z OIDC. Pierwszy to aplikacja, która prosi serwer Keycloak o uwierzytelnienie użytkownika. Po pomyślnym zalogowaniu aplikacja otrzyma token tożsamości i token dostępu. Token tożsamości zawiera informacje o użytkowniku, takie jak nazwa użytkownika, adres e-mail i inne informacje o profilu. Token dostępu jest cyfrowo podpisany przez strefę (*ang. realm*) i zawiera informacje o dostępie (takie jak odwzorowania ról użytkownika), których aplikacja może użyć do ustalenia, jakie zasoby użytkownik może uzyskać w aplikacji.

Drugi typ przypadków użycia dotyczy klienta, który chce uzyskać dostęp do usług zdalnych. W takim przypadku klient prosi Keycloak o uzyskanie tokena dostępu, którego może użyć do wywołania innych usług zdalnych w imieniu użytkownika. Keycloak uwierzytelnia użytkownika, a następnie prosi użytkownika o zgodę na udzielenie dostępu do klienta, który o to poprosił. Następnie klient otrzymuje token dostępu. Ten token dostępu jest cyfrowo podpisany przez strefę (ang. realm). Klient może wykonywać wywołania REST w zdalnych usługach przy użyciu tego tokena dostępu. Usługa REST wyodrębnia token dostępu, weryfikuje podpis tokena, a następnie decyduje na podstawie informacji o dostępie w tokenie, czy przetworzyć żądanie.

1.3.2. SAML 2.0

SAML 2.0 jest podobną specyfikacją do OIDC, ale o wiele starszą i bardziej dojrzałą. Ma swoje korzenie w SOAP i wielu specyfikacjach WS- *, więc jest bardziej rozwlekła niż OIDC. SAML 2.0 to przede wszystkim protokół uwierzytelniania, który działa poprzez wymianę dokumentów XML między serwerem uwierzytelniającym a aplikacją. Podpisy i szyfrowanie XML służą do weryfikacji żądań i odpowiedzi.

W Keycloak SAML obsługuje dwa typy przypadków użycia: aplikacje przeglądarki i wywołania REST.

Istnieją dwa typy przypadków użycia podczas korzystania z SAML. Pierwszy to aplikacja, która prosi serwer Keycloak o uwierzytelnienie użytkownika. Po pomyślnym zalogowaniu aplikacja otrzyma dokument XML zawierający coś o nazwie asercja SAML, która określa różne atrybuty dotyczące użytkownika. Ten dokument XML jest podpisany cyfrowo przez domenę i zawiera informacje o dostępie (takie jak odwzorowania ról użytkownika), których aplikacja może użyć do ustalenia, jakie zasoby użytkownik może uzyskać w aplikacji.

Drugi typ przypadków użycia dotyczy klienta, który chce uzyskać dostęp do usług zdalnych. W takim przypadku klient prosi Keycloak o uzyskanie asercji SAML, której może użyć do wywołania innych usług zdalnych w imieniu użytkownika.

1.3.3. OpenID Connect versus SAML

Wybór między OpenID Connect a SAML to nie tylko dylemat, czy użyć nowszego protokołu (OIDC) zamiast starszego, bardziej dojrzałego protokołu (SAML).

W większości przypadków Keycloak zaleca stosowanie OIDC.

SAML jest nieco bardziej szczegółowy niż OIDC.

Oprócz szczegółowości wymienianych danych, jeśli porównasz specyfikacje, przekonasz się, że OIDC został zaprojektowany do pracy z Internetem, a SAML został zmodernizowany do pracy w sieci. Na przykład OIDC jest również bardziej odpowiedni dla aplikacji HTML5 / JavaScript, ponieważ łatwiej jest go wdrożyć po stronie klienta

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

niż SAML. Ponieważ tokeny są w formacie JSON, łatwiej je konsumować przez JavaScript. Znajdziesz również kilka ciekawych funkcji, które ułatwiają wdrażanie zabezpieczeń w aplikacjach internetowych. Na przykład sprawdź sztuczkę iframe, której używa specyfikacja, aby łatwo ustalić, czy użytkownik jest nadal zalogowany, czy nie.

SAML ma jednak swoje zastosowania. W miarę ewolucji specyfikacji OIDC widać, że wprowadzają one coraz więcej funkcji, które SAML ma od lat. Często widzimy, że ludzie wybierają SAML zamiast OIDC ze względu na wrażenie, że jest on bardziej dojrzały, a także dlatego, że mają już zabezpieczone aplikacje w tym protokole.

2. OpenID Connect

W tej sekcji opisano, w jaki sposób można zabezpieczyć aplikacje i usługi za pomocą *OpenID Connect* i adapterów Keycloak lub ogólnych bibliotek *OpenID Connect Relying Party*.

2.1. Adaptery Java

Keycloak jest wyposażony w szereg różnych adapterów do aplikacji Java. Wybór odpowiedniego adaptera zależy od platformy docelowej.

Wszystkie adaptery Java współużytkują zestaw typowych opcji konfiguracji opisanych w rozdziale Konfiguracja adapterów Java.

2.1.1. Konfiguracja adaptera Java

Każdy adapter Java obsługiwany przez Keycloak można skonfigurować za pomocą prostego pliku JSON, który może wyglądać tak:

```
{
  "realm" : "demo",
  "resource" : "customer-portal",
  "realm-public-key" : "MIGfMA0GCSqGSIb3D...31LwIDAQAB",
  "auth-server-url" : "https://localhost:8443/auth",
  "ssl-required" : "external",
  "use-resource-role-mappings" : false,
  "enable-cors" : true,
  "cors-max-age" : 1000,
  "cors-allowed-methods" : "POST, PUT, DELETE, GET",
  "cors-exposed-headers" : "WWW-Authenticate, My-custom-exposed-Header",
  "bearer-only" : false,
  "enable-basic-auth" : false,
  "expose-token" : true,
  "verify-token-audience" : true,
  "credentials" : {
    "secret" : "234234-234234-234234"
  },
  "connection-pool-size" : 20,
  "disable-trust-manager": false,
  "allow-any-hostname" : false,
  "truststore" : "path/to/truststore.jks",
  "truststore-password" : "geheim",
  "client-keystore" : "path/to/client-keystore.jks",
  "client-keystore-password" : "geheim",
  "client-key-password" : "geheim",
  "token-minimum-time-to-live" : 10,
  "min-time-between-jwks-requests" : 10,
  "public-key-cache-ttl": 86400,
  "redirect-rewrite-rules" : {
    "^/wsmaster/api/(.*)$" : "/api/$1"
```

```
    }
  }
```

Możesz użyć operatora `${…}` do wstawienia właściwości systemowych Java. Na przykład `${jboss.server.config.dir}` zostałby zastąpiony przez `/path/to/Keycloak`. Zastąpienie zmiennych środowiskowych jest również obsługiwane za pomocą przedrostka `env`, np. `${env.MY_ENVIRONMENT_VARIABLE}`.

Początkowy plik konfiguracyjny można uzyskać z konsoli administracyjnej. Można to zrobić, otwierając konsolę administracyjną, wybierając z menu Klientów i klikając odpowiedniego klienta. Po otwarciu strony klienta kliknij kartę Instalacja i wybierz Keycloak OIDC JSON.

Oto opis każdej opcji konfiguracji:

Parametr	Opis
realm	Nazwa strefy (ang. realm). WYMAGANY.
resource	Identyfikator klienta aplikacji. Każda aplikacja ma identyfikator klienta używany do jej identyfikacji. WYMAGANY.
realm-public-key	Klucz publiczny strefy (ang. realm) w formacie PEM. Można go uzyskać z konsoli administracyjnej. OPCJONALNY, nie zaleca się jego ustawiania. Jeśli nie jest ustawiony, adapter pobierze aktualne klucze z Keycloak i będzie je pobierać ponownie, gdy zajdzie taka potrzeba (np. Keycloak wygeneruje nowe klucze). Jeśli jednak ustawiono klucz ręcznie, adapter nigdy nie pobierze nowych kluczy z Keycloak, więc gdy Keycloak odnowi klucze, to adapter przestanie działać.
auth-server-url	Podstawowy adres URL serwera Keycloak. Wszystkie inne strony Keycloak i punkty końcowe usługi REST są pochodną tego adresu. Zwykle ma postać `https://host:port/auth`. WYMAGANY.
ssl-required	Zapewnia, że cała komunikacja do i z serwera Keycloak odbywa się przez HTTPS. W produkcji należy ustawić na wartość `'all'` (wszystko). Wartością domyślną jest wartość `'external'`, co oznacza, że HTTPS jest domyślnie wymagany w przypadku żądań zewnętrznych. OPCJONALNY. Prawidłowe wartości to `'all'`, `'external'` and `'none'`.
confidential-port	Poufny port używany przez serwer Keycloak do bezpiecznych połączeń przez SSL/TLS. OPCJONALNY. Wartość domyślna to 8443.
use-resource-role-mappings	Jeśli ustawiono wartość `true`, adapter będzie szukał wewnątrz tokena mapowań do ról dla użytkownika. Jeśli `false`, adapter będzie role sprawdzał na poziomie strefy (*ang. realm*). OPCJONALNY. Wartość domyślna to `false`.
public-client	Jeśli ustawiono wartość `true`, adapter nie będzie wysyłał poświadczeń klienta do Keycloak. OPCJONALNY. Wartość domyślna to `false`.
enable-cors	Parametr włącza obsługę CORS: poprzedzające żądania CORS, przeszukiwanie tokenów dostępu, aby ustalić prawidłowe pochodzenie. OPCJONALNY. Wartość domyślna to `false`.
cors-max-age	Jeśli CORS jest włączony, parametr ustawia wartość nagłówka `Access-Control-Max-Age`. OPCJONALNY. Jeśli nie jest ustawiony, to ten nagłówek nie jest zwracany w odpowiedziach CORS.
cors-allowed-methods	Jeśli CORS jest włączony, parametr ustawia wartość nagłówka `Access-Control-Allow-Methods`. Powinien to być ciąg oddzielony przecinkami.

	OPCJONALNY. Jeśli nie jest ustawiony, to ten nagłówek nie jest zwracany w odpowiedziach CORS.
cors-allowed-headers	Jeśli CORS jest włączony, parametr ustawia wartość nagłówka `Access-Control-Allow-Headers`. Powinien to być ciąg oddzielony przecinkami. OPCJONALNY. Jeśli nie jest ustawiony, to ten nagłówek nie jest zwracany w odpowiedziach CORS.
cors-exposed-headers	Jeśli CORS jest włączony, parametr ustawia wartość nagłówka `Access-Control-Expose-Headers`. Powinien to być ciąg oddzielony przecinkami. OPCJONALNY. Jeśli nie jest ustawiony, to ten nagłówek nie jest zwracany w odpowiedziach CORS.
bearer-only	Jeśli ta opcja jest włączona, adapter nie będzie próbował uwierzytelniać użytkowników, a jedynie weryfikować tokeny nośnika. Dla usług należy ustawić wartość `true`. OPCJONALNY. Wartość domyślna to `false`.
autodetect-bearer-only	To ustawienie powinno mieć wartość `true`, jeśli aplikacja obsługuje zarówno aplikację internetową, jak i usługi sieciowe (np. SOAP lub REST). Pozwala przekierowywać nieuwierzytelnionych użytkowników aplikacji internetowej na stronę logowania Keycloak, ale wysyłać też kod błędu HTTP 401 do nieuwierzytelnionych klientów SOAP lub REST, ponieważ nie rozumieliby przekierowania na stronę logowania. Keycloak automatycznie wykrywa klientów SOAP lub REST na podstawie typowych nagłówków, takich jak X-Requested-With, SOAPAction lub Accept. Wartość domyślna to `false`.
enable-basic-auth	Oznacza to, że adapter obsługuje również podstawowe uwierzytelnianie. Jeśli ta opcja jest włączona, należy również podać klucz tajny. OPCJONALNY. Wartość domyślna to `false`.
expose-token	Jeśli `true`, to uwierzytelniony klient przeglądarki (poprzez wywołanie JavaScript HTTP) może uzyskać podpisany token dostępu poprzez adres URL `root/k_query_bearer_token`. OPCJONALNY. Wartość domyślna to `false`.
credentials	Podaj poświadczenia aplikacji. Jest to notacja obiektowa, w której kluczem jest typ referencji. Obecnie obsługiwane jest `password` (hasło) i `jwt`. WYMAGANY tylko w przypadku klientów z dostępem poufnym typu `Confidential`.
connection-pool-size	Ta opcja konfiguracji określa liczbę połączeń z serwerem Keycloak, które powinny zostać połączone. OPCJONALNY. Wartość domyślna to `20`.
disable-trust-manager	Jeśli serwer Keycloak wymaga HTTPS i ta opcja konfiguracji jest ustawiona na wartość `true`, nie trzeba określać magazynu zaufanych certyfikatów. To ustawienie powinno być używane tylko podczas programowania (nigdy w systemach produkcyjnych), ponieważ spowoduje to wyłączenie weryfikacji certyfikatów SSL. OPCJONALNY. Wartość domyślna to `false`.
allow-any-hostname	Jeśli serwer Keycloak wymaga HTTPS, a ta opcja konfiguracji ma wartość `true`, certyfikat serwera Keycloak jest sprawdzany za pośrednictwem magazynu zaufanych certyfikatów, ale sprawdzanie nazwy hosta nie jest wykonywane. To ustawienie powinno być używane tylko podczas programowania (nigdy w systemach produkcyjnych), ponieważ spowoduje to wyłączenie weryfikacji certyfikatów SSL. To ustawienie może być przydatne w środowiskach testowych. OPCJONALNY. Wartość domyślna to `false`.
proxy-url	Adres URL serwera proxy HTTP, jeśli jest używany.
truststore	Wartością jest ścieżka do pliku zaufanych certyfikatów. Ścieżka może zostać poprzedzona dyrektywą `classpath:`. Służy do wychodzącej komunikacji HTTPS z serwerem Keycloak. Klient wysyłający żądania HTTPS potrzebuje sposobu na zweryfikowanie certyfikatu serwera, z którym rozmawia. Magazyn

