# Wykład 2
# Inżynieria Oprogramowania

## Bezpieczeństwo(1)
## Wprowadzenie do mechanizmów bezpieczeństwa w programach Javy EE

wg „The Java  EE 5 Tutorial”

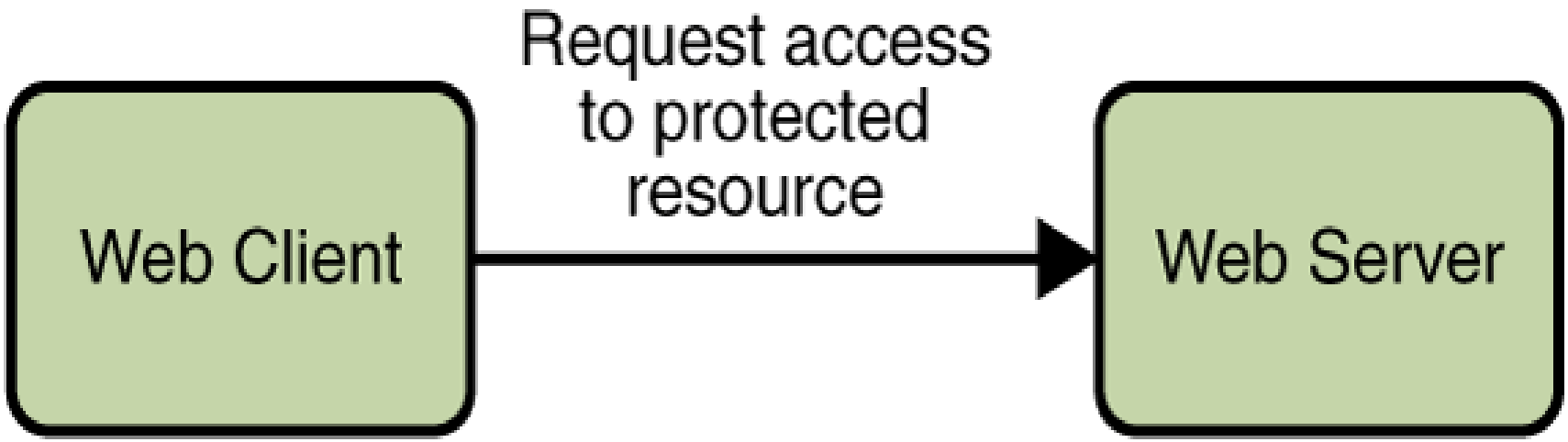
Autor: Zofia Kruczkiewicz

# Struktura wykładu

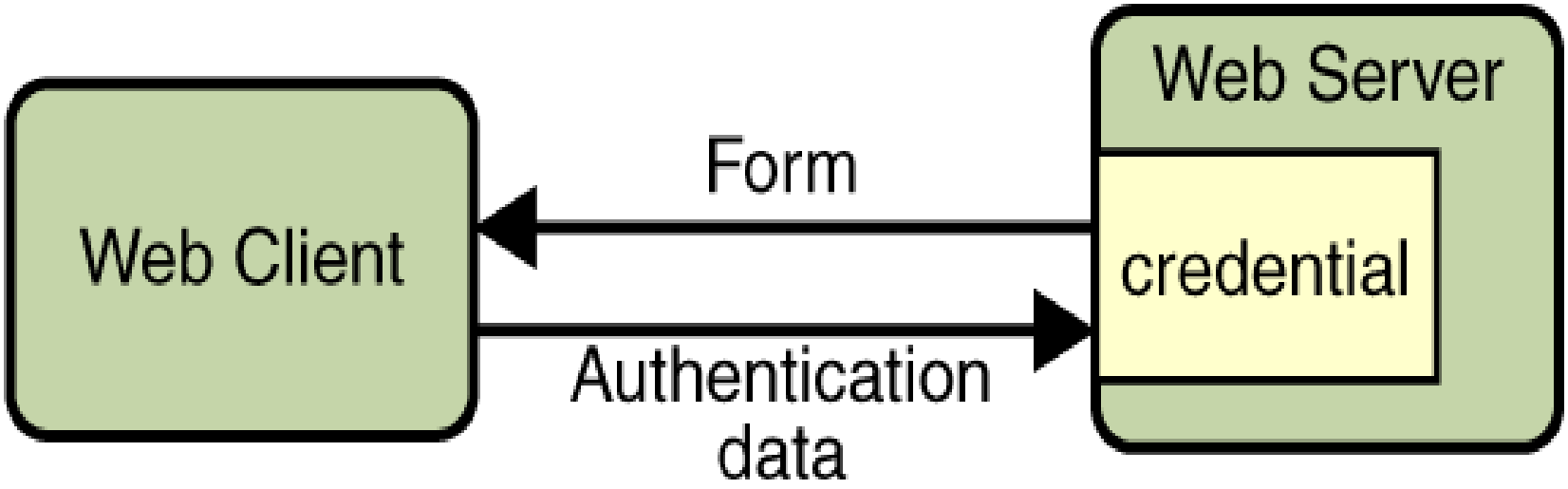
1. **Przykład przetwarzania strony**
2. **Cechy zabezpieczeń**
3. **Pojęcia systemu zabezpieczeń**
4. **Java EE - Poziomy implementacji mechanizmów bezpieczeństwa**
5. **Poziomy zabezpieczeń**
6. **Rodzaje mechanizmów bezpieczeństwa w kontenerach**
7. **Rodzaje mechanizmów bezpieczeństwa w serwerze aplikacji**
8. **Bazy użytkowników i grup, Użytkownik, Grupa, Rola**
9. **Implementacja mechanizmów bezpieczeństwa w platformie Java SE**

