

SOA Lab 11

Security w Java EE - standard JASS

API bezpieczeństwa w języku Java

Ostatnim elementem omawiania środowiska JavaEE jest kwestia bezpieczeństwa aplikacji stanowiąca kluczowy element każdej aplikacji biznesowej. Trzeba zastanowić się, kto będzie miał dostęp do aplikacji i jakie operacje będzie mógł w niej wykonywać. Specyfikacja Java EE definiuje dla komponentów webowych i EJB (Enterprise JavaBeans) prosty model bezpieczeństwa bazujący na rolach – standard JASS.

Usługi bezpieczeństwa w Javie EE zapewniają elastyczny i łatwy w konfiguracji mechanizm bezpieczeństwa zapewniający uwierzytelnienie użytkowników i autoryzację dostępu do funkcji powiązanych z nimi danych. Aby lepiej zrozumieć kwestie związane z bezpieczeństwem przypomnienie najważniejszych pojęć związanych z kwestiami bezpieczeństwa.

Uwierzytelnienie (autentykacja) to proces weryfikacji, kto, czy raczej co, naprawdę korzysta z aplikacji, niezależnie od tego, czy jest to EJB, czy serwet. Uwierzytelnienie najczęściej wykonuje moduł Login znajdujący się w aplikacji webowej lub uruchamiany samodzielnie.

Autoryzacja to proces, w którym sprawdza się, czy użytkownik ma prawo (uprawnienie) dostępu do określonych zasobów. Autoryzacja wymaga więc wcześniejszego uwierzytelnienia, bo trudno byłoby sprawdzić uprawnienia, gdy nie wie się, z kim ma się do czynienia.

W Javie EE kontenery komponentów odpowiadają za zapewnienie bezpieczeństwa w aplikacji. Kontener udostępnia dwa podstawowe typy bezpieczeństwa — **deklaratywny i programowy**.

Bezpieczeństwo deklaratywne wyraża wymagania bezpieczeństwa komponentów aplikacji, zastosowane są w nim deskryptory wdrożenia. Ponieważ dane deskryptora wdrożenia znajdują się w osobnym pliku, można je zmieniać bez potrzeby ponownej kompilacji lub znajomości kodu źródłowego.

Komponenty Enterprise JavaBeans stosują deskryptor wdrożenia EJB, który musi nosić nazwę ejb-jar.xml i znajdować się w folderze META-INF pliku JAR z EJB.

Komponenty webowe korzystają z deskryptora wdrożenia nazwanego web.xml znajdującego się w folderze WEB-INF.

Od Javy EE 6 bezpieczeństwo deklaratywne można również definiować za pomocą adnotacji, podobnie jak ma to miejsce dla innych kluczowych API (EJB, usług sieciowych i tym podobnych). Adnotacje znajdują się w pliku klasy i w momencie wdrożenia są zamieniane przez serwer aplikacji na odpowiednie wpisy bezpieczeństwa.

Bezpieczeństwo programowe znajduje się w aplikacji i służy do podejmowania decyzji związanych z bezpieczeństwem. Używa się go, gdy deklaracyjny model bezpieczeństwa nie jest w stanie odpowiednio dobrze opisać modelu bezpieczeństwa aplikacji. API bezpieczeństwa Javy EE umożliwia programiście sprawdzenie, czy aktualny użytkownik ma dostęp do konkretnej roli, po wykonaniu następujących wywołań funkcji:

- isUserInRole() dla serwletów i JSF (znajduje się w javax.servlet.http.HttpServletRequest),
- isCallerInRole() dla EJB (znajduje się w javax.ejb.SessionContext).

Dodatkowo można skorzystać z wywołań API dotyczących tożsamości użytkownika:

- getUserPrincipal() dla serwletów i JSF (znajduje się w javax.servlet.http.HttpServletRequest),
- getCallerPrincipal() dla EJB (znajduje się w javax.ejb.SessionContext).

Użycie przedstawionych API umożliwia tworzenie modeli autoryzacji o dowolnie dużej złożoności.