	kluczy zawiera jeden lub więcej zaufanych certyfikatów hosta lub urzędów certyfikacji. Możesz utworzyć ten magazyn zaufanych certyfikatów, wyodrębniając publiczny certyfikat z magazynu kluczy SSL serwera Keycloak. WYMAGANY, chyba, że `ssl-required='none'` lub `disable-trust-manager='true'`.
truststore-password	Hasło do magazynu zaufania. WYMAGANY, jeśli magazyn zaufanych certyfikatów jest ustawiony i wymaga hasła.
client-keystore	Jest to ścieżka do pliku z magazynem kluczy. Ten magazyn kluczy zawiera certyfikat klienta do dwukierunkowego protokołu SSL, gdy adapter wysyła żądania HTTPS do serwera Keycloak. OPCJONALNY.
client-keystore-password	Hasło do magazynu kluczy klienta. WYMAGANY, jeśli ustawiony jest `client-keystore`.
client-key-password	Hasło do klucza klienta. WYMAGANY, jeśli ustawiony jest `client-keystore`.
always-refresh-token	Jeśli `true`, to adapter odświeży token przy każdym żądaniu. Ostrzeżenie: włączenie spowoduje wysłanie żądania do Keycloak po każdym żądaniu wysłanym do aplikacji.
register-node-at-startup	Jeśli `true`, to adapter wyśle żądanie rejestracji do Keycloak. Domyślna wartość to `false` i jest użyteczna tylko wtedy, gdy aplikacja jest klastrowana.
register-node-period	Okres na ponowną rejestrację adaptera do Keycloak. Przydatne, gdy aplikacja jest klastrowana. Aby uzyskać szczegółowe informacje, zobacz rozdziały dotyczące klastrowania.
token-store	Możliwe wartości to `'session'` i `'cookie'`. Domyślna wartość to `'session'`, która oznacza, że adapter przechowuje informacje o koncie w sesji HTTP. Alternatywna wartość `'cookie'` oznacza przechowywanie informacji w ciasteczku. Aby uzyskać szczegółowe informacje, zobacz rozdziały dotyczące klastrowania.
token-cookie-path	Gdy korzystasz z ciasteczek ta opcja określa ścieżkę do pliku cookie używanego do przechowywania informacji o koncie. Jeśli jest to ścieżka względna, zakłada się, że aplikacja działa w kontekstowym katalogu głównym i jest interpretowana względem tego kontekstowego katalogu głównego. Jeśli jest to ścieżka bezwzględna, wówczas do ustawienia ścieżki do pliku cookie używana jest ścieżka bezwzględna. Domyślnie używane są ścieżki względne do kontekstowego katalogu głównego.
principal-attribute	Atrybut z *OpenID Connect ID Token*, aby uzyskać nazwę UserPrincipal. Jeśli atrybut tokena ma wartość `null`, domyślnie przyjmuje się sub. Możliwe wartości to `sub`, `preferred_username`, `email`, `name`, `nickname`, `given_name`, `family_name`.
turn-off-change-session-id-on-login	Identyfikator sesji jest domyślnie zmieniany po udanym logowaniu (ze względu na bezpieczeństwo i potencjalne ataki). Zmień to na prawdę, jeśli chcesz to wyłączyć. OPCJONALNY. Wartość domyślna to `false`.
token-minimum-time-to-live	Czas w sekundach na zapobiegawcze odświeżenie aktywnego tokena dostępu na serwerze Keycloak przed jego wygaśnięciem. Jest to szczególnie przydatne, gdy token dostępu jest wysyłany do innego klienta REST, gdzie może wygasnąć przed oceną. Ta wartość nigdy nie powinna przekraczać żywotności tokena dostępu dla strefy (*ang. realm*). OPCJONALNY. Wartość domyślna to `0` sekund, więc adapter odświeży token dostępu, bezpośrednio po wygaśnięciu.
min-time-between-jwks-requests	Czas w sekundach, określający minimalny odstęp między dwoma żądaniami do Keycloak w celu pobrania nowych kluczy publicznych. Domyślnie jest to 10 sekund. Adapter zawsze będzie próbował pobrać nowy klucz publiczny, gdy

	rozpozna token z nieznanym `kid`. Jednak nie będzie próbować więcej niż raz na `10` sekund (domyślnie). Ma to na celu uniknięcie DoS, gdy atakujący wysyła wiele tokenów ze złym `kid` zmuszającym adapter do wysyłania wielu żądań do Keycloak.
public-key-cache-ttl	Czas w sekundach, określający maksymalny odstęp między dwoma żądaniami do Keycloak w celu pobrania nowych kluczy publicznych. Domyślnie jest to `86400` sekund (1 dzień). Adapter zawsze będzie próbował pobrać nowy klucz publiczny, gdy rozpozna token z nieznanym `kid`. Jeśli rozpozna token ze znanym `kid`, użyje klucza publicznego pobranego wcześniej. Jednak przynajmniej raz na ten skonfigurowany interwał (domyślnie 1 dzień) zawsze będzie pobierany nowy klucz publiczny, nawet jeśli `kid` tokena jest już znany.
ignore-oauth-query-parameter	Domyślnie `false`. Wartość `true`, wyłączy przetwarzanie parametru `access_token` do przetwarzania tokenu okaziciela. Użytkownicy nie będą mogli się uwierzytelnić, jeśli przekażą tylko token dostępu
redirect-rewrite-rules	W razie potrzeby określ regułę przepisywania URI przekierowania. Jest to notacja obiektowa, w której kluczem jest wyrażenie regularne, do którego ma zostać dopasowany identyfikator URI przekierowania, a wartością jest łańcuch zastępujący. Znak `$` może być użyty do odwołań wstecznych w łańcuchu zastępującym.
verify-token-audience	Jeśli ustawiono wartość `true`, to podczas uwierzytelniania za pomocą tokenu okaziciela adapter zweryfikuje, czy token zawiera tę nazwę klienta (zasób) jako odbiorców. Ta opcja jest szczególnie przydatna w przypadku usług, które przede wszystkim obsługują żądania uwierzytelnione przez token nośnika. Domyślnie jest ustawiona na `false`, jednak dla zwiększenia bezpieczeństwa zaleca się włączenie tej opcji.

2.1.2. Adapter WildFly (JBoss EAP)

Aby móc zabezpieczyć aplikacje WAR wdrożone w WildFly, JBoss EAP lub JBoss AS, musisz zainstalować i skonfigurować podsystem adaptera Keycloak. Dostępne są dwie opcje, aby zabezpieczyć swoje WARy.

Możesz podać plik konfiguracyjny adaptera w swoim WAR i zmienić metodę autoryzacji na KEYCLOAK w pliku `web.xml`.

Alternatywnie nie musisz wcale modyfikować pliku WAR i możesz ją zabezpieczyć za pomocą konfiguracji podsystemu adaptera Keycloak w pliku konfiguracyjnym, takim jak `standalone.xml`. Obie metody opisano w tej sekcji.

2.1.2.1. Instalowanie adaptera

Adaptery są dostępne jako osobne archiwum w zależności od używanej wersji serwera.

Adapter jest testowany i utrzymywany tylko z najnowszą wersją *WildFly*.

Przykładowa instalacja na *WildFly 18* lub nowszym:

```
$ cd $WILDFLY_HOME
$ unzip keycloak-wildfly-adapter-dist-10.0.2.zip
```

To archiwum ZIP zawiera moduły JBoss specyficzne dla adaptera Keycloak. Zawiera także skrypty CLI JBoss do konfigurowania podsystemu adaptera.

Aby skonfigurować podsystem adaptera, wykonaj na wyłączonym serwerze:

```
$ ./bin/jboss-cli.sh --file=bin/adapter-elytron-install-offline.cli
```

albo na włączonym:

```
$ ./bin/jboss-cli.sh -c --file=bin/adapter-elytron-install.cli
```

Alternatywnie można określić właściwość `server.config` podczas instalowania adapterów z wiersza polecenia, aby zainstalować adaptery przy użyciu innej konfiguracji, na przykład: `Dserver.config=standalone-ha.xml`.

WildFly ma wbudowaną obsługę pojedynczego logowania dla aplikacji internetowych wdrożonych w tej samej instancji WildFly. Nie należy tego włączać podczas korzystania z Keycloak.

Kontekst bezpieczeństwa jest automatycznie propagowany do warstwy EJB.

2.1.2.2. Konfigurowanie pakietów WAR

W tej sekcji opisano, jak zabezpieczyć WAR bezpośrednio, dodając pliki konfiguracyjne i edycyjne w pakiecie WAR. Pierwszą rzeczą, którą musisz zrobić, to utworzyć plik konfiguracyjny adaptera `WEB-INF/keycloak.json` w katalogu twojego pakietu WAR.

Format tego pliku konfiguracyjnego opisano w rozdziale Konfiguracja adaptera Java.

Następnie należy ustawić metodę autoryzacji na KEYCLOAK w pliku `web.xml`. Musisz także użyć standardowych zabezpieczeń serwletów, aby określić ograniczenia bazowe dla ról w swoich adresach URL.

Przykład:

```
   <web-app xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         version="3.0">

       <module-name>application</module-name>

       <security-constraint>
           <web-resource-collection>
               <web-resource-name>Admins</web-resource-name>
               <url-pattern>/admin/*</url-pattern>
           </web-resource-collection>
           <auth-constraint>
               <role-name>admin</role-name>
           </auth-constraint>
           <user-data-constraint>
               <transport-guarantee>CONFIDENTIAL</transport-guarantee>
           </user-data-constraint>
       </security-constraint>
       <security-constraint>
           <web-resource-collection>
               <web-resource-name>Customers</web-resource-name>
               <url-pattern>/customers/*</url-pattern>
           </web-resource-collection>
           <auth-constraint>
               <role-name>user</role-name>
           </auth-constraint>
           <user-data-constraint>
               <transport-guarantee>CONFIDENTIAL</transport-guarantee>
           </user-data-constraint>
       </security-constraint>

       <login-config>
           <auth-method>KEYCLOAK</auth-method>
           <realm-name>this is ignored currently</realm-name>
```

```
        </login-config>

        <security-role>
            <role-name>admin</role-name>
        </security-role>
        <security-role>
            <role-name>user</role-name>
        </security-role>
    </web-app>
```

2.1.2.3. Zabezpieczanie pakietów WAR za pomocą podsystemu adaptera

Nie musisz modyfikować pakietów WAR, aby zabezpieczyć ją za pomocą Keycloak. Zamiast tego możesz go zabezpieczyć zewnętrznie za pomocą podsystemu adaptera Keycloak. Chociaż nie musisz określać `KEYCLOAK` jako metody uwierzytelniania, nadal musisz zdefiniować ograniczenia bezpieczeństwa w pliku `web.xml`. Nie musisz jednak tworzyć pliku `WEB-INF/keycloak.json`. Te metadane są natomiast zdefiniowane w konfiguracji serwera (np. `standalone.xml`) w definicji podsystemu Keycloak.

```
    <extensions>
      <extension module="org.keycloak.keycloak-adapter-subsystem"/>
    </extensions>

    <profile>
      <subsystem xmlns="urn:jboss:domain:keycloak:1.1">
         <secure-deployment name="WAR MODULE NAME.war">
            <realm>demo</realm>
            <auth-server-url>http://localhost:8081/auth</auth-server-url>
            <ssl-required>external</ssl-required>
            <resource>customer-portal</resource>
            <credential name="secret">password</credential>
         </secure-deployment>
      </subsystem>
    </profile>
```

Atrybut `name` w `secure-deployment` określa WAR, który chcemy zabezpieczyć. Jego wartością jest nazwa modułu zdefiniowana w pliku `web.xml` z dołączonym tekstem `.war`. Reszta konfiguracji odpowiada praktycznie konfiguracji `keycloak.json`. Wyjątkiem jest element `credential`.

2.1.3. Spring Boot Adapter

Aby móc zabezpieczyć aplikacje *Spring Boot*, musisz dodać do swojej aplikacji plik JAR adaptera *Keycloak Spring Boot*. Następnie musisz podać dodatkową konfigurację za pomocą normalnej konfiguracji Spring Boot (`application.properties`).

2.1.3.1. Instalacja adaptera

Adapter *Keycloak Spring Boot* wykorzystuje autokonfigurację *Spring Boot*, więc wystarczy, że dodasz do swojego projektu odpowiedni starter Keycloak.

Aby dodać go za pomocą Maven, uzupełnij następujące elementy w zależnościach:

```
    <dependency>
        <groupId>org.keycloak</groupId>
        <artifactId>keycloak-spring-boot-starter</artifactId>
    </dependency>
```

Zależność adaptera BOM:

```
    <dependencyManagement>
```

```
    <dependencies>
      <dependency>
        <groupId>org.keycloak.bom</groupId>
        <artifactId>keycloak-adapter-bom</artifactId>
        <version>10.0.2</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
```

Obecnie obsługiwane są następujące serwery wbudowane i nie wymagają żadnych dodatkowych zależności, jeśli używasz Startera:

- Tomcat,
- Undertow,
- Jetty.

2.1.3.2. Wymagana konfiguracja adaptera Spring Boot

W tej sekcji opisano, jak skonfigurować aplikację Spring Boot do korzystania z Keycloak.

Zamiast pliku `keycloak.json` konfigurujesz dziedzinę adaptera Keycloak Spring Boot za pomocą normalnej konfiguracji Spring Boot. Na przykład:

```
keycloak.realm = demorealm
keycloak.auth-server-url = http://127.0.0.1:8080/auth
keycloak.ssl-required = external
keycloak.resource = demoapp
keycloak.credentials.secret = 11111111-1111-1111-1111-111111111111
keycloak.use-resource-role-mappings = true
```

Możesz wyłączyć Keycloak Spring Boot Adapter (na przykład w trakcie testów), ustawiając `keycloak.enabled = false`.

Aby skonfigurować *Policy Enforcer*, należy użyć `policy-enforcer-config` zamiast `policy-enforcer`.

Musisz także określić konfigurację zabezpieczeń *Java EE*, która normalnie byłaby dostępna w pliku `web.xml`. *Adapter Spring Boot* ustawi metodę logowania na KEYCLOAK i skonfiguruje ograniczenia bezpieczeństwa podczas uruchamiania. Oto przykładowa konfiguracja:

```
keycloak.securityConstraints[0].authRoles[0] = admin
keycloak.securityConstraints[0].authRoles[1] = user
keycloak.securityConstraints[0].securityCollections[0].name = insecure stuff
keycloak.securityConstraints[0].securityCollections[0].patterns[0] = /insecure

keycloak.securityConstraints[1].authRoles[0] = admin
keycloak.securityConstraints[1].securityCollections[0].name = admin stuff
keycloak.securityConstraints[1].securityCollections[0].patterns[0] = /admin
```

Jeśli planujesz wdrożyć aplikację Spring jako pakiet WAR, nie powinieneś używać *Spring Boot Adapter*. Twój *Spring Boot* powinien również zawierać plik `web.xml`.

2.1.4. Kontekst bezpieczeństwa

Interfejs KeycloakSecurityContext jest dostępny, jeśli potrzebujesz bezpośredniego dostępu do tokenów. Może to być przydatne, jeśli chcesz odzyskać dodatkowe dane z tokena (takie jak informacje o profilu użytkownika) lub chcesz wywołać usługę RESTful, która jest chroniona przez Keycloak.