# 1. Przykład przetwarzania strony

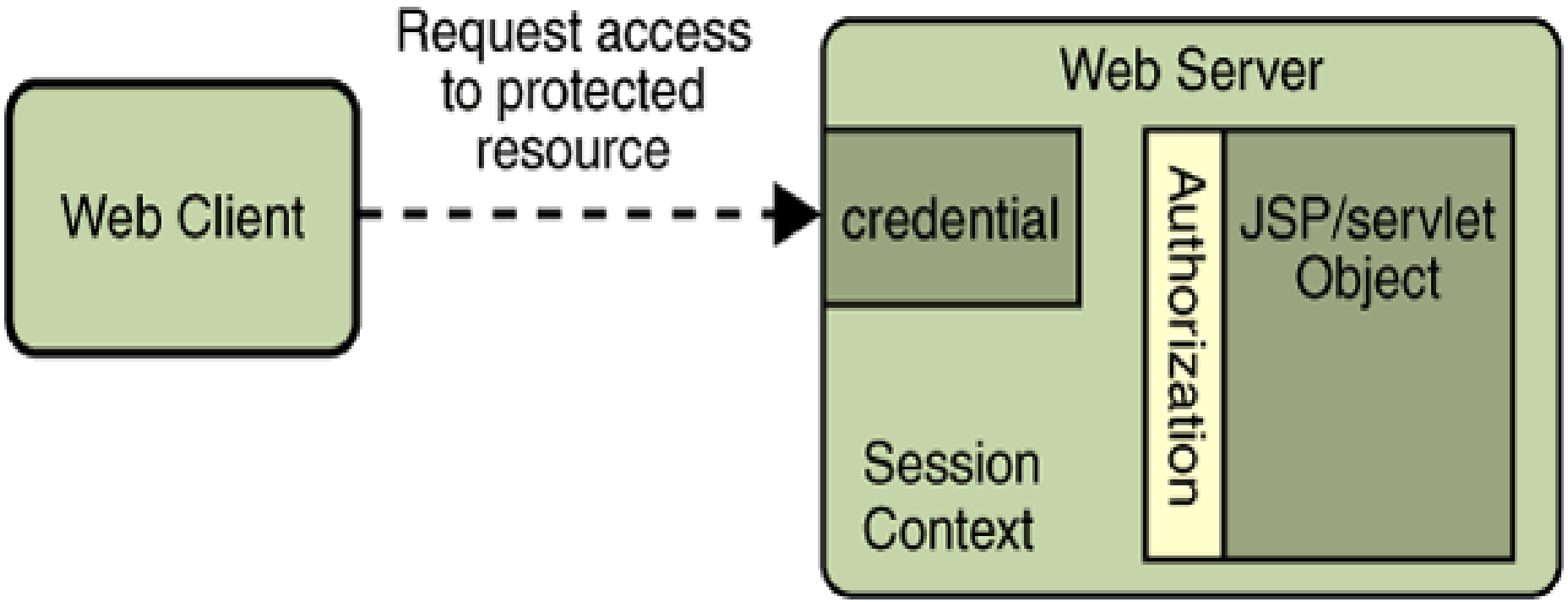
- Krok 1: Żądanie inicjujące