----- Podsystem bezpieczeństwa w WILDFLY -----

Bezpieczeństwo WildFly stanowi rozszerzenie serwera aplikacji i jest dołączane domyślnie zarówno w konfiguracji samodzielnej, jak i domenowej.

```
<extension module="org.jboss.as.security"/>
```

Poniżej znajduje się fragment dotyczący podsystemu bezpieczeństwa, pochodzący z pliku konfiguracyjnego serwera (standalone.xml). Zawiera moduł logowania RealmUsersRoles

```
<subsystem xmlns="urn:jboss:domain:security:1.1">

  <security-domains>
    <security-domain name="other" cache-type="default">
      <authentication>
        <login-module code="Remoting" flag="optional">
          <module-option name="password-stacking" value="useFirstPass"/>
        </login-module>
        <login-module code="RealmUsersRoles" flag="required">
          <module-option name="usersProperties" value="${jboss.server.config.dir}/applicationusers.properties"/>
          <module-option name="rolesProperties" value="${jboss.server.config.dir}/applicationroles.properties"/>
          <module-option name="realm" value="ApplicationRealm"/>
          <module-option name="password-stacking" value="useFirstPass"/>
        </login-module>
      </authentication>
    </security-domain>
    . . .
  </security-domains>
```

```
</subsystem>
```

```
<security-domain name="customSecurity" cache-type="default">
  <authentication>
    <login-module code="com.CustomModule" flag="required"/>
  </authentication>
  <authorization>
    <policy-module code="PermitAll" flag="required"/>
  </authorization>
</security-domain>
```

Konfiguracja jest krótka i dosyć intensywnie korzysta z wartości domyślnych, szczególnie w przypadku struktur wysokopoziomowych, takich jak obszar zarządzania bezpieczeństwem. Definiując swoje opcje zarządzania bezpieczeństwem, można na przykład użyć własnych implementacji menedżerów uwierzytelniania i autoryzacji. Ponieważ jednak w większości sytuacji tego rodzaju zastąpienia nie są konieczne, skupimy się na elemencie security-domain stanowiącym główny aspekt bezpieczeństwa w JBoss.

Domenę bezpieczeństwa można traktować jak bramkę wstępu. Zanim żądanie przekroczy granicę serwera JBoss, domena bezpieczeństwa wykonuje wszystkie sprawdzenia dotyczące **uwierzytelnienia** i **autoryzacji**, by upewnić się, że żądanie ma prawo przejść dalej.

Domeny bezpieczeństwa określa się na etapie uruchomienia serwera. Są powiązane z drzewem JNDI przy użyciu klucza java:/jaas/. Wewnątrz domeny bezpieczeństwa konfiguruje się własne moduły uwierzytelnienia, by można było w łatwy sposób zmienić dostawcę uwierzytelnienia przez prostą podmianę dotyczącego go modułu login-module.

Domyślnie system zawiera kilka różnych implementacji modułów logowania. Oczywiście, nie mamy tu wystarczającej ilości miejsca, by opisać każdy z nich, więc skupimy się na dwóch najpopularniejszych:

- module logowania RealmUsersRoles służącym do prostego uwierzytelniania bazującego na pliku,
- module logowania Database sprawdzającym dane użytkownika w relacyjnej bazie danych.

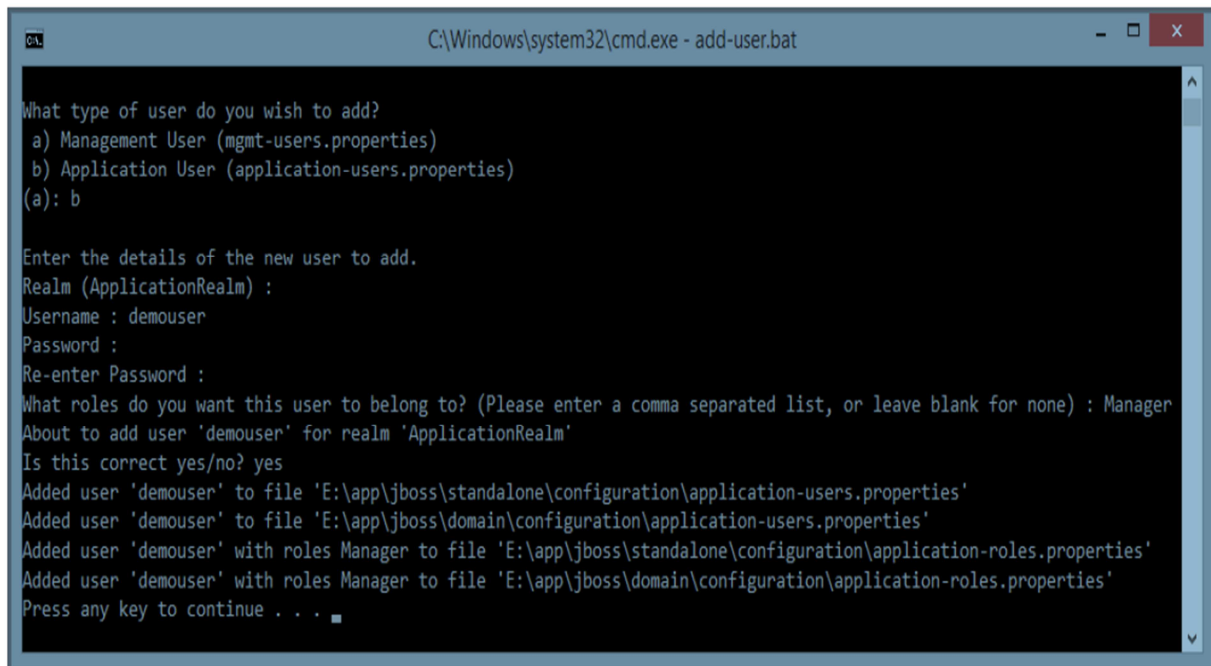
Dodatkowe informacje na temat modułów logowania zawiera dokumentacja JBoss AS 7.1 dostępna pod adresem <https://docs.jboss.org/author/display/WFLY10/Security+subsystem+configuration>.