W środowiskach serwletów jest dostępny w zabezpieczonych wywołaniach jako atrybut w HttpServletRequest:

```
httpServletRequest.getAttribute(KeycloakSecurityContext.class.getName());
```

Lub jest dostępny w niezabezpieczonych żądaniach w HttpSession:

```
httpServletRequest.getSession()
    .getAttribute(KeycloakSecurityContext.class.getName());
```

2.1.5. Obsługa błędów

Keycloak ma kilka funkcji obsługi błędów dla adapterów klienckich opartych na serwletach. Gdy wystąpi błąd podczas uwierzytelniania, Keycloak wywoła `HttpServletResponse.sendError()`. Możesz ustawić stronę błędu w pliku `web.xml`, aby obsłużyć błąd w dowolny sposób. Keycloak może zgłaszać błędy 400, 401, 403 i 500.

```
<error-page>
    <error-code>403</error-code>
    <location>/ErrorHandler</location>
</error-page>
```

Keycloak ustawia również atrybut `org.keycloak.adapters.spi.AuthenticationError`, który można pobrać z `HttpServletRequest`. Wartość należy zrzutować jako `OIDCAuthenticationError`.

Przykład:

```
import org.keycloak.adapters.OIDCAuthenticationError;
import org.keycloak.adapters.OIDCAuthenticationError.Reason;
...
OIDCAuthenticationError error = (OIDCAuthenticationError) httpServletRequest
    .getAttribute('org.keycloak.adapters.spi.AuthenticationError');
Reason reason = error.getReason();
System.out.println(reason.name());
```

2.1.6. Wylogowanie

Można wylogować się z aplikacji internetowej na wiele sposobów. W przypadku serwletów Java EE można wywołać `HttpServletRequest.logout()`. W przypadku innych aplikacji możesz przekierować przeglądarkę na `http://auth.domena.pl/auth/realms/{strefy}/protocol/openid-connect/logout?redirect_uri=encodedRedirectUri`, który wyloguje Cię, jeśli masz sesje SSO w przeglądarce.

Podczas korzystania z opcji `HttpServletRequest.logout()` adapter wykonuje wywołanie POST kanału zwrotnego do serwera Keycloak przekazując token odświeżania. Jeśli metoda jest wykonywana z niezabezpieczonej strony (strony, która nie sprawdza poprawnego tokena), token odświeżania może być niedostępny i w takim przypadku adapter pomija wywołanie. Z tego powodu zaleca się użycie chronionej strony do wykonania `HttpServletRequest.logout()`, aby zawsze brać pod uwagę bieżące tokeny i w razie potrzeby przeprowadzić interakcję z serwerem Keycloak.

Jeśli chcesz uniknąć wylogowywania się z zewnętrznego dostawcy tożsamości w ramach procesu wylogowywania, możesz podać parametr `initiating_idp`, którego wartością jest tożsamość (alias) danego dostawcy tożsamości. Jest to przydatne, gdy punkt końcowy wylogowania jest wywoływany jako część pojedynczego wylogowania zainicjowanego przez zewnętrznego dostawcę tożsamości.

2.1.7. Przekazywanie parametrów

Inicjujące żądanie do punktu końcowego autoryzacji Keycloak obsługuje różne parametry. Większość parametrów opisano w specyfikacji OIDC. Niektóre parametry są dodawane automatycznie przez adapter na podstawie konfiguracji adaptera. Istnieje jednak kilka parametrów, które można dodać na podstawie wywołania. Po otwarciu bezpiecznego identyfikatora URI aplikacji określony parametr zostanie przesłany do punktu końcowego autoryzacji Keycloak.

Na przykład jeśli poprosisz o token offline, możesz otworzyć zabezpieczony identyfikator URI aplikacji za pomocą parametru scope, takiego jak: `http://myappserver/mysecuredapp?scope=offline_access`,

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

a parametr `scope=offline_access` zostanie automatycznie przesłany do punktu końcowego autoryzacji Keycloak.

Obsługiwane parametry to:

1. `scope` – użyj rozdzielonej spacjami listy zakresów. Lista rozdzielana spacjami zwykle odwołuje się do zakresów klienta zdefiniowanych na konkretnym kliencie. Należy zauważyć, że adapter `openid` zawsze będzie dodawany do listy zakresów przez adapter. Na przykład, jeśli wpiszesz `scope=address phone`, wówczas żądanie do Keycloak będzie zawierało parametr zakresu `scope=openid address phone`.
2. `prompt` – Keycloak obsługuje następujące ustawienia:
    a. `login` – logowanie jednokrotne zostanie zignorowane, a strona logowania Keycloak będzie zawsze wyświetlana, nawet jeśli użytkownik jest już uwierzytelniony;
    b. `consent` – dotyczy tylko klientów wymagających zgody. Jeśli jest używana, strona zgody będzie zawsze wyświetlana, nawet jeśli użytkownik wcześniej udzielił zgody temu klientowi;
    c. none – strona logowania nigdy nie będzie wyświetlana; zamiast tego użytkownik zostanie przekierowany do aplikacji, z błędem, jeśli użytkownik nie jest jeszcze uwierzytelniony; to ustawienie pozwala utworzyć filtr/przechwytywacz po stronie aplikacji i wyświetlić użytkownikowi niestandardową stronę błędu. Zobacz więcej szczegółów w specyfikacji.
3. `max_age` – Używany tylko wtedy, gdy użytkownik jest już uwierzytelniony. Określa maksymalny dozwolony czas trwania uwierzytelnienia, mierzony od momentu uwierzytelnienia użytkownika. Jeśli użytkownik jest uwierzytelniany dłużej niż `max_age`, logowanie jednokrotne jest ignorowane i musi zostać ponownie uwierzytelniony.
4. `login_hint` – Służy do wstępnego wypełnienia pola nazwy użytkownika / adresu e-mail w formularzu logowania.
5. `kc_idp_hint` – Służy do informowania Keycloak o pominięciu strony logowania i automatycznym przekierowaniu do określonego dostawcy tożsamości. Więcej informacji w dokumentacji dostawcy tożsamości.

Większość parametrów jest opisanych w specyfikacji *OIDC*. Jedynym wyjątkiem jest parametr `kc_idp_hint`, który jest specyficzny dla Keycloak i zawiera nazwę dostawcy tożsamości do automatycznego użycia. Aby uzyskać więcej informacji, zobacz rozdziały Identity Brokering w **Podręczniku administracji serwerem**.

Jeśli otworzysz adres URL przy użyciu dołączonych parametrów, adapter nie przekieruje Cię do Keycloak, jeśli jesteś już uwierzytelniony w aplikacji. Np. otwarcie `http://myappserver/mysecuredapp?prompt=login` nie spowoduje automatycznego przekierowania do strony logowania Keycloak, jeśli jesteś już uwierzytelniony w aplikacji `mysecredapp`. To zachowanie może ulec zmianie w przyszłości.

2.1.8. Uwierzytelnianie klienta

Gdy poufny klient OIDC musi wysłać żądanie kanału zwrotnego (na przykład, aby wymienić kod na token lub odświeżyć token), musi się uwierzytelnić na serwerze Keycloak. Domyślnie istnieją trzy sposoby uwierzytelnienia klienta: identyfikator klienta i klucz tajny klienta, uwierzytelnienie klienta za pomocą podpisanego JWT lub uwierzytelnienie klienta za pomocą podpisanego JWT za pomocą klucza tajnego klienta.

2.1.8.1. Identyfikator klienta i klucz tajny klienta

Jest to tradycyjna metoda opisana w specyfikacji *OAuth2*. Klient ma sekret, który musi być znany zarówno adapterowi (aplikacji), jak i serwerowi Keycloak. Możesz wygenerować klucz tajny dla konkretnego klienta w konsoli administracyjnej Keycloak, a następnie wkleić ten klucz tajny do pliku `keycloak.json` po stronie aplikacji:

```
"credentials": {
    "secret": "19666a4f-32dd-4049-b082-684c74115f28"
}
```

2.1.8.2. Uwierzytelnianie klienta za pomocą podpisanego JWT

Jest to oparte na specyfikacji *RFC7523* i działa w następujący sposób:

1. Klient musi mieć klucz prywatny i certyfikat. W przypadku Keycloak jest to dostępne za pośrednictwem tradycyjnego pliku magazynu kluczy, który jest dostępny na ścieżce *classpath* aplikacji klienta lub w systemie plików.
2. Po uruchomieniu aplikacja kliencka umożliwia pobranie klucza publicznego w formacie JWKS przy użyciu adresu URL takiego jak `http://myhost.com/myapp/k_jwks`, przy założeniu, że `http://myhost.com/myapp` jest podstawowym adresem URL twojej aplikacji klienckiej. Ten adres URL może być używany przez Keycloak (patrz poniżej).
3. Podczas uwierzytelniania klient generuje token JWT i podpisuje go swoim kluczem prywatnym i wysyła go do Keycloak w określonym żądaniu kanału zwrotnego (na przykład żądanie `code-to-token`) w parametrze `client_assertion`.
4. Keycloak musi mieć klucz publiczny lub certyfikat klienta, aby mógł zweryfikować podpis na JWT. W Keycloak musisz skonfigurować poświadczenia klienta dla swojego klienta. Najpierw musisz wybrać opcję `Podpisany JWT` jako metodę uwierzytelniania klienta na karcie `Credentials` w konsoli administracyjnej. Następnie możesz wybrać:
    a. Skonfiguruj adres JWKS URL, z którego Keycloak może pobierać klucze publiczne klienta. Może to być adres URL taki jak http://myhost.com/myapp/k_jwks (patrz szczegóły powyżej). Ta opcja jest najbardziej elastyczna, ponieważ klient może w dowolnym momencie odnowić klucze, a Keycloak zawsze pobiera nowe klucze w razie potrzeby, bez konieczności zmiany konfiguracji. Dokładniej, Keycloak pobiera nowe klucze, gdy widzi token podpisany przez nieznany `kid` (identyfikator klucza).
    b. Prześlij klucz publiczny lub certyfikat klienta w formacie PEM, w formacie JWK lub magazynu kluczy. Dzięki tej opcji klucz publiczny jest zakodowany na stałe i musi zostać zmieniony, gdy klient wygeneruje nową parę kluczy. Możesz nawet wygenerować własny magazyn kluczy z konsoli administracyjnej Keycloak, jeśli nie masz własnego. Aby uzyskać więcej informacji na temat konfigurowania konsoli administracyjnej Keycloak, patrz Podręcznik administracji serwera.

Aby skonfigurować po stronie adaptera, musisz mieć coś takiego w pliku `keycloak.json`:

```
"credentials": {
  "jwt": {
    "client-keystore-file": "classpath:keystore-client.jks",
    "client-keystore-type": "JKS",
    "client-keystore-password": "storepass",
    "client-key-password": "keypass",
    "client-key-alias": "clientkey",
    "token-expiration": 10
  }
}
```

W tej konfiguracji plik kluczy `keystore-client.jks` musi być dostępny w classpath pliku WAR. Można też wskazać dowolny plik w systemie plików, do którego ma dostęp aplikacja kliencka.

2.1.8.3. Uwierzytelnianie klienta za pomocą JWT podpisanego kluczem tajnym klienta

Metoda analogiczna jak opisana wcześniej, z wyjątkiem tego, że używa klucza tajnego klienta zamiast klucza prywatnego i certyfikatu.

Klient ma sekret, który musi być znany zarówno adapterowi (aplikacji), jak i serwerowi Keycloak. Musisz wybrać opcję `Signed JWT with Client Secret` jako metodę uwierzytelniania klienta na karcie `Credentials` w konsoli administracyjnej, a następnie wkleić ten klucz tajny do pliku `keycloak.json` po stronie aplikacji:

```
"credentials": {
  "secret-jwt": {
    "secret": "19666a4f-32dd-4049-b082-684c74115f28",
    "algorithm": "HS512"
  }
```

```
    }
```

Parametr `"algorithm"` określa algorytm podpisywania JWT. Musi to być jedna z następujących wartości: HS256, HS384 lub HS512. Aby uzyskać szczegółowe informacje, zapoznaj się z *JSON Web Algorithms (JWA)*.

Parametr `"algorithm"` jest opcjonalny, a wartość domyślna to HS256.

2.1.8.4. Dodaj własną metodę uwierzytelniania klienta

Możesz także dodać własną metodę uwierzytelniania klienta. Konieczne będzie wdrożenie dostawców zarówno po stronie klienta, jak i serwera. Aby uzyskać więcej informacji, zobacz rozdział Interfejs SPI uwierzytelniania w Podręczniku dewelopera serwera.

2.1.9. Wielopodmiotowość (ang. Multi Tenancy)

W tym kontekście Multi Tenancy oznacza, że jedną aplikację docelową (WAR) można zabezpieczyć za pomocą wielu stref (ang. realms) Keycloak. Strefy mogą znajdować się w tej samej instancji Keycloak lub w różnych instancjach.

W praktyce oznacza to, że aplikacja musi mieć wiele plików konfiguracyjnych adaptera `keycloak.json`.

Możesz mieć wiele wystąpień WAR z różnymi plikami konfiguracyjnymi adaptera wdrożonymi w różnych ścieżkach kontekstu. Może to być jednak niewygodne i możesz również wybrać strefę na podstawie czegoś innego niż ścieżka kontekstu.

Keycloak pozwala mieć niestandardowy program rozpoznawania konfiguracji, dzięki czemu możesz wybrać konfigurację adaptera dla każdego żądania.

Najpierw należy utworzyć implementację `org.keycloak.adapters.KeycloakConfigResolver`.

Na przykład:

```
   package example;
   import org.keycloak.adapters.KeycloakConfigResolver;
   import org.keycloak.adapters.KeycloakDeployment;
   import org.keycloak.adapters.KeycloakDeploymentBuilder;

   public class PathBasedKeycloakConfigResolver implements KeycloakConfigResolver
{

       @Override
       public KeycloakDeployment resolve(OIDCHttpFacade.Request request) {
           if (path.startsWith("alternative")) {
               KeycloakDeployment deployment = cache.get(realm);
               if (null == deployment) {
                   InputStream is = getClass().getResourceAsStream("/tenant1-
keycloak.json");
                   return KeycloakDeploymentBuilder.build(is);
               }
           } else {
               InputStream is = getClass().getResourceAsStream("/default-
keycloak.json");
               return KeycloakDeploymentBuilder.build(is);
           }
       }
   }
```

Musisz także skonfigurować, której implementacji `KeycloakConfigResolver` używać, przy pomocy parametru `keycloak.config.resolver` w pliku `web.xml`:

```
   <web-app>
       ...
```

```
        <context-param>
            <param-name>keycloak.config.resolver</param-name>
            <param-value>example.PathBasedKeycloakConfigResolver</param-value>
        </context-param>
    </web-app>
```

2.1.10. Klastrowanie aplikacji

Ten rozdział dotyczy obsługi aplikacji klastrowych wdrożonych w *WildFly*, *JBoss EAP* i *JBoss AS*.