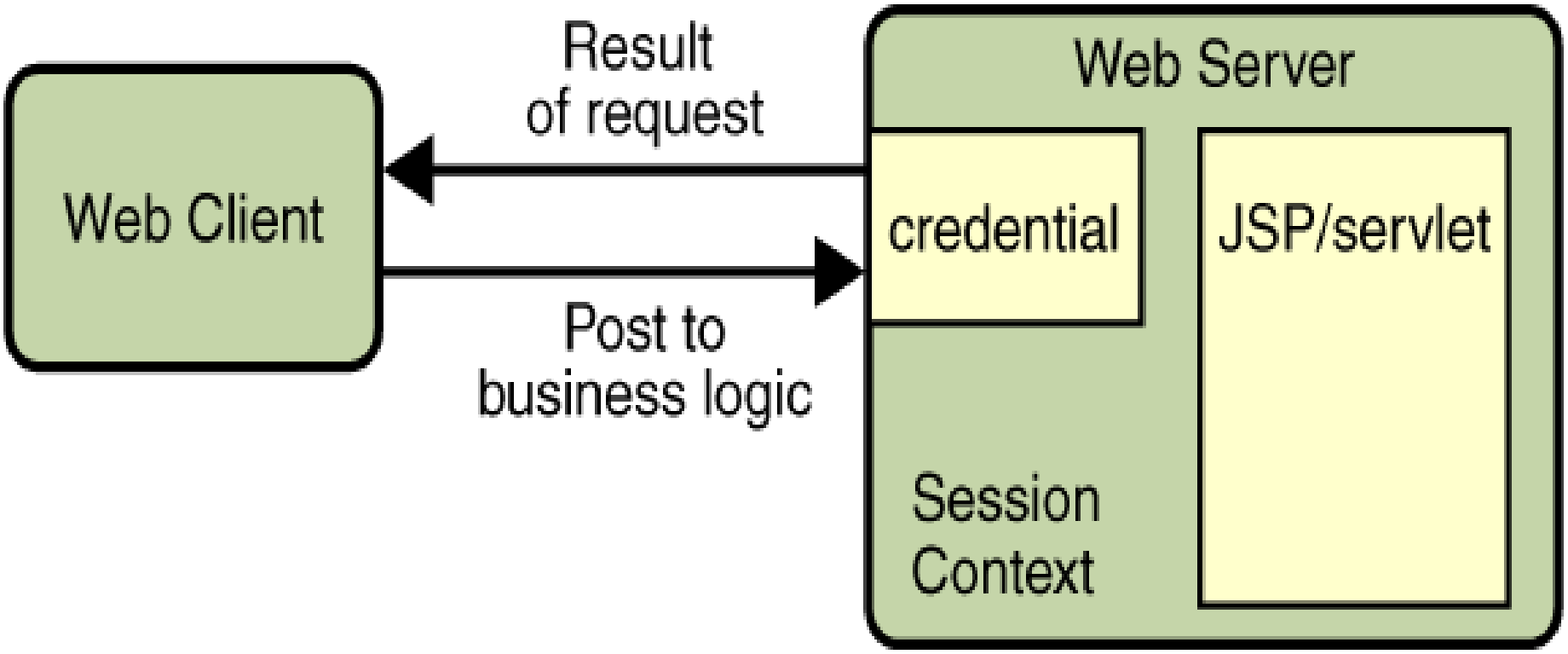
# Krok 2: Uwierzytelnienie inicjujące (login i hasło)