Konfiguracja pierwszego modułu logowania

Zdefiniujmy w jaki sposób zabezpieczyć aplikację, używając opisanej wcześniej domeny bezpieczeństwa RealmUsersRoles. Moduł logowania RealmUsersRoles bazuje na dwóch plikach:

- *application-users.properties* — zawiera listę nazw użytkowników i haseł,
- *application-roles.properties* — zawiera odwzorowanie między użytkownikami i rolami.

Pliki te znajdują się w folderze z konfiguracją serwera aplikacji i są aktualizowane za każdym razem, gdy nowy użytkownik zostaje dodany przy użyciu skryptu *add-user.sh* (*add-user.bat*). Utwórzmy nowego użytkownika aplikacji o nazwie demouser, który należy do roli Manager. Wszystko to zostało przedstawione na poniższym zrzucie.



```
C:\Windows\system32\cmd.exe - add-user.bat

What type of user do you wish to add?
a) Management User (mgmt-users.properties)
b) Application User (application-users.properties)
(a): b

Enter the details of the new user to add.
Realm (ApplicationRealm) :
Username : demouser
Password :
Re-enter Password :
What roles do you want this user to belong to? (Please enter a comma separated list, or leave blank for none) : Manager
About to add user 'demouser' for realm 'ApplicationRealm'
Is this correct yes/no? yes
Added user 'demouser' to file 'E:\app\jboss\standalone\configuration\application-users.properties'
Added user 'demouser' to file 'E:\app\jboss\domain\configuration\application-users.properties'
Added user 'demouser' with roles Manager to file 'E:\app\jboss\standalone\configuration\application-roles.properties'
Added user 'demouser' with roles Manager to file 'E:\app\jboss\domain\configuration\application-roles.properties'
Press any key to continue . . .
```

Po dodaniu użytkownika w pliku *application-users.properties* pojawi się nazwa użytkownika i i zaszyfrowane skrótem MD5 hasło:

```
demouser=7504700805d5d619e4bf0b3b453b5a41
```

W pliku *application-roles.properties* pojawią się role przypisane użytkownikowi demouser po zalogowaniu się:

```
demouser=Manager
```

Użycie modułu logowania w przykładowej aplikacji

Najpierw przeanalizujemy uwierzytelnianie bazujące na mechanizmie BASIC, a następnie pokażemy bardziej zaawansowany przykład, w którym użyjemy formularza.

Uwierzytelnianie bazujące na mechanizmie BASIC to najprostszy sposób poproszenia o nazwę użytkownika i hasło przy wykonywaniu żądań w przeglądarce internetowej.

Metoda ta działa na zasadzie przekazywania zakodowanych danych użytkownika. Tekst w postaci zakodowanej (Base64) jest przesyłany do serwera, który go dekoduje i odczytuje oddzielone dwukropkiem nazwę użytkownika i hasło.