Dostępnych jest kilka opcji w zależności od tego, czy twoja aplikacja jest:

1. stanowa lub bezstanowa (*ang. stateful or stateless*),
2. dystrybuowana (replikowana sesja HTTP),
3. wykorzystująca lepkie sesje zapewniane przez moduł równoważenia obciążenia,
4. hostowana w tej samej domenie co Keycloak.

Radzenie sobie z klastrowaniem nie jest tak proste, jak w przypadku zwykłych instalacji. Głównie dlatego, że zarówno przeglądarka, jak i aplikacja po stronie serwera wysyła żądania do Keycloak, więc nie jest to tak proste, jak włączenie sesji trwałych w module równoważenia obciążenia.

2.1.10.1. Bezstanowy magazyn z tokenami

Domyślnie aplikacja internetowa zabezpieczona przez Keycloak używa sesji HTTP do przechowywania kontekstu bezpieczeństwa. Oznacza to, że musisz włączyć sesje trwałe (*ang. sticky sessions*) lub zreplikować sesję HTTP.

Alternatywnie do przechowywania kontekstu bezpieczeństwa w sesji HTTP, adapter można skonfigurować tak, aby przechowywał go w plikach cookie. Jest to przydatne, jeśli chcemy uczynić swoją aplikację bezstanową lub jeśli nie chcesz przechowywać kontekstu bezpieczeństwa w sesji HTTP.

Aby użyć sklepu z plikami cookie do zapisania kontekstu bezpieczeństwa do `WEB-INF/keycloak.json` dodaj:

```
    "token-store": "cookie"
```

Domyślną wartością dla `token-store` jest `session`, dla której kontekst bezpieczeństwa przechowywany jest w sesji HTTP.

Jednym z ograniczeń korzystania z magazynu z ciasteczkami jest to, że cały kontekst bezpieczeństwa jest przekazywany do pliku cookie dla każdego żądania HTTP. Może to wpłynąć na wydajność.

Kolejnym niewielkim ograniczeniem jest ograniczona obsługa jednokrotnego wylogowania. Działa bez problemów, jeśli zainicjujesz wylogowanie serwletu (`HttpServletRequest.logout`) z samej aplikacji, ponieważ adapter usunie cookie KEYCLOAK_ADAPTER_STATE. Jednak wylogowanie kanału zwrotnego zainicjowane z innej aplikacji nie jest propagowane przez Keycloak do aplikacji korzystających z cookie. Dlatego zaleca się użycie krótkiej wartości dla limitu czasu tokena dostępu (na przykład 1 minuta).

Niektóre moduły równoważenia obciążenia nie zezwalają na żadną konfigurację nazwy lub zawartości cookie sesji trwałej (na przykład Amazon ALB). W takim przypadku zaleca się ustawienie opcji `shouldAttachRoute` na wartość `false`.

2.1.10.2. Optymalizacja względnych URI

W scenariuszach wdrażania, w których Keycloak i aplikacja są hostowane w tej samej domenie (przez odwrotne proxy lub moduł równoważenia obciążenia), wygodne może być użycie względnych opcji URI w konfiguracji klienta.

W przypadku względnych identyfikatorów URI są one rozpoznawane jako względne do adresu URL używanego do uzyskania dostępu do Keycloak.

Na przykład jeśli adres URL Twojej aplikacji to `https://acme.org/myapp`, a URL do Keycloak to `https://acme.org/auth`, możesz użyć redirect-uri `/myapp` zamiast `https://acme.org/myapp`.

2.1.10.3. Konfiguracja adresu URL administratora

Adres URL administratora dla konkretnego klienta można skonfigurować w konsoli administracyjnej Keycloak. Jest używany przez serwer Keycloak do wysyłania żądań dla różnych zadań, np. takich jak wylogowanie użytkowników. Mechanizm wylogowania z kanału wstecznego może być następujący:

1. Użytkownik wysyła żądanie wylogowania z jednej aplikacji.
2. Aplikacja wysyła żądanie wylogowania do Keycloak.
3. Serwer Keycloak unieważnia sesję użytkownika.
4. Serwer Keycloak wysyła następnie żądanie kanału zwrotnego do aplikacji z adresem URL administratora powiązanym z sesją.
5. Gdy aplikacja otrzyma żądanie wylogowania, unieważnia odpowiednią sesję HTTP.

Jeśli adres URL administratora zawiera `${application.session.host}`, zostanie zastąpiony adresem URL węzła powiązanego z sesją HTTP.

2.1.10.4. Rejestracja węzłów aplikacji

W poprzedniej sekcji opisano, w jaki sposób Keycloak może wysyłać żądanie wylogowania do węzła powiązanego z określoną sesją HTTP. Jednak w niektórych przypadkach administrator może chcieć propagować zadania administracyjne do wszystkich zarejestrowanych węzłów klastra, a nie tylko do jednego z nich. Na przykład, aby przekazać nową politykę *not before* lub wylogować wszystkich użytkowników z aplikacji.

W takim przypadku Keycloak musi znać wszystkie węzły klastra aplikacji, aby mógł wysłać zdarzenie do wszystkich z nich. Aby to osiągnąć, wspieramy mechanizm automatycznego wykrywania:

1. Gdy nowy węzeł aplikacji dołącza do klastra, wysyła żądanie rejestracji do serwera Keycloak.
2. Żądanie może być ponownie wysłane okresowo do Keycloak w skonfigurowanych odstępach czasu.
3. Jeśli serwer Keycloak nie otrzyma żądania ponownej rejestracji w określonym czasie, automatycznie wyrejestruje określony węzeł.
4. Węzeł jest również wyrejestrowany z Keycloak, gdy wysyła żądanie wyrejestrowania, co zwykle ma miejsce podczas zamykania węzła lub wyrejestrowywania *(ang. undeployment)* aplikacji. Może to nie działać poprawnie w przypadku wymuszonego zamknięcia, gdy detektory wyrejestrowywania nie są wywoływane, co powoduje konieczność automatycznego wyrejestrowania.

Wysyłanie rejestracji startowej i okresowych jest domyślnie wyłączone, ponieważ jest to wymagane tylko przez niektóre aplikacje klastrowe.

Aby włączyć tą funkcję, edytuj plik `WEB-INF/keycloak.json` dla swojej aplikacji i dodaj:

```
"register-node-at-startup": true,
"register-node-period": 600,
```

Oznaczają one, że adapter wyśle żądanie rejestracji podczas uruchamiania i będzie rejestrować się ponownie co 10 minut.

W konsoli administracyjnej Keycloak można określić maksymalny limit czasu ponownej rejestracji węzła (powinien być większy niż okres rejestracji węzła w konfiguracji adaptera). Możesz także ręcznie dodawać i usuwać węzły klastra za pomocą Konsoli administracyjnej, co jest przydatne, jeśli nie chcesz polegać na funkcji automatycznej rejestracji lub jeśli chcesz usunąć przestarzałe węzły aplikacji w przypadku, gdy nie korzystasz z funkcji automatycznego wyrejestrowywania.

2.1.10.5. Odśwież token w każdym żądaniu

Domyślnie adapter aplikacji odświeża token dostępu dopiero po jego wygaśnięciu. Można jednak również skonfigurować adapter, aby odświeżał token co każde żądanie. Może to mieć wpływ na wydajność, ponieważ aplikacja wyśle więcej żądań do serwera Keycloak.

Aby włączyć tę funkcję, edytuj plik `WEB-INF/keycloak.json` dla swojej aplikacji i dodaj:

```
"always-refresh-token": true
```

Włącz tę funkcję tylko wtedy, gdy nie możesz polegać na komunikatach w kanale zwrotnym w celu propagowania wylogowania i polityce *not before*. Inną rzeczą do rozważenia jest, aby tokeny dostępu miały domyślnie krótki termin ważności – nawet jeśli wylogowanie nie zostanie ropropagowane, to token wygaśnie w ciągu kilku minut od wylogowania.

2.2. Adaptery JavaScript

Keycloak jest wyposażony w bibliotekę JavaScript po stronie klienta, która może być używana do zabezpieczania aplikacji *HTML5/JavaScript*. Adapter JavaScript ma wbudowaną obsługę aplikacji *Cordova*.

Bibliotekę można pobrać bezpośrednio z serwera Keycloak pod adresem /auth/js/keycloak.js i jest ona również dystrybuowana jako archiwum ZIP.

Najlepszą praktyką jest ładowanie adaptera JavaScript bezpośrednio z serwera Keycloak, ponieważ zostanie on automatycznie zaktualizowany po uaktualnieniu serwera. Jeśli zamiast tego skopiujesz adapter do aplikacji internetowej, upewnij się, że uaktualniłeś adapter dopiero po uaktualnieniu serwera.

Jedną ważną rzeczą, o której należy pamiętać przy korzystaniu z aplikacji po stronie klienta, jest to, że klient musi być klientem publicznym, ponieważ nie ma bezpiecznego sposobu przechowywania poświadczeń klienta w aplikacji po stronie klienta. Dlatego bardzo ważne jest, aby upewnić się, że identyfikatory URI przekierowania skonfigurowane dla klienta są poprawne i jak najbardziej szczegółowe.

Aby użyć adaptera JavaScript, musisz najpierw utworzyć klienta dla swojej aplikacji w konsoli administracyjnej Keycloak. Upewnij się, że jako `Access Type` wybrano `public`.

Musisz także skonfigurować `Valid Redirect URIs` i `Web Origins`. Podajemy je jak najdokładniej, ponieważ nieprzestrzeganie tego może spowodować lukę w zabezpieczeniach.

Po utworzeniu klienta klikamy na kartę `Installation` i dla `Format Option` wybieramy `Keycloak OIDC JSON`, a następnie klikamy `Download`. Pobrany plik `keycloak.json` powinien znajdować się na serwerze Web w tej samej lokalizacji, co strony HTML.

Alternatywnie możesz pominąć plik konfiguracyjny i ręcznie skonfigurować adapter.

Poniższy przykład pokazuje, jak zainicjować adapter JavaScript:

```
<head>
    <script src="keycloak.js"></script>
    <script>
        var keycloak = new Keycloak();
        keycloak.init().then(function(authenticated) {
            alert(authenticated ? 'authenticated' : 'not authenticated');
        }).catch(function() {
            alert('failed to initialize');
        });
    </script>
</head>
```

Jeśli plik `keycloak.json` znajduje się w innej lokalizacji, możesz go określić:

```
var keycloak = new Keycloak('http://localhost:8080/myapp/keycloak.json');
```

Można też przekazać obiekt JavaScript z wymaganą konfiguracją:

```
var keycloak = new Keycloak({
    url: 'http://keycloak-server/auth',
    realm: 'myrealm',
    clientId: 'myapp'
});
```

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

Domyślnie w celu uwierzytelnienia musisz wywołać funkcję logowania. Dostępne są jednak dwie opcje automatycznego uwierzytelniania adaptera. Możesz do funkcji `init` przekazać `login-required`, które uwierzytelni klienta, jeśli użytkownik jest zalogowany do Keycloak lub wyświetli stronę logowania, jeśli nie. Alternatywnie możesz przekazać `check-sso`, które uwierzytelni klienta tylko wtedy, gdy użytkownik jest już zalogowany, jeśli użytkownik nie jest zalogowany, przeglądarka zostanie przekierowana z powrotem do aplikacji i pozostanie nieuwierzytelniona.

Możesz skonfigurować opcję cichej `check-sso`. Po włączeniu tej funkcji przeglądarka nie dokonuje pełnego przekierowania do serwera Keycloak i z powrotem do aplikacji, ale ta czynność zostanie wykonana w ukrytym elemencie iframe. Zasoby aplikacji muszą wtedy zostać załadowane i przeanalizowane tylko raz przez przeglądarkę po zainicjowaniu aplikacji, a nie ponownie po przekierowaniu z powrotem z Keycloak do aplikacji. Jest to szczególnie przydatne w przypadku aplikacje pojedynczej strony (ang. Single Page Application, SPA).

Aby włączyć cichą `check-sso`, musisz podać atrybut `silentCheckSsoRedirectUri` w metodzie `init`. Ten identyfikator URI musi być prawidłowym punktem końcowym w aplikacji (i oczywiście musi być skonfigurowany jako prawidłowe przekierowanie dla klienta w konsoli administracyjnej Keycloak):

```
   keycloak.init({
       onLoad: 'check-sso',
       silentCheckSsoRedirectUri: window.location.origin + '/silent-check-
sso.html'
   })
```

Strona przy cichym `check-sso`, URI przekierowania ładuje do ramki iframe po pomyślnym sprawdzeniu stanu uwierzytelnienia i pobraniu tokenów z serwera Keycloak. Nie ma innego zadania niż wysłanie otrzymanych tokenów do głównej aplikacji i powinna wyglądać tylko tak:

```
   <html>
   <body>
       <script>
           parent.postMessage(location.href, location.origin)
       </script>
   </body>
   </html>
```

Należy pamiętać, że powyższa strona w określonej lokalizacji musi być dostarczona przez samą aplikację i nie jest częścią adaptera JavaScript!

Począwszy od wersji Chrome 80 (wydanej w lutym 2020 r.), funkcja cichego `check-sso` będzie działać tylko wtedy, gdy połączenie po stronie Keycloak to SSL/TLS.

Aby włączyć wymagane logowanie, ustaw `onLoad` na wymagane logowanie i przejdź do metody `init`:

```
   keycloak.init({
       onLoad: 'login-required'
   })
```

Po uwierzytelnieniu użytkownika aplikacja może wysyłać żądania do usług RESTful zabezpieczonych przez Keycloak, umieszczając token na okaziciela w nagłówku `Authorization`. Na przykład:

```
   var loadData = function () {
       document.getElementById('username').innerText = keycloak.subject;
       var url = 'http://localhost:8080/restful-service';
       var req = new XMLHttpRequest();
       req.open('GET', url, true);
       req.setRequestHeader('Accept', 'application/json');
       req.setRequestHeader('Authorization', 'Bearer ' + keycloak.token);
       req.onreadystatechange = function () {
           if (req.readyState == 4) {
```

```
            if (req.status == 200) {
                alert('Success');
            } else if (req.status == 403) {
                alert('Forbidden');
            }
        }
    }
    req.send();
};
```

Należy pamiętać, że token dostępu domyślnie ma krótki okres ważności, więc może być konieczne odświeżenie tokena dostępu przed wysłaniem żądania. Możesz to zrobić za pomocą metody `updateToken`. Metoda `updateToken` zwraca obietnicę, która ułatwia wywołanie usługi tylko wtedy, gdy token został pomyślnie odświeżony i wyświetla komunikat o błędzie użytkownikowi, jeśli tak nie było. Na przykład:

```
keycloak.updateToken(30).then(function() {
    loadData();
}).catch(function() {
    alert('Failed to refresh token');
});
```

2.2.1. Status sesji iframe

Domyślnie adapter JavaScript tworzy ukryty element iframe, który służy do wykrywania, czy nastąpiło wylogowanie. Nie wymaga to żadnego ruchu sieciowego, zamiast tego status jest pobierany poprzez sprawdzenie specjalnego pliku cookie dla statusu. Tę funkcję można wyłączyć, ustawiając `checkLoginIframe: false` w opcjach przekazanych do metody `init`.