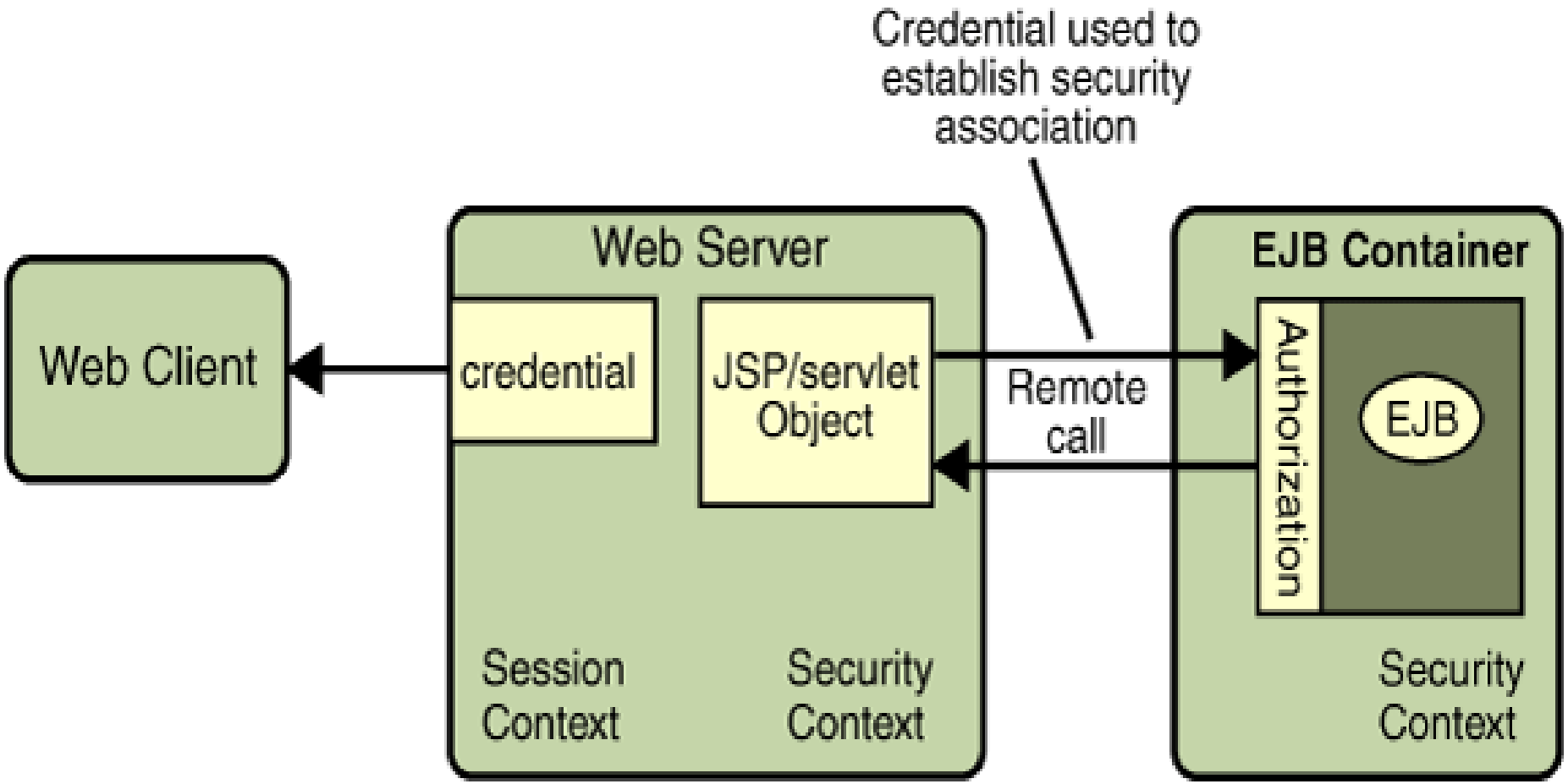
# Krok 3: Autoryzacja URL – serwer www określa rolę zalogowanego użytkownika i może kontrolować dostęp do konkretnych zasobów



# Krok 4:  Zakończenie przetwarzania żądania



# Krok 5:  Wykonanie funkcji biznesowej



# 2. Cechy zabezpieczeń

**Prawidłowo zastosowane mechanizmy zabezpieczeń:**

- Uniemożliwiają nieautoryzowany dostęp do funkcji aplikacji oraz danych osobistych i biznesowych
- Utrzymują odpowiedzialność użytkowników systemu za wykonywane operacje
- Chronią system przed niezaplanowanymi przerwami w pracy – podnoszą jakość usług systemu

**W idealnym przypadku mamy:**

- Łatwe administrowanie systemem
- Przezroczysty system dla użytkowników
- W pełni współpracujące części aplikacji w ramach całego systemu

# 3. Pojęcia systemu zabezpieczeń