Włączenie uwierzytelnienia wymaga zdefiniowania elementu `security-constraints` w pliku konfiguracyjnym aplikacji webowej (*web.xml*). Istotne fragmenty zostały pogrubione.

```
<web-app>
.....
<security-constraint>

  <web-resource-collection>
    <web-resource-name>HtmlAuth</web-resource-name>
    <description>zasady bezpieczeństwa aplikacji</description>
    <url-pattern>/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
```

```

<role-name>Manager</role-name>
</auth-constraint>
</security-constraint>
<login-config>
<auth-method>BASIC</auth-method>
<realm-name>file</realm-name>
</login-config>
<security-role>
<role-name>Manager</role-name>
</security-role>
</web-app>

```

Przedstawiona konfiguracja spowoduje dodanie ograniczenia bezpieczeństwa do każdego serwletu lub JSP aplikacji webowej. Dostęp do aplikacji będą miały tylko osoby, które się uwierzytelnią i dodatkowo należą do roli Manager. Wszystkie moduły logowania przedstawione we wcześniejszych punktach definiują taką rolę, więc można użyć dowolnego modułu logowania spełniającego oczekiwania.

Druga modyfikacja konfiguracji dotyczy deskryptora wdrożenia webowego JBoss, czyli pliku *WEB-INF/jboss-web.xml*. W tym miejscu trzeba wskazać domenę bezpieczeństwa używaną do uwierzytelniania użytkowników.

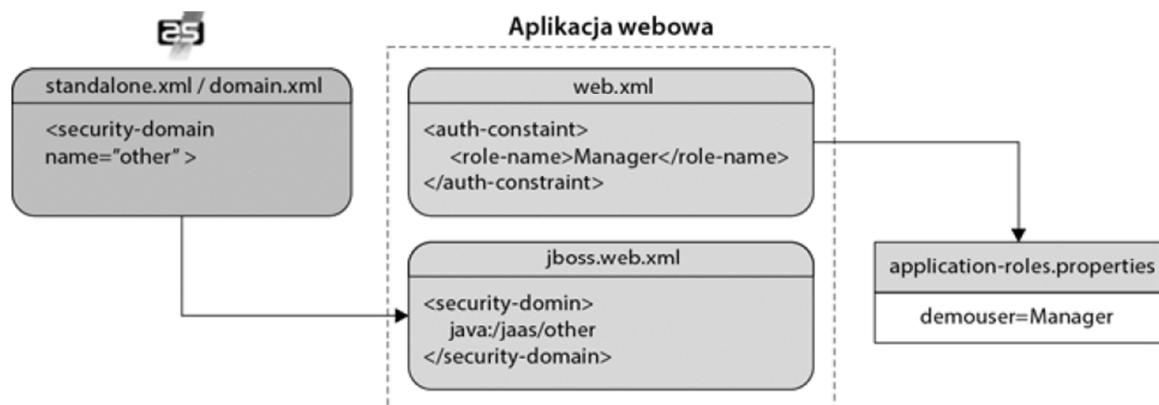
Ponieważ korzystamy z RealmUsersRoles stanowiącego część innego, wbudowanego modułu logowania, jako kontekst musimy wskazać *java:/jaas/other*.

```

<jboss-web>
<security-domain>java:/jaas/other</security-domain>
</jboss-web>

```

Poniższy diagram to ilustracja całej sekwencji konfiguracji używanej dla modułu logowania bazującego na plikach.



Po wdrożeniu aplikacji powinno pojawić się okno z zapytaniem o dane niezbędne do uwierzytelnienia.

Zalogowanie się jako użytkownik demouser z prawidłowym hasłem zapewni dostęp do aplikacji z rolą Manager.

Przełączenie na bezpieczeństwo bazujące na formularzu

Uwierzytelnianie bazujące na formularzu umożliwia programistom dostosowanie interfejsu logowania do własnych potrzeb (na przykład do wyglądu zgodnego z standardami firmy).

Konfiguracja w aplikacji wymaga jedynie modyfikacji fragmentu login-config zawartego w pliku *web.xml*.