Nie powinieneś polegać na bezpośrednim sprawdzaniu tego pliku cookie. Jego format może się zmienić i jest również powiązany z adresem URL serwera Keycloak, a nie Twojej aplikacji.

W najnowszych przeglądarkach status przez iframe jest dostępny tylko przez skonfigurowane połączenia SSL/TLS po stronie Keycloak. Korzystanie z niezabezpieczonego połączenia może prowadzić do przekierowania do Keycloak za każdym razem, gdy iframe sprawdza status. Można tego uniknąć, wyłączając element iframe lub konfigurując protokół SSL/TLS po stronie Keycloak.

2.2.2. Przepływ implicit i hybrydowy

Domyślnie adapter *JavaScript* używa przepływu kodu autoryzacji (ang. Authorization Code Flow).

Dzięki temu przepływowi serwer Keycloak zwraca do aplikacji kod autoryzacji, a nie token uwierzytelnienia. Adapter JavaScript wymienia kod na token dostępu i token odświeżania (po przekierowaniu przeglądarki z powrotem do aplikacji).

Keycloak obsługuje również przepływ implicit, w którym token dostępu jest wysyłany natychmiast po udanym uwierzytelnieniu za pomocą Keycloak. Może to mieć lepszą wydajność niż przepływ standardowy, ponieważ nie ma dodatkowego żądania wymiany kodu na tokeny, ale ma to wpływ na wygaśnięcie tokena dostępu.

Jednak wysłanie tokena dostępu we fragmencie adresu URL może stanowić lukę w zabezpieczeniach. Na przykład token może wyciec z dzienników serwera WWW lub historii przeglądarki.

Aby włączyć przepływ implicit, należy włączyć flagę `Implicit Flow Enabled` dla klienta w konsoli administracyjnej Keycloak. Musisz także przekazać parametr `flow` z wartością `implicit` do metody `init`:

```
keycloak.init({
    flow: 'implicit'
})
```

Należy zauważyć, że dostarczany jest tylko token dostępu i nie ma tokena odświeżania. Oznacza to, że po wygaśnięciu tokena dostępu aplikacja musi ponownie wykonać przekierowanie do Keycloak, aby uzyskać nowy token dostępu.

Keycloak obsługuje również przepływ hybrydowy.

Wymaga to aby włączyć dla klienta w konsoli administracyjnej obie flagi: `Standard Flow Enabled` oraz `Implicit Flow Enabled`. Serwer Keycloak wyśle wtedy zarówno kod, jak i tokeny do Twojej aplikacji. Z tokena dostępu można natychmiast skorzystać, a kod można wymienić na tokeny dostępu i odświeżania. Podobnie jak przepływ implicit, przepływ hybrydowy poprawia wydajność, ponieważ token dostępu jest dostępny natychmiast. Jednak token jest nadal wysyłany w adresie URL, a wspomniana wcześniej luka w zabezpieczeniach może nadal obowiązywać.

Jedną z przewag przepływu hybrydowego nad implicit jest to, że aplikacji jest udostępniany token odświeżania.

W przypadku przepływu hybrydowego należy przekazać parametr `flow` z wartością `hybrid` do metody `init`:

```
keycloak.init({
    flow: 'hybrid'
})
```

2.2.3. Aplikacje hybrydowe z Cordova

Keycloak obsługuje hybrydowe aplikacje mobilne opracowane przy użyciu Apache Cordova. Adapter JavaScript ma do tego dwa tryby: `cordova` i `cordova-native`.

Domyślny tryb to `cordova`, którą adapter wybierze automatycznie, jeśli nie skonfigurowano żadnego typu adaptera i jest dostęp do `window.cordova`. Podczas logowania otworzy się przeglądarka InApp, która pozwala użytkownikowi na interakcję z Keycloak, a następnie powraca do aplikacji poprzez przekierowanie na `http://localhost`. Z tego powodu musisz dodać ten adres URL do białej listy jako prawidłowy identyfikator przekierowania w sekcji konfiguracji klienta w konsoli administracyjnej.

Chociaż ten tryb jest łatwy w konfiguracji, ma również pewne wady:

1. InApp-Browser to przeglądarka osadzona w aplikacji i nie jest domyślną przeglądarką telefonu. Dlatego będzie miała inne ustawienia, a przechowywane poświadczenia nie będą dostępne.
2. Przeglądarka InApp może również działać wolniej, szczególnie przy renderowaniu bardziej złożonych motywów.
3. Przed użyciem tego trybu należy rozważyć obawy związane z bezpieczeństwem, takie jak możliwość uzyskania przez aplikację dostępu do poświadczeń użytkownika, ponieważ ma ona pełną kontrolę nad przeglądarką wyświetlającą stronę logowania, więc nie zezwalaj na jego użycie w aplikacjach, którym nie ufasz.

Tryb alternatywny `cordova-native` ma inne podejście. Otwiera stronę logowania za pomocą przeglądarki systemu. Po uwierzytelnieniu użytkownika przeglądarka przekierowuje z powrotem do aplikacji za pomocą specjalnego adresu URL. Stamtąd adapter Keycloak może zakończyć logowanie, czytając kod lub token z adresu URL.

Możesz aktywować tryb natywny, przekazując odpowiednie parametry do metody `init`:

```
keycloak.init({
    adapter: 'cordova-native'
})
```

Ten adapter wymaga dwóch dodatkowych wtyczek:

- `cordova-plugin-browsertab`: pozwala aplikacji na otwieranie stron internetowych w przeglądarce systemu,
- `cordova-plugin-deeplinks`: zezwól przeglądarce na przekierowanie z powrotem do Twojej aplikacji za pomocą specjalnych adresów URL.

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

Szczegóły techniczne łączenia z aplikacją różnią się na poszczególnych platformach i wymagana jest specjalna konfiguracja. Dalsze instrukcje znajdują się w sekcjach Androida i iOS w dokumentacji wtyczki deeplinks.

Istnieją różne rodzaje linków do otwierania aplikacji: schematy niestandardowe (tj. `myapp://login` lub `android-app://com.example.myapp/https/example.com/login`) i *Universal Links (iOS)* / *Deep Links (Android)*. Podczas gdy te pierwsze są łatwiejsze do skonfigurowania i działają bardziej niezawodnie, później oferują dodatkowe bezpieczeństwo, ponieważ są unikalne i tylko właściciel domeny może je zarejestrować. Niestandardowe adresy URL są przestarzałe w systemie iOS. Zalecamy korzystanie z uniwersalnych linków w połączeniu z rezerwową witryną z niestandardowym linkiem do niej, aby zapewnić najwyższą niezawodność.

Ponadto zalecamy następujące kroki w celu poprawy zgodności z adapterem Keycloak:

Wydaje się, że *Universal Links w iOS* działa bardziej niezawodnie, gdy `response-mode` ustawiony jest na `query`.

Aby uniemożliwić Androidowi otwarcie nowej instancji aplikacji podczas przekierowania, dodaj następujący fragment do pliku `config.xml`:

```
    <preference name="AndroidLaunchMode" value="singleTask" />
```

2.2.4. Wcześniejsze przeglądarki

Adapter JavaScript zależy od *Base64* (`window.btoa` i `window.atob`), interfejsu *HTML5 History API* i opcjonalnie interfejsu *Promise API*. Jeśli chcesz obsługiwać przeglądarki, które nie mają ich dostępnych (na przykład IE9), musisz dodać dodatkowe biblioteki. Przykładowo:

- Base64 – https://github.com/davidchambers/Base64.js
- HTML5 History – https://github.com/devote/HTML5-History-API
- Promise – https://github.com/stefanpenner/es6-promise

2.2.5. Dokumentacja adaptera JavaScript

2.2.5.1. Konstruktor

```
   new Keycloak();
   new Keycloak('http://localhost/keycloak.json');
   new Keycloak({ url: 'http://localhost/auth', realm: 'myrealm', clientId:
'myApp' });
```

2.2.5.2. Pola (ang. properties)

Nazwa pola	Opis
authenticated	true, jeśli użytkownik jest uwierzytelniony.
token	Token zakodowany base64, który może być wysyłany w nagłówku `Authorization` w żądaniach do usług.
tokenParsed	Przetworzony token jako obiekt JavaScript.
subject	The user id.
idToken	The base64 encoded ID token.
idTokenParsed	The parsed id token as a JavaScript object.
realmAccess	The realm roles associated with the token.
resourceAccess	Role zasobów powiązane z tokenem.
refreshToken	Token odświeżania zakodowany w standardzie base64, którego można użyć do uzyskania nowego tokena.
refreshTokenParsed	Przetworzony token odświeżania jako obiekt JavaScript.

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

timeSkew	Szacowana różnica czasu między czasem przeglądarki a serwerem Keycloak w sekundach. Ta wartość jest tylko szacunkiem, ale jest wystarczająco dokładna przy ustalaniu, czy token wygasł, czy nie.
responseMode	Tryb odpowiedzi przekazany w init (wartość domyślna to `fragment`).
flow	Przepływ ustalony w `init`.
adapter	Umożliwia zastąpienie sposobu, w jaki biblioteka będzie obsługiwać przekierowania i inne funkcje związane z przeglądarką. Dostępne opcje: • "default" – biblioteka korzysta z interfejsu przeglądarki do przekierowań (jest to ustawienie domyślne), • "cordova" – biblioteka spróbuje użyć wtyczki cordova InAppBrowser, aby załadować strony logowania i rejestracji keycloak (ta funkcja jest używana automatycznie, gdy biblioteka działa w ekosystemie Cordova), • "cordova-native" – biblioteka próbuje otworzyć stronę logowania i rejestracji za pomocą przeglądarki systemowej telefonu używając wtyczki BrowserTabs cordova; wymaga to dodatkowej konfiguracji przekierowania z powrotem do aplikacji (patrz Aplikacje hybrydowe z Cordova), • "custom" – umożliwia wdrożenie niestandardowego adaptera (tylko w zaawansowanych przypadkach użycia).
responseType	Typ odpowiedzi wysyłany do Keycloak z żądaniami logowania. Jest to określane na podstawie wartości przepływu używanej podczas inicjalizacji, ale można to zmienić, ustawiając ten parametr.

2.2.5.3. Metody

init(options)

Wywoływana w celu zainicjowania adaptera.

`options` to obiekt, w którym:

- `useNonce` – dodaje kod kryptograficzny, aby sprawdzić, czy odpowiedź uwierzytelnienia odpowiada żądaniu (domyślnie jest to `true`).
- `onLoad` – określa akcję do wykonania podczas ładowania. Obsługiwane wartości to `login-required` lub `check-sso`.
- `silentCheckSsoRedirectUri` – ustawia URI przekierowania dla cichego uwierzytelniania, jeśli `onLoad` jest ustawiony na `check-sso`.
- `token` – ustawia wartość początkową tokena dostępu.
- `refreshToken` – ustawia wartość początkową tokena odświeżania.
- `idToken` – ustawia wartość początkową id tokena (tylko razem z `token` lub `refreshToken`).
- `timeSkew` – ustawia wartość początkową przesunięcia między czasem lokalnym a serwerem Keycloak w sekundach (tylko razem z `token` lub `refreshToken`).
- `checkLoginIframe` – ustawia, aby włączyć monitorowanie stanu logowania (domyślnie `true`).
- `checkLoginIframeInterval` – ustawia interwał sprawdzania stanu logowania (domyślnie 5 sekund).
- `responseMode` – ustawia tryb odpowiedzi *OpenID Connect* wysyłany do serwera Keycloak na żądanie logowania. Prawidłowe wartości to `query` lub `fragment`. Wartość domyślna to `fragment`, która oznacza, że po udanym uwierzytelnieniu Keycloak przekieruje do aplikacji JavaScript z parametrami OpenID Connect dodanymi w fragmencie adresu URL. Jest to generalnie bezpieczniejsze i bardziej zalecane niż `query`.
- `flow` – ustawia przepływ *OpenID Connect*. Prawidłowe wartości to `standard`, `implicit` lub `hybrid`.
- `enableLogging` – włącza rejestrowanie wiadomości z Keycloak do konsoli (domyślnie `false`).
- `pkceMethod` – metoda użycia Proof Key Code Exchange (PKCE). Skonfigurowanie tej wartości włącza mechanizm PKCE. Dostępne opcje:
  - S256 – Metoda PKCE oparta na SHA256.

Zwraca obietnicę, która rozwiązuje się po zakończeniu inicjalizacji.

login(options)

Przekierowuje do formularza logowania.

`options` to obiekt, w którym:

- `redirectUri` – Określa identyfikator URI, na który przeglądarka ma się przekierować po zalogowaniu.
- `prompt` – Ten parametr pozwala na nieznaczne dostosowanie przepływu logowania po stronie serwera Keycloak. Na przykład pozwala wymusić wyświetlanie ekranu logowania dla wartości `logon`. Szczegółowe informacje można znaleźć w rozdziale Przekazywanie parametrów.
- `maxAge` – Używane tylko wtedy, gdy użytkownik jest już uwierzytelniony. Określa maksymalny czas od uwierzytelnienia użytkownika. Jeśli użytkownik jest już uwierzytelniony przez dłuższy czas niż maxAge, autoryzacja SSO jest ignorowana i użytkownik będzie musiał się ponownie uwierzytelnić.
- `loginHint` – Służy do wstępnego wypełnienia pola nazwy użytkownika/adresu e-mail w formularzu logowania.
- `scope` – Służy do przekazywania parametru zakresu do punktu końcowego logowania Keycloak.
- Użyj listy zakresów rozdzielonej spacjami. Zazwyczaj odnoszą się one do zakresów zdefiniowanych dla konkretnego klienta. Należy zauważyć, że zakres openid zawsze będzie dodawany do listy zakresów przez adapter. Na przykład, jeśli przekażesz `'address phone'`, wówczas żądanie do Keycloak będzie zawierało parametr `scope=openid address phone`.
- `idpHint` – Służy do informowania Keycloak o pominięciu wyświetlania strony logowania i automatycznym przekierowaniu do określonego dostawcy tożsamości. Więcej informacji w dokumentacji dostawcy tożsamości.
- `action` – Dla `register` użytkownik zostanie przekierowany na stronę rejestracji, w innym przypadku na stronę logowania.
- `locale` – Ustawia parametr zapytania `"ui_locales"` zgodnie z sekcją *3.1.2.1* specyfikacji *OIDC 1.0*.
- `kcLocale` – Określa pożądane ustawienia regionalne Keycloak dla interfejsu użytkownika. Różni się to od parametru `locale` tym, że nakazuje serwerowi Keycloak ustawienie pliku cookie i zaktualizowanie profilu użytkownika do nowej preferowanej lokalizacji.
- `cordovaOptions` – Określa argumenty przekazywane do przeglądarki *Cordova in-app-browser* (jeśli dotyczy). Argumenty te nie mają wpływu na opcje ukryte i lokalizację. Wszystkie dostępne opcje są zdefiniowane w dokumentacji *cordova-plugin-inappbrowser*. Przykład użycia: `{ zoom: "no", hardwareback: "yes" };`

createLoginUrl(options)

Zwraca adres URL formularza logowania.

`options` to obiekt, który obsługuje takie same parametry, jak funkcja `login`.

logout(options)

Przekierowuje do strony wylogowania.

`options` to obiekt, w którym:

- `redirectUri` – Określa identyfikator URI, na który przeglądarka ma się przekierować po wylogowaniu.

createLogoutUrl(options)

Zwraca adres URL strony wylogowania.

`options` to obiekt, który obsługuje takie same parametry, jak funkcja `logout`.

register(options)

Przekierowuje do strony rejestracji. Ekwiwalent funkcji `login({action: 'register'})`.

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

`options` to obiekt, który obsługuje takie same parametry, jak funkcja `login`.

createRegisterUrl(options)

Zwraca adres URL strony rejestracji. Ekwiwalent funkcji `createLoginUrl({action: 'register'})`.

`options` to obiekt, który obsługuje takie same parametry, jak funkcja `createLoginUrl`.

accountManagement()

Przekierowuje do strony do zarządzania profilem konta (*ang. Account Management Console*).

createAccountUrl()

Zwraca adres URL strony do zarządzania profilem konta (*ang. Account Management Console*).

hasRealmRole(role)

Zwraca wartość `true`, jeśli token ma daną rolę strefy (*ang. realm role*).

hasResourceRole(role, resource)

Zwraca wartość `true`, jeśli token ma daną rolę dla zasobu. `resource` jest opcjonalny, domyślnie przyjmowany jest `clientId`.

loadUserProfile()

Wczytuje profil użytkownika.

Zwraca obietnicę, która rozwiązuje się obiektem profilu.

Na przykład:

```
keycloak.loadUserProfile()
    .then(function(profile) {
        alert(JSON.stringify(profile, null, "  "))
    }).catch(function() {
        alert('Failed to load user profile');
    });
```

isTokenExpired(minValidity)

Zwraca wartość `true`, jeśli token ma mniej niż `minValidity` sekund ważności (`minValidity` jest opcjonalny, domyślnie przyjmuje `0`).

updateToken(minValidity)

Jeżeli token przeterminowałby się w czasie `minValidity` sekund (`minValidity` jest opcjonalny, domyślnie przyjmuje 5), odświeża token. Jeżeli status sesji iframe jest włączony, to status sesji jest również sprawdzany.

Zwraca obietnicę, która rozwiązuje się do wartości logicznej, w której `true` oznacza, że token został odświeżony.

Na przykład:

```
keycloak.updateToken(5)
    .then(function(refreshed) {
        if (refreshed) {
            alert('Token was successfully refreshed');
        } else {
            alert('Token is still valid');
        }
    }).catch(function() {
        alert('Failed to refresh the token, or the session has expired');
    });
```

clearToken()

Wyczyść stan uwierzytelnienia, w tym tokeny. Może to być przydatne, jeśli aplikacja wykryła, że sesja wygasła, na przykład jeśli aktualizacja tokena nie powiodła się.

Wywołanie powoduje wywołanie detektora wywołania zwrotnego onAuthLogout.

2.2.5.4. Wywołania zwrotne

Adapter obsługuje ustawianie funkcji zwrotnych dla niektórych zdarzeń.

Na przykład:

```
keycloak.onAuthSuccess = function() { alert('authenticated'); }
```

Dostępne zdarzenia to:

- `onReady(authenticated)` – Wywoływany po inicjalizacji adaptera.
- `onAuthSuccess` – Wywoływany, gdy użytkownik zostanie pomyślnie uwierzytelniony.
- `onAuthError` – Wywoływany, jeśli wystąpił błąd podczas uwierzytelniania.
- `onAuthRefreshSuccess` – Wywoływany, gdy token jest odświeżany.
- `onAuthRefreshError` – Wywoływany, jeśli wystąpił błąd podczas próby odświeżenia tokena.
- `onAuthLogout` – Wywoływany, gdy użytkownik jest wylogowany (będzie wywoływany tylko wtedy, gdy włączony jest statusu sesji iframe lub w trybie Cordova).
- `onTokenExpired` – Wywoływany, gdy token dostępu wygasł. Jeśli token odświeżania jest dostępny, token można odświeżyć za pomocą `updateToken` lub w przypadkach, gdy nie jest (tj. implicite flow), możesz przekierować na ekran logowania, aby uzyskać nowy token dostępu.

2.3. Adapter Node.js

Keycloak zapewnia adapter Node.js, umożliwiający m.in. integrację z frameworkami takimi jak Express.js.

Aby użyć adaptera Node.js, najpierw musisz utworzyć klienta dla swojej aplikacji w konsoli administracyjnej Keycloak. Adapter obsługuje typ dostępu publicznego (*ang. public*), poufnego (*ang. confidential*) i tylko na okaziciela (*ang. bearer-only*).

Po utworzeniu klienta kliknij kartę `Installation`, w polu `Format Option` wybierz `Keycloak OIDC JSON`, a następnie `Download`. Pobrany plik `keycloak.json` powinien znajdować się w folderze głównym projektu.

2.3.1. Instalacja

Zakładamy, że Node.js jest zainstalowany i projekt *npm* dla Twojej aplikacji istnieje.

Dodajemy adapter Keycloak Connect do listy zależności:

```
"dependencies": {
    "keycloak-connect": "10.0.2"
}
```

2.3.2. Użycie

Utwórz obiekt klasy Keycloak. Klasa Keycloak stanowi centralny punkt konfiguracji i integracji z aplikacją. Najprostsze wdrożenie nie wymaga żadnych argumentów.

```
var session = require('express-session');
var Keycloak = require('keycloak-connect');
var memoryStore = new session.MemoryStore();
var keycloak = new Keycloak({ store: memoryStore });
```

Domyślnie plik o nazwie `keycloak.json` będzie szukany obok głównego pliku wykonywalnego aplikacji w celu zainicjowania ustawień specyficznych dla keycloak (klucz publiczny, nazwa dziedziny, różne adresy URL). Plik `keycloak.json` możesz pobrać z konsoli administracyjnej Keycloak.

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

Utworzenie instancji za pomocą tej metody powoduje użycie wszystkich rozsądnych wartości domyślnych. Alternatywnie można również podać obiekt konfiguracyjny zamiast pliku `keycloak.json`:

```
        let kcConfig = {
            clientId: 'myclient',
            bearerOnly: true,
            serverUrl: 'http://localhost:8080/auth',
            realm: 'myrealm',
            realmPublicKey: 'MIIBIjANB...'
        };
        let keycloak = new Keycloak({ store: memoryStore }, kcConfig);
```

Aplikacje mogą również przekierowywać użytkowników do preferowanego dostawcy tożsamości:

```
        let keycloak = new Keycloak({ store: memoryStore, idpHint: myIdP },
kcConfig);
```

2.3.2.1. Konfigurowanie magazynu sesji

Jeśli chcesz używać sesji web do zarządzania stanem po stronie serwera w celu uwierzytelnienia, musisz zainicjować `Keycloak(…)` co najmniej z parametrem `store`, przekazując rzeczywisty magazyn sesji express-session, z którego korzystasz.

```
        var session = require('express-session');
        var memoryStore = new session.MemoryStore();
        var keycloak = new Keycloak({ store: memoryStore });
```

2.3.2.2. Przekazywanie niestandardowej wartości zakresu

Domyślnie jako `scope` przekazywany jest `openid` w zapytaniu do adresu URL logowania Keycloak, ale możesz dodać dodatkową wartość niestandardową:

```
        var keycloak = new Keycloak({ scope: 'offline_access' });
```

2.3.3. Instalowanie Middleware

Po utworzeniu instancji zainstaluj middleware w aplikacji connect-capable:

```
        var app = express();
        app.use( keycloak.middleware() );
```

2.3.4. Sprawdzanie uwierzytelnienia

Aby sprawdzić, czy użytkownik jest uwierzytelniony przed uzyskaniem dostępu do zasobu, wystarczy użyć `keycloak.checkSso()`. Uwierzytelni się tylko, jeśli użytkownik jest już zalogowany. Jeśli użytkownik nie jest zalogowany, przeglądarka zostanie przekierowana z powrotem na pierwotnie żądany adres URL i pozostanie nieuwierzytelniona:

```
        app.get( '/check-sso', keycloak.checkSso(), checkSsoHandler );
```

2.3.5. Ochrona zasobów

Proste uwierzytelnianie

Aby wymusić uwierzytelnienie użytkownika przed dostępem do zasobu, po prostu użyj `keycloak.protect()` bez argumentów:

```
        app.get( '/complain', keycloak.protect(), complaintHandler );
```

Autoryzacja oparta na rolach

Aby zabezpieczyć zasób za pomocą roli aplikacji (ang. *application role*):

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

```
app.get( '/special', keycloak.protect('special'), specialHandler );
```

Aby zabezpieczyć zasób za pomocą roli aplikacji, ale pochodzącej z innej aplikacji:

```
app.get( '/extra-special', keycloak.protect('other-app:special'),
extraSpecialHandler );
```

Aby zabezpieczyć zasób z rolą strefy (*ang. realm role*):

```
app.get( '/admin', keycloak.protect( 'realm:admin' ), adminHandler );
```

Autoryzacja oparta na zasobach

Autoryzacja oparta na zasobach pozwala chronić zasoby oraz ich specyficzne akcje/polecenia w oparciu o zestaw polityk zdefiniowanych w Keycloak, tym samym wyodrębniając autoryzację z twojej aplikacji. Osiąga się to poprzez ujawnienie metody keycloak.enforcer, której można użyć do ochrony zasobów.

```
app.get('/apis/me', keycloak.enforcer('user:profile'), userProfileHandler);
```

Metoda keycloak-enforcer działa w dwóch trybach, w zależności od wartości opcji konfiguracyjnej `response_mode`.

```
app.get('/apis/me', keycloak.enforcer('user:profile', {response_mode:
'token'}), userProfileHandler);
```

Jeśli `response_mode` ustawiony jest na `token`, uprawnienia są uzyskiwane z serwera w imieniu podmiotu reprezentowanego przez token na okaziciela (ang. *bearer token*), który został wysłany do Twojej aplikacji. W takim przypadku Keycloak wydaje nowy token dostępu z uprawnieniami nadanymi przez serwer. Jeśli serwer nie odpowiedział tokenem z oczekiwanymi uprawnieniami, żądanie zostanie odrzucone. Korzystając z tego trybu, można uzyskać token z żądania w następujący sposób:

```
app.get('/apis/me', keycloak.enforcer('user:profile', {response_mode:
'token'}), function (req, res) {
    var token = var token = req.kauth.grant.access_token.content;
    var permissions = token.authorization ? token.authorization.permissions
: undefined;
    // show user profile
});
```

Preferuj ten tryb, gdy aplikacja korzysta z sesji i chcesz buforować poprzednie decyzje z serwera, a także automatycznie obsługiwać tokeny odświeżania. Ten tryb jest szczególnie przydatny w przypadku aplikacji działających jako klient i serwer zasobów.

Jeśli `response_mode` jest ustawiony na `permissions` (tryb domyślny), serwer zwraca tylko listę przyznanych uprawnień, bez wydawania nowego tokena dostępu. Ta metoda przekazuje uprawnienia nadane przez serwer za pośrednictwem następującego żądania:

```
app.get('/apis/me', keycloak.enforcer('user:profile', {response_mode:
'permissions'}), function (req, res) {
    var permissions = req.permissions;
    // show user profile
});
```

Niezależnie od używanego `response_mode` metoda `keycloak.enforcer` najpierw spróbuje sprawdzić uprawnienia w ramach tokenu elementu nośnego, który został wysłany do aplikacji. Jeśli token nośny ma już oczekiwane uprawnienia, nie ma potrzeby interakcji z serwerem w celu uzyskania decyzji. Jest to szczególnie przydatne, gdy Twoi klienci są w stanie uzyskać tokeny dostępu z serwera z oczekiwanymi uprawnieniami przed uzyskaniem dostępu do chronionego zasobu, dzięki czemu mogą korzystać z niektórych funkcji zapewnianych przez Usługi autoryzacji Keycloak, takich jak autoryzacja przyrostowa i unikać dodatkowych żądań do serwera gdy `keycloak.enforcer` nadzoruje dostęp do zasobu.

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

Domyślnie moduł egzekwujący zasady użyje `client_id` zdefiniowanego dla aplikacji (na przykład przez `keycloak.json`). W takim przypadku klient nie może być publiczny, biorąc pod uwagę, że w rzeczywistości jest to serwer zasobów.

Jeśli aplikacja działa zarówno jako klient publiczny (frontend), jak i serwer zasobów (backend), możesz użyć następującej konfiguracji, aby odwoływać się do innego klienta w Keycloak za pomocą zasad, które chcesz egzekwować:

```
keycloak.enforcer('user:profile', {resource_server_id: 'my-apiserver'})
```

Zaleca się używanie różnych klientów w Keycloak do reprezentowania frontendu i backendu.

Jeśli chroniona aplikacja jest włączona z usługami autoryzacji Keycloak i zdefiniowałeś poświadczenia klienta w `keycloak.json`, możesz przesłać dodatkowe roszczenia do serwera i udostępnić je swoim politykom w celu podjęcia decyzji. W tym celu możesz zdefiniować opcję konfiguracji oświadczeń, która oczekuje funkcji, która zwraca JSON z oświadczeniami, które chcesz przekazać:

```
        app.get('/protected/resource', keycloak.enforcer(['resource:view',
'resource:write'], {
            claims: function(request) {
              return {
                "http.uri": ["/protected/resource"],
                "user.agent": // get user agent  from request
              }
            }
          }), function (req, res) {
            // access granted
```

Aby uzyskać więcej informacji na temat konfigurowania Keycloak w celu ochrony zasobów aplikacji, zapoznaj się z Przewodnikiem po usługach autoryzacyjnych.