Wskazuje się w nim stronę logowania (*login.jsf*) oraz stronę błędu (*error.jsf*) wyświetlaną w przypadku niepowodzenia w trakcie logowania. Oto cały fragment.

```
<login-config>
<auth-method>FORM</auth-method>
<realm-name>file</realm-name>
<form-login-config>
<form-login-page>/login.jsf</form-login-page>
<form-error-page>/error.jsf</form-error-page>
</form-login-config>
</login-config>
```

Formularz logowania musi zawierać pola umożliwiające wpisanie nazwy użytkownika i hasła. Pola te muszą nosić nazwy (odpowiednio) *j_username* i *j_password*. Formularz powinien postać wynik do nazwy logicznej *j_security_check*. Poniżej znajduje się przykład prostej strony *login.jsf* zawierającej wszystkie wymagane elementy.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
· "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html">
<body>
<h1>Proszę się zalogować</h1>

<form method="post" action="j_security_check" name="loginForm">
<h:panelGrid columns="2">
<h:outputLabel id="userNameLabel" for="j_username" value=
· "Nazwa użytkownika:"/>
```

```

<h:inputText id="j_username" />
<h:outputLabel id="passwordLabel" for="j_password"
· value="Hasło:"/>
<h:inputSecret id="j_password" />
<div/>
<h:panelGroup>
<h:commandButton type="submit" value="Zaloguj się"/>
<h:commandButton type="reset" value="Wyczyść"/>
</h:panelGroup>
</h:panelGrid>
</form>
</body>
</html>

```

Nie omawiam strony błędu, ponieważ stanowi ona prostą stronę informacyjną wskazującą użytkownikowi, iż popełnił błąd w trakcie logowania. Strona logowania będzie się wyświetlała, aż użytkownik wpisze poprawną nazwę użytkownika i hasło umożliwiające wyświetlenie pozostałych treści aplikacji webowej.



Tworzenie modułu logowania wykorzystującego bazę danych

Moduł logowania UserRoles to dobry punkt startowy, by rozpocząć naukę wszystkich elementów niezbędnych do zapewnienia bezpieczeństwa aplikacji webowych. W produkcyjnych wersjach systemów stosuje się jednak bardziej zaawansowane rozwiązania, na przykład moduł logowania Database. Domena bezpieczeństwa wykorzystuje tę samą logikę, co wcześniejsze przykłady, ale przechowuje dane o użytkownikach w bazie danych.

```

<security-domain name="mysqlDomain" cache-type="default">
<authentication>
<login-module code="Database" flag="required">

<module-option name="dsJndiName" value="
· java:jboss/datasources/jbossdevelopment"/>
<module-option name="principalsQuery" value="select passwd
· from USERS where login=?"/>
<module-option name="rolesQuery" value="select role, 'Roles'
· from USER_ROLES where login=?"/>
</login-module>
</authentication>
</security-domain>

```

Aby przedstawiona konfiguracja zadziałała prawidłowo, trzeba jeszcze utworzyć odpowiednie tabele i wypełnić je danymi.

```

CREATE TABLE USERS(login VARCHAR(64) PRIMARY KEY, passwd VARCHAR(64));
CREATE TABLE USER_ROLES(login VARCHAR(64), role VARCHAR(32));
INSERT into USERS values('admin', 'admin');
INSERT into USER_ROLES values('admin', 'Manager');

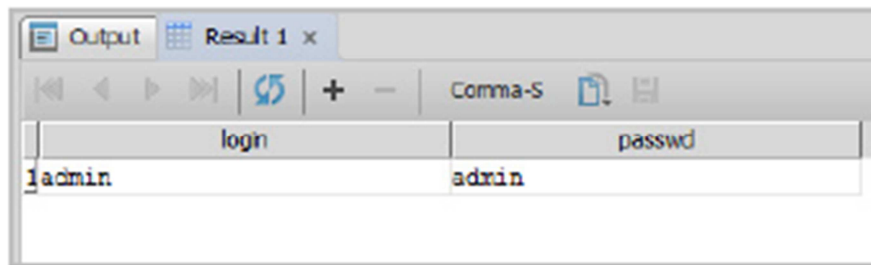
```

Ponownie użytkownik admin zostanie odwzorowany na rolę Manager. Jedyny problem z przedstawioną

konfiguracją jest taki, iż w bazie danych hasła są zapisane w postaci otwartego tekstu. Przed umieszczeniem modułu w systemie produkcyjnym warto rozważyć dodanie dodatkowych zabezpieczeń.

Szyfrowanie haseł

Przechowywanie w bazie haseł w postaci jawnego tekstu nie jest uważane za dobrą praktykę. W zasadzie baza danych posiada znacznie więcej potencjalnych luk związanych z bezpieczeństwem niż system plików. Wyobraźmy sobie, iż administrator bazy danych dodał publiczne synonimy do niektórych tabel, ale zapomniał, że część z nich zawiera wrażliwe informacje, takie jak hasła! Trzeba mieć pewność, iż potencjalny atakujący nigdy nie będzie mógł wykonać następującego zapytania.



login	passwd
admin	admin

Na szczęście, zabezpieczenie haseł jest stosunkowo proste. Wystarczy do modułu logowania dodać kilka opcji, by hasła były przechowywane w stanie zaszyfrowanym przy wcześniejszym użyciu funkcji skrótu. Do utworzonego wcześniej modułu logowania dodajemy pogrubione wpisy.