Zaawansowana autoryzacja

Aby zabezpieczyć zasoby na podstawie części adresu URL, zakładając, że dla każdej sekcji istnieje rola:

```
      function protectBySection(token, request) {
        return token.hasRole( request.params.section );
      }
      app.get( '/:section/:page', keycloak.protect( protectBySection ),
sectionHandler );
```

2.3.6. Dodatkowe adresy URL

Jawne wylogowanie uruchamiane przez użytkownika

Domyślnie oprogramowanie pośrednie przechwytuje wywołania do `/logout`. Można to zmienić, określając parametr konfiguracyjny `logout` dla `middleware()`:

```
      app.use( keycloak.middleware( { logout: '/logoff' } ));
```

Wywołania zwrotne Keycloak Admin

Oprogramowanie pośrednie obsługuje połączenia zwrotne z konsoli Keycloak w celu wylogowania jednej sesji lub wszystkich sesji. Domyślnie tego rodzaju wywołania zwrotne administratora występują w odniesieniu do głównego adresu URL `/`, ale można je zmienić, podając parametr `admin` do wywołania `middleware()`:

```
      app.use( keycloak.middleware( { admin: '/callbacks' } );
```

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

2.4. Moduł mod_auth_openidc dla Apache HTTPD

Moduł `mod_auth_openidc` to wtyczka do HTTP Apache obsługująca OpenID Connect. Jeśli twoja architektura wykorzystuje Apache HTTPD jako serwer proxy, możesz użyć mod_auth_openidc do zabezpieczenia swojej aplikacji internetowej za pomocą OpenID Connect.

Aby skonfigurować `mod_auth_openidc` potrzebujesz:

- `client_id`,
- `client_secret`,
- `redirect_uri` do twojej aplikacji,
- adres URL z konfiguracją openid.

Przykładowa konfiguracja wygląda następująco:

```
   LoadModule auth_openidc_module modules/mod_auth_openidc.so
   ServerName ${HOSTIP}

   <VirtualHost *:80>
       ServerAdmin webmaster@localhost
       DocumentRoot /var/www/html

       #this is required by mod_auth_openidc
       OIDCCryptoPassphrase a-random-secret-used-by-apache-oidc-and-balancer

       OIDCProviderMetadataURL ${KC_ADDR}/auth/realms/${KC_REALM}/.well-
known/openid-configuration

       OIDCClientID ${CLIENT_ID}
       OIDCClientSecret ${CLIENT_SECRET}
       OIDCRedirectURI http://${HOSTIP}/${CLIENT_APP_NAME}/redirect_uri

       # maps the prefered_username claim to the REMOTE_USER environment variable
       OIDCRemoteUserClaim preferred_username

       <Location /${CLIENT_APP_NAME}/>
           AuthType openid-connect
           Require valid-user
       </Location>
   </VirtualHost>
```

Konfiguracja tego modułu wykracza poza zakres tego dokumentu. Więcej informacji na temat konfigurowania `mod_auth_openidc` można znaleźć na stronie projektu oraz w repozytorium `mod_auth_openidc` na GitHub.

2.5. Inne biblioteki OpenID Connect

Keycloak można zabezpieczyć za pomocą dostarczonych adapterów, które zwykle są łatwiejsze w użyciu i zapewniają lepszą integrację z Keycloak. Jeśli jednak adapter nie jest dostępny dla twojego języka programowania, frameworku lub platformy, możesz zamiast tego użyć ogólnej biblioteki *OpenID Connect Relying Party (RP)*. Ten rozdział opisuje szczegółowe informacje dotyczące Keycloak i nie zawiera szczegółowych informacji o protokole. Aby uzyskać więcej informacji, zobacz specyfikacje *OpenID Connect* i specyfikację *OAuth2*.

2.5.1. Punkty końcowe

Najważniejszy jest punkt końcowy konfiguracji `well-known`. Zawiera listę punktów końcowych i inne opcje konfiguracji istotne dla implementacji *OpenID Connect* w Keycloak. Punktem końcowym jest:

```
   /realms/{realm-name}/.well-known/openid-configuration
```

Aby uzyskać pełny adres URL, dodaj podstawowy adres URL Keycloak i zastąp {realm-name} nazwą swojej strefy. Na przykład:

```
https://auth.domena.pl/auth/realms/master/.well-known/openid-configuration
```

Niektóre biblioteki RP pobierają wszystkie wymagane punkty końcowe z tego punktu końcowego, ale w przypadku innych konieczne może być podanie listy punktów końcowych indywidualnie.

Punkt końcowy autoryzacji

```
/realms/{realm-name}/protocol/openid-connect/auth
```

Punkt końcowy autoryzacji wykonuje uwierzytelnianie użytkownika końcowego. Odbywa się to poprzez przekierowanie klienta użytkownika do tego punktu końcowego.

Aby uzyskać więcej informacji, zobacz sekcję *Authorization Endpoint* w specyfikacji *OpenID Connect*.

Punkt końcowy tokena

```
/realms/{realm-name}/protocol/openid-connect/token
```

Punkt końcowy tokenu służy do uzyskiwania tokenów. Tokeny można uzyskać, wymieniając kod autoryzacyjny lub dostarczając poświadczenia bezpośrednio, w zależności od używanego przepływu. Punkt końcowy tokenu służy również do uzyskiwania nowych tokenów dostępu po ich wygaśnięciu.

Aby uzyskać więcej informacji, zobacz sekcję *Token Endpoint* w specyfikacji *OpenID Connect*.

Punkt końcowy informacji o użytkowniku

```
/realms/{realm-name}/protocol/openid-connect/userinfo
```

Punkt końcowy `userinfo` zwraca standardowe oświadczenia dotyczące uwierzytelnionego użytkownika i jest chroniony przez token na okaziciela.

Aby uzyskać więcej informacji, zobacz sekcję *Userinfo Endpoint* w specyfikacji *OpenID Connect*.

Punkt końcowy wylogowania

```
/realms/{realm-name}/protocol/openid-connect/logout
```

Punkt końcowy wylogowania wylogowuje uwierzytelnionego użytkownika.

Agent użytkownika może zostać przekierowany do punktu końcowego, w którym to przypadku aktywna sesja użytkownika zostanie wylogowana. Następnie agent użytkownika zostaje przekierowany z powrotem do aplikacji.

Punkt końcowy można również wywołać bezpośrednio przez aplikację. Aby bezpośrednio wywołać ten punkt końcowy, należy dołączyć token odświeżania, a także poświadczenia wymagane do uwierzytelnienia klienta.

Punkt końcowy certyfikatu

```
/realms/{realm-name}/protocol/openid-connect/certs
```

Punkt końcowy certyfikatu zwraca klucze publiczne włączone przez strefę, zakodowane jako *JSON Web Key (JWK)*. W zależności od ustawień dziedziny może być włączony jeden lub więcej kluczy do weryfikacji tokenów. Aby uzyskać więcej informacji, zobacz Podręcznik administracji serwera i specyfikację *JSON Web Key*.

Punkt końcowy introspekcji

```
/realms/{realm-name}/protocol/openid-connect/token/introspect
```

Punkt końcowy introspekcji służy do pobierania stanu aktywnego tokena. Innymi słowy, możesz go użyć do sprawdzenia poprawności tokena dostępu lub odświeżenia. Może być wywoływane tylko przez poufnych klientów.

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

Aby uzyskać więcej informacji na temat sposobu wywoływania w tym punkcie końcowym, zobacz specyfikację *OAuth 2.0 Token Introspection*.

Punkt końcowy dynamicznej rejestracji klienta

```
/realms/{realm-name}/clients-registrations/openid-connect
```

Dynamiczny punkt końcowy rejestracji klienta służy do dynamicznej rejestracji klientów.

Aby uzyskać więcej informacji, zobacz rozdział Client Registration i specyfikację OpenID Connect Dynamic Client Registration.

Punkt końcowy odwołania tokenu

```
/realms/{realm-name}/protocol/openid-connect/revoke
```

Punkt końcowy odwołania tokenu służy do unieważnienia tokenu.

Aby uzyskać więcej informacji na temat sposobu wywoływania w tym punkcie końcowym, zobacz specyfikację *OAuth 2.0 Token Revocation*.

2.5.2. Sprawdzanie poprawności tokenów dostępu

Jeśli musisz ręcznie sprawdzić poprawność tokenów dostępu wydanych przez Keycloak, możesz wywołać punkt końcowy introspekcji. Wadą tego podejścia jest konieczność połączenia sieciowego do serwera Keycloak. Może to być powolne i może spowodować przeciążenie serwera, jeśli masz zbyt wiele żądań weryfikacji w tym samym czasie. Tokeny dostępu wydane przez Keycloak to JSON Web Token (JWT) podpisane cyfrowo i zakodowane przy użyciu JSON Web Signature (JWS). Ponieważ są one w ten sposób zabezpieczone, pozwala to na lokalną weryfikację tokenów dostępu przy użyciu klucza publicznego strefy wydającej. Możesz albo umieścić na stałe klucz publiczny strefy (*ang. realm*) w kodzie weryfikacyjnym, albo wyszukać i buforować klucz publiczny przy użyciu punktu końcowego certyfikatu z identyfikatorem klucza (KID) osadzonym w JWS. W zależności od języka, w którym kodujesz, istnieje wiele bibliotek stron trzecich, które mogą pomóc w sprawdzaniu poprawności JWS.

2.5.3. Przepływy (*ang. flows*)

Kod autoryzacji *(ang. Authorization Code)*

Przepływ kodu autoryzacji przekierowuje klienta użytkownika do Keycloak. Po pomyślnym uwierzytelnieniu użytkownika za pomocą Keycloak tworzony jest kod autoryzacji, a klient użytkownika zostaje przekierowany z powrotem do aplikacji. Następnie aplikacja używa kodu autoryzacji wraz z poświadczeniami do uzyskania tokena dostępu, tokena odświeżania i tokena identyfikacyjnego od Keycloak.

Przepływ jest ukierunkowany na aplikacje internetowe, ale jest również zalecany w przypadku aplikacji natywnych, w tym aplikacji mobilnych, w których można osadzić klienta użytkownika.

Aby uzyskać więcej informacji, patrz *Authorization Code Flow* w specyfikacji *OpenID Connect*.

Implicit

Przekierowanie implicit działa podobnie do przepływu kodu autoryzacji, ale zamiast zwracać kod autoryzacji, zwracany jest token dostępu i token identyfikatora. Zmniejsza to potrzebę dodatkowego wywołania w celu wymiany kodu autoryzacji na token dostępu. Jednak nie zawiera tokenu odświeżania. Powoduje to konieczność zezwolenia na tokeny dostępu z długim terminem ważności, co jest problematyczne, ponieważ bardzo trudno je unieważnić. Wymaga też nowego przekierowania w celu uzyskania nowego tokena dostępu po wygaśnięciu początkowego tokena dostępu. Przepływ implicit jest przydatny, jeśli aplikacja chce tylko uwierzytelnić użytkownika i sama zajmie się jego wylogowaniem.

Istnieje również przepływ hybrydowy, w którym zwracany jest zarówno token dostępu, jak i kod autoryzacyjny.

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

Należy zauważyć, że zarówno przepływ implicit, jak i przepływ hybrydowy stanowią potencjalne zagrożenie bezpieczeństwa, ponieważ token dostępu (przekazywany w URL) może wycieknąć z dzienników serwera WWW i historii przeglądarki. Jest to nieco złagodzone, gdy użyjemy krótkiego terminu ważności tokenów dostępu.

Aby uzyskać więcej informacji, zobacz *Implicit Flow* w specyfikacji *OpenID Connect*.

Poświadczenie hasła właściciela zasobu

Poświadczenie hasła właściciela zasobu, zwane `Direct Grant` w Keycloak, umożliwia wymianę poświadczeń użytkownika na tokeny. Nie zaleca się korzystania z tego przepływu, chyba że jest to absolutnie konieczne. Przykłady, w których może to być przydatne, to starsze aplikacje i interfejsy wiersza polecenia.

Istnieje wiele ograniczeń korzystania z tego przepływu, w tym:

1. Poświadczenia użytkownika są udostępniane aplikacji.
2. Aplikacje wymagają stron logowania.
3. Aplikacja musi znać schemat uwierzytelniania.
4. Zmiany w procesie uwierzytelniania wymagają zmian w aplikacji.
5. Brak obsługi pośrednictwa tożsamości i logowania społecznościowego.
6. Przepływy nie są obsługiwane (samodzielna rejestracja użytkownika, wymagane działania itp.).

Aby klient mógł korzystać z poświadczenia hasła właściciela zasobu, musi mieć włączoną opcję przyznania bezpośredniego dostępu.

Ten przepływ nie jest uwzględniony w OpenID Connect, ale jest częścią specyfikacji OAuth 2.0.

Aby uzyskać więcej informacji, zapoznaj się z rozdziałem Resource Owner Password Credentials Grant w specyfikacji OAuth 2.0.

Przykład użycia CURL

Poniższy przykład pokazuje, jak uzyskać token dostępu dla użytkownika w głownej strefie z nazwą użytkownika i hasłem. Przykład używa poufnego klienta `myclient`:

```
curl \
  -d "client_id=myclient" \
  -d "client_secret=40cc097b-2a57-4c17-b36a-8fdf3fc2d578" \
  -d "username=user" \
  -d "password=password" \
  -d "grant_type=password" \
  "http://localhost:8080/auth/realms/master/protocol/openid-connect/token"
```

Poświadczenia klienta

Poświadczenia klientów są używane, gdy klienci (aplikacje i usługi) chcą uzyskać dostęp w imieniu własnym, a nie w imieniu użytkownika. Może to być na przykład przydatne w przypadku usług w tle, które stosują zmiany w systemie ogólnie, a nie dla konkretnego użytkownika.

Keycloak zapewnia klientom wsparcie w zakresie uwierzytelniania przy użyciu klucza tajnego lub pary kluczy publicznego i prywatnego.

Ten przepływ nie jest uwzględniony w *OpenID Connect*, ale jest częścią specyfikacji *OAuth 2.0*.

Aby uzyskać więcej informacji, zapoznaj się z rozdziałem Client Credentials Grant w specyfikacji OAuth 2.0.

2.5.4. Identyfikatory URI przekierowań

Podczas korzystania z przepływów opartych na przekierowaniu ważne jest, aby używać prawidłowego adresu przekierowania dla swoich klientów. Adres przekierowania powinien być jak najbardziej szczegółowy. Dotyczy to zwłaszcza aplikacji po stronie klienta (klientów publicznych). Niezastosowanie się do tego może spowodować:

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

Otwarte przekierowania – może to pozwolić atakującym na tworzenie fałszywych linków, które wyglądają, jakby pochodziły z Twojej domeny.

Nieautoryzowany wpis – gdy użytkownicy są już uwierzytelnieni za pomocą Keycloak, osoba atakująca może użyć klienta publicznego, w którym adres przekierowania nie został poprawnie skonfigurowane, aby uzyskać dostęp poprzez przekierowanie użytkowników bez ich wiedzy.

W produkcji dla aplikacji internetowych zawsze używaj HTTPS dla wszystkich identyfikatorów URI przekierowań. Nie zezwalaj na przekierowania na HTTP.

Istnieje również kilka specjalnych identyfikatorów URI przekierowań:

`http://localhost` – Ten identyfikator URI przekierowania jest przydatny dla aplikacji natywnych i pozwala aplikacji rodzimej na utworzenie serwera WWW na losowym porcie, którego można użyć do uzyskania kodu autoryzacji. Ten identyfikator przekierowania pozwala na dowolny port.

`urn:ietf:wg:oauth:2.0:oob` – Jeśli nie można uruchomić serwera WWW w kliencie (lub przeglądarka nie jest dostępna), można użyć specjalnego adresu przekierowania `urn:ietf:wg:oauth:2.0:oob`. Gdy używa się tego przekierowania, Keycloak wyświetla stronę z kodem w tytule oraz w polu na stronie. Aplikacja może albo wykryć zmianę tytułu przeglądarki, albo użytkownik może skopiować i wkleić kod ręcznie do aplikacji. Dzięki temu identyfikatorowi przekierowania użytkownik może również użyć innego urządzenia, aby uzyskać kod do wklejenia z powrotem do aplikacji.

3. SAML

W tej sekcji opisano, w jaki sposób można zabezpieczyć aplikacje i usługi za pomocą SAML przy użyciu adapterów klienta Keycloak lub ogólnych bibliotek SAML.

3.1. Adaptery Java

Keycloak jest wyposażony w szereg różnych adapterów do aplikacji Java. Wybór odpowiedniego adaptera zależy od platformy docelowej.

3.1.1. Ogólna konfiguracja adaptera

Każdy adapter klienta SAML obsługiwany przez Keycloak można skonfigurować za pomocą prostego pliku tekstowego XML. Przykładowa zawartość:

```
   <keycloak-saml-adapter xmlns="urn:keycloak:saml:adapter"
                          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                          xsi:schemaLocation="urn:keycloak:saml:adapter
https://www.keycloak.org/schema/keycloak_saml_adapter_1_10.xsd">
       <SP entityID="http://localhost:8081/sales-post-sig/"
           sslPolicy="EXTERNAL"
           nameIDPolicyFormat="urn:oasis:names:tc:SAML:1.1:nameid-
format:unspecified"
           logoutPage="/logout.jsp"
           forceAuthentication="false"
           isPassive="false"
           turnOffChangeSessionIdOnLogin="false"
           autodetectBearerOnly="false">
           <Keys>
               <Key signing="true" >
                   <KeyStore resource="/WEB-INF/keystore.jks" password="store123">
                       <PrivateKey alias="http://localhost:8080/sales-post-sig/"
password="test123"/>
                       <Certificate alias="http://localhost:8080/sales-post-
sig/"/>
```

```
                </KeyStore>
            </Key>
        </Keys>
        <PrincipalNameMapping policy="FROM_NAME_ID"/>
        <RoleIdentifiers>
            <Attribute name="Role"/>
        </RoleIdentifiers>
        <RoleMappingsProvider id="properties-based-role-mapper">
            <Property name="properties.resource.location" value="/WEB-INF/role-
mappings.properties"/>
        </RoleMappingsProvider>
        <IDP entityID="idp"
             signaturesRequired="true">
        <SingleSignOnService requestBinding="POST"

bindingUrl="http://localhost:8081/auth/realms/demo/protocol/saml"
                    />
            <SingleLogoutService
                    requestBinding="POST"
                    responseBinding="POST"

postBindingUrl="http://localhost:8081/auth/realms/demo/protocol/saml"

redirectBindingUrl="http://localhost:8081/auth/realms/demo/protocol/saml"
                    />
            <Keys>
                <Key signing="true">
                    <KeyStore resource="/WEB-INF/keystore.jks"
password="store123">
                        <Certificate alias="demo"/>
                    </KeyStore>
                </Key>
            </Keys>
        </IDP>
    </SP>
</keycloak-saml-adapter>
```

Niektóre z tych przełączników konfiguracji mogą być specyficzne dla adaptera, a niektóre są wspólne dla wszystkich adapterów. W przypadku Java można użyć operatora ${…} do wstawienia właściwości systemowych. Na przykład ${jboss.server.config.dir}.

Element SP

Here is the explanation of the SP element attributes:

```
<SP entityID="sp"
    sslPolicy="ssl"
    nameIDPolicyFormat="format"
    forceAuthentication="true"
    isPassive="false"
    keepDOMAssertion="true"
    autodetectBearerOnly="false">
...
</SP>
```

entityID

To jest identyfikator klienta. IdP potrzebuje tej wartości, aby ustalić, jaki klient się z nim komunikuje. WYMAGANY.

sslPolicy

Polityka wymuszania SSL przez adapter. Poprawne wartości to:

ALL – wszystkie żądania muszą być przesyłane za pośrednictwem protokołu HTTPS,

EXTERNAL – tylko nieprywatne adresy IP muszą być przesyłane przez HTTPS,

NONE – HTTPS nie jest wymagany.

OPCJONALNY. Wartość domyślna to EXTERNAL.

nameIDPolicyFormat

Klienci SAML mogą żądać określonego formatu NameID Subject. Wpisz tę wartość, jeśli chcesz mieć określony format. Musi to być standardowy identyfikator `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`. OPCJONALNY. Domyślnie nie jest wymagany żaden specjalny format.

forceAuthentication

Klienci SAML mogą zażądać ponownego uwierzytelnienia użytkownika, nawet jeśli są już zalogowani na IdP. Ustaw ten atrybut na `true`, aby włączyć. OPCJONALNY. Wartość domyślna to `false`.

isPassive

Klienci SAML mogą zażądać, aby użytkownik nigdy nie był proszony o uwierzytelnienie, nawet jeśli nie są zalogowani w IdP. Nie używaj razem z `forceAuthentication`, ponieważ są one sprzeczne. OPCJONALNY. Wartość domyślna to `false`.

turnOffChangeSessionIdOnLogin

Identyfikator sesji jest domyślnie zmieniany przy udanym logowaniu. Zaleca się, aby nie wyłączać tego mechanizmu. Wartość domyślna to `false`.

autodetectBearerOnly

Ten ustawienie powinno mieć wartość `true`, jeśli aplikacja obsługuje zarówno aplikację internetową, jak i usługi sieciowe (np. *SOAP* lub *REST*). Pozwala przekierowywać nieuwierzytelnionych użytkowników aplikacji internetowej na stronę logowania Keycloak, ale wysyłać też błąd HTTP 401 do nieuwierzytelnionych klientów SOAP lub REST (ponieważ nie rozumieliby przekierowania na stronę logowania). Keycloak automatycznie wykrywa klientów SOAP lub REST na podstawie typowych nagłówków, takich jak `X-Requested-With`, SOAPAction lub `Accept`. Wartość domyślna to `false`.

logoutPage

Ustawia stronę wyświetlaną po wylogowaniu. Jeśli strona jest pełnym adresem URL, takim jak `http://web.example.com/logout.html`, użytkownik jest przekierowywany po wylogowaniu do tej strony przy użyciu HTTP 302. Jeśli określono łącze skrócone, takie jak `/logout.jsp`, strona jest wyświetlana po wylogowaniu, niezależnie od tego, czy znajduje się w obszarze chronionym zgodnie z deklaracjami ograniczeń bezpieczeństwa w pliku `web.xml`.

keepDOMAssertion

Ten atrybut należy ustawić na wartość `true`, aby adapter przechowywał reprezentację DOM potwierdzenia w oryginalnej formie w `SamlPrincipal`. Dokument potwierdzający można pobrać za pomocą metody getAssertionDocument. Jest to szczególnie przydatne podczas ponownego używania podpisanego potwierdzenia. Zwrócony dokument to ten, który został wygenerowany podczas analizowania odpowiedzi SAML otrzymanej przez serwer Keycloak. OPCJONALNY, a wartość domyślna to `false` (dokument nie jest zapisywany w elemencie głównym).

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

3.1.2. Adapter WildFly/JBoss EAP

Aby móc zabezpieczyć aplikacje WAR wdrożone w WildFly lub JBoss EAP, musisz zainstalować i skonfigurować podsystem *SAML Keycloak Adapter*.

Konfigurację podajemy w pliku `/WEB-INF/keycloak-saml.xml` w pakiecie WAR, a jako metodę autoryzacji podejmy `KEYCLOAK-SAML` w pliku `web.xml`.

Adapter jest osobnym plikiem do pobrania na stronie pobierania Keycloak, który należy rozpakować. Istnieje skrypt CLI, który pomoże ci zmodyfikować konfigurację serwera. Uruchom serwer i uruchom skrypt z katalogu `bin` serwera.

```
$ cd $WILDFLY_HOME
$ unzip keycloak-saml-wildfly-adapter-dist.zip
$ ./bin/jboss-cli.sh -c --file=bin/adapter-elytron-install-saml.cli
```

Skrypt doda rozszerzenie, podsystem i opcjonalną domenę zabezpieczeń, jak opisano poniżej.

```
<server xmlns="urn:jboss:domain:1.4">
    <extensions>
        <extension module="org.keycloak.keycloak-saml-adapter-subsystem"/>
          ...
    </extensions>
    <profile>
        <subsystem xmlns="urn:jboss:domain:keycloak-saml:1.1"/>
         ...
    </profile>
```

3.1.3. Adapter Tomcat SAML

Aby móc zabezpieczyć aplikacje WAR wdrożone w *Tomcat*, musisz zainstalować adapter adapter *Keycloak Tomcat SAML* w folderze Tomcat. Następnie musisz podać dodatkową konfigurację w każdym WAR. Przejdźmy przez te kroki.

Musisz rozpakować dystrybucję adaptera do katalogu `lib/` Tomcata. Umieszczenie bibliotek adaptera w katalogu `WEB-INF/lib` nie będzie działać!

```
$ cd $TOMCAT_HOME/lib
$ unzip keycloak-saml-tomcat-adapter-dist.zip
```

zabezpieczamy poprzez dodanie plików konfiguracji i edycji w pakiecie WAR.

Pierwszą rzeczą, którą musisz zrobić aby zabezpieczyć WAR, to utworzyć plik `META-INF/context.xml`. To jest plik konfiguracyjny specyficzny dla Tomcat i musi zdefiniować *Valve* dla Keycloak.

```
<Context path="/your-context-path">
    <Valve
className="org.keycloak.adapters.saml.tomcat.SamlAuthenticatorValve"/>
</Context>
```

Następnie musisz utworzyć plik konfiguracyjny adaptera `WEB-INF/keycloak-saml.xml`. Format tego pliku konfiguracyjnego opisano wcześniej.

Na koniec musisz określić konfigurację logowania i użyć standardowych zabezpieczeń serwletu, aby określić ograniczenia bazowe dla ról i adresów URL. Oto przykład:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
      version="3.0">
        <module-name>customer-portal</module-name>
```

```
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Customers</web-resource-name>
            <url-pattern>/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>user</role-name>
        </auth-constraint>
    </security-constraint>
    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>this is ignored currently</realm-name>
    </login-config>
    <security-role>
        <role-name>admin</role-name>
    </security-role>
    <security-role>
        <role-name>user</role-name>
    </security-role>
</web-app>
```

Jeśli plik `keycloak-saml.xml` nie ustawia jawnie `assertionConsumerServiceUrl`, adapter SAML będzie domyślnie nasłuchiwał asercji SAML w lokalizacji `/my-context-path/saml`.

3.1.4. Uzyskiwanie atrybutów potwierdzenia

Po udanym logowaniu SAML kod aplikacji może chcieć uzyskać wartości atrybutów przekazywane z potwierdzeniem (*ang. assertion*) SAML. `HttpServletRequest.getUserPrincipal()` zwraca obiekt, który można rzutować do klasy specyficznej dla Keycloak o nazwie `org.keycloak.adapters.saml.SamlPrincipal`. Ten obiekt pozwala spojrzeć na surowe potwierdzenie, a także ma wygodne funkcje do wyszukiwania wartości atrybutów.

```
package org.keycloak.adapters.saml;
public class SamlPrincipal implements Serializable, Principal {
    /**
     * Get full saml assertion
     *
     * @return
     */
    public AssertionType getAssertion() {
       ...
    }
    /**
     * Get SAML subject sent in assertion
     *
     * @return
     */
    public String getSamlSubject() {
        ...
    }
    /**
     * Subject nameID format
     *
     * @return
     */
    public String getNameIDFormat() {
        ...
    }
```

```
        @Override
        public String getName() {
            ...
        }
        /**
         * Convenience function that gets Attribute value by attribute name
         *
         * @param name
         * @return
         */
        public List<String> getAttributes(String name) {
            ...
        }
        /**
         * Convenience function that gets Attribute value by attribute friendly
name
         *
         * @param friendlyName
         * @return
         */
        public List<String> getFriendlyAttributes(String friendlyName) {
            ...
        }
        /**
         * Convenience function that gets first  value of an attribute by attribute
name
         *
         * @param name
         * @return
         */
        public String getAttribute(String name) {
            ...
        }
        /**
         * Convenience function that gets first  value of an attribute by attribute
name
         *
         *
         * @param friendlyName
         * @return
         */
        public String getFriendlyAttribute(String friendlyName) {
            ...
        }
        /**
         * Get set of all assertion attribute names
         *
         * @return
         */
        public Set<String> getAttributeNames() {
            ...
        }
        /**
         * Get set of all assertion friendly attribute names
         *
         * @return
         */
        public Set<String> getFriendlyNames() {
```

```
            ...
        }
    }
```

3.1.5. Obsługa błędów

Keycloak ma kilka funkcji obsługi błędów dla adapterów klienckich opartych na serwletach. Gdy wystąpi błąd podczas uwierzytelniania, Keycloak wywoła `HttpServletResponse.sendError()`. Możesz ustawić stronę błędu w pliku `web.xml`, aby obsłużyć błąd w dowolny sposób. Keycloak może zgłaszać błędy 400, 401, 403 i 500.

```
    <error-page>
        <error-code>403</error-code>
        <location>/ErrorHandler</location>
    </error-page>
```

Keycloak ustawia również atrybut `org.keycloak.adapters.spi.AuthenticationError`, który można pobrać z `HttpServletRequest`. Wartość należy zrzutować jako `SamlAuthenticationError`. Ta klasa może dokładnie powiedzieć, co się stało. Jeśli ten atrybut nie jest ustawiony, adapter nie ponosi odpowiedzialności za kod błędu.

```
    public class SamlAuthenticationError implements AuthenticationError {
        public static enum Reason {
            EXTRACTION_FAILURE,
            INVALID_SIGNATURE,
            ERROR_STATUS
        }
        public Reason getReason() {
            return reason;
        }
        public StatusResponseType getStatus() {
            return status;
        }
    }
```

3.1.6. Rozwiązywanie problemów

Najlepszym sposobem rozwiązywania problemów jest włączenie debugowania SAML zarówno w kliencie, jak i na serwerze Keycloak. Za pomocą podsystemu logowania ustaw poziom dziennika na DEBUG dla pakietów `org.keycloak.saml`. Pozwali to zobaczyć żądania SAML i dokumenty odpowiedzi wysyłane do i z serwera.

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.