```
<login-module code="Database" flag="required">
<module-option name="dsJndiName" value="
java:jboss/datasources/jbossdevelopment"/>
<module-option name="principalsQuery" value="select passwd from USERS
· where login=?"/>
```

Wskazaliśmy, iż hasła będą tworzone z użyciem funkcji skrótu, w tym przypadku za pomocą algorytmu MD5. Można zastosować dowolny algorytm skrótu zapewniany przez dostawcę JCA, na przykład SHA.

By zapewnić możliwość sprawdzenia całego przykładu, prezentuje kod małej aplikacji, w której klasy `java.security.MessageDigest` i `org.jboss.security.Base64Utils` wykorzystane zostały do wygenerowania skrótu hasła w postaci Base64 i wstawienia go do bazy danych.

```
public class Hash {
public static void main(String[] args) throws Exception{
String password = args[0];
MessageDigest md = null;
md = MessageDigest.getInstance("MD5");
byte[] passwordBytes = password.getBytes();
byte[] hash = md.digest(passwordBytes);
String passwordHash = org.jboss.security.Base64Utils.toBase64(hash);
System.out.println("skrót hasła: "+passwordHash);
}
```

Uruchomienie programu z parametrem `admin` spowoduje wygenerowanie skrótu `X8oyfUbUbfqE9IWvAW1/3`. Trzeba jeszcze zaktualizować wpis w bazie danych, by zawierał uaktualnioną postać hasła.

```
UPDATE USERS SET PASSWD = 'X8oyfUbUbfqE9IWvAW1/3' WHERE LOGIN = 'admin';
```

Aktualizację można wykonać w dowolnym kliencie SQL.

Użycie modułu logowania Database w aplikacji

Po zakończeniu konfiguracji modułu logowania nie wolno zapomnieć o użyciu właściwego wpisu w pliku deskryptora webowego dla JBoss (*WEB-INF/jboss-web.xml*).

```
<jboss-web>
```

```
<security-domain>java:/jaas/mysqldomain</security-domain>
```

```
</jboss-web>
```

Zabezpieczenie komponentów EJB

Zabezpieczanie aplikacji przez dodanie formularza logowania to chyba najpopularniejsza metoda sprawdzania tożsamości w aplikacjach biznesowych. Trzeba jednak pamiętać, że protokół HTTP nie jest jedynym protokołem dającym dostęp do aplikacji. Komponenty EJB mogą być dostępne dla zdalnych klientów przy użyciu protokołu RMI-IIOP.

W takiej sytuacji trzeba zastosować dodatkowe reguły bezpieczeństwa ograniczające dostęp do komponentów EJB, które przecież odpowiadają za warstwę logiki biznesowej.

Jak działają zabezpieczenia na poziomie komponentów EJB?

Uwierzytelnienie trzeba przeprowadzić przed wywołaniem jakiejkolwiek metody EJB.

Autoryzacja następuje w momencie rozpoczynania każdej z metod EJB.

To jeden z obszarów, które zostały znacząco usprawnione w Javie EE 6 wraz z wprowadzeniem adnotacji, bo można bardzo łatwo określić podstawowe testy bezpieczeństwa. Istnieje pięć adnotacji odpowiedzialnych za bezpieczeństwo.

- `@org.jboss.ejb3.annotation.SecurityDomain` określa domenę bezpieczeństwa związaną z klasą lub metodą.
- `@javax.annotation.security.RolesAllowed` definiuje listę ról dopuszczonych do korzystania z metod aplikacji EJB.
- `@javax.annotation.security.RunAs` przypisuje rolę dynamicznie do aplikacji EJB w trakcie wywoływania metody; umożliwia to tymczasową zmianę uprawnień w celu wykonania niezbędnych metod.
- `@javax.annotation.security.PermitAll` wskazuje, że aplikację EJB może wywołać dowolny klient; celem tej adnotacji jest rozszerzenie dostępu do metod w sytuacji, gdy nie wiadomo dokładnie, która rola będzie korzystała z aplikacji EJB (na przykład część modułów realizuje zewnętrzny dostawca i nie wiadomo dokładnie, jakie role zdefiniował).
- `@javax.annotation.security.DenyAll` wskazuje, że aplikacji EJB nie może wywoływać żaden zewnętrzny klient; pozostałe aspekty działania odpowiadają `PermitAll`.

Oto krótki przykład zabezpieczenia stanowego ziarna sesyjnego

```
@RolesAllowed("Manager")
```

```
@SecurityDomain("mysqldomain")
```

```
@Stateful
```

```
@Remote(TheatreBooker.class)
```

```
public class TheatreBookerBean implements TheatreBooker {  
}
```

Adnotacje można również stosować na poziomie metod.

Jeśli chcemy zabezpieczyć jedynie metodę bookSeat klasy TheatreBookerBean, adnotacje mogą mieć postać:

@RolesAllowed("Manager")

@SecurityDomain("mysqldomain")

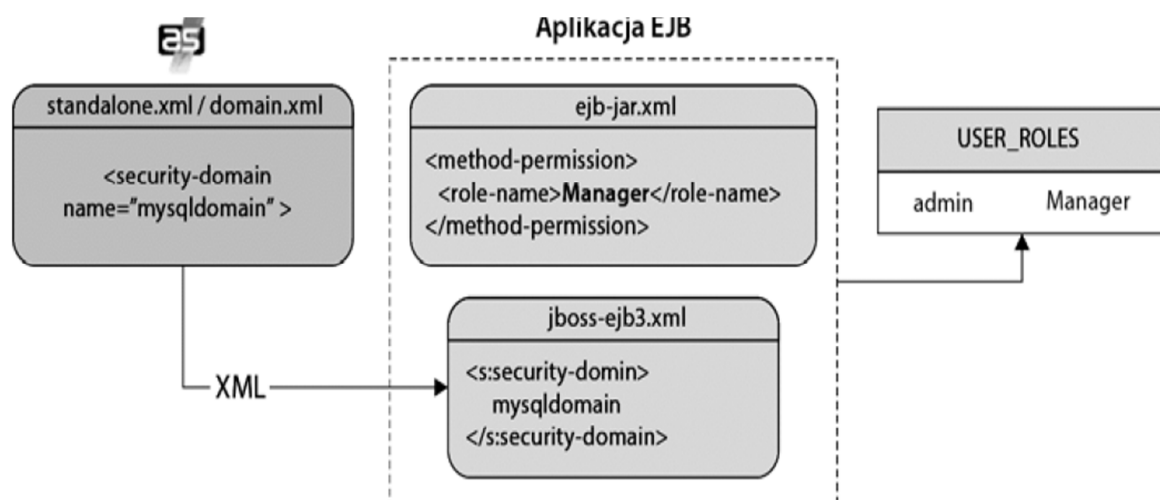
```
public String bookSeat(int seatId) throws SeatBookedException {  
}
```

Co zrobić, jeśli nie chcemy używać adnotacji do wskazywania ról? Jeśli pewna rola bezpieczeństwa pojawia się we wszystkich aplikacjach EJB, zapewne łatwiej zdefiniować ją w konfiguracji XML, zamiast oznaczać wszystkie komponenty EJB adnotacjami. Używając tego podejścia, trzeba przede wszystkim zadeklarować ograniczenia bezpieczeństwa w ogólnym pliku *META-INF/ejb-jar.xml*.

```
<method-permission>  
<role-name>Manager</role-name>  
<method>  
<ejb-name>*</ejb-name>  
<method-name>*</method-name>  
</method>  
</method-permission>
```

Następnie w pliku konfiguracyjnym *META-INF/jboss-ejb3.xml* należy dodać referencję do domeny bezpieczeństwa.

```
<jboss:ejb-jar>  
<assembly-descriptor>  
<s:security>  
<ejb-name>*</ejb-name>  
<s:security-domain>mysqldomain</s:security-domain>  
</s:security>  
</assembly-descriptor>  
</jboss:ejb-jar>
```



Zadanie do realizacji.

Wykorzystując aplikacje do obsługi biblioteki (lab6_EJB – zadanie 4) rozszerzyć ją o możliwość logowania oraz dostępu do poszczególnych funkcjonalności dla wybranych użytkowników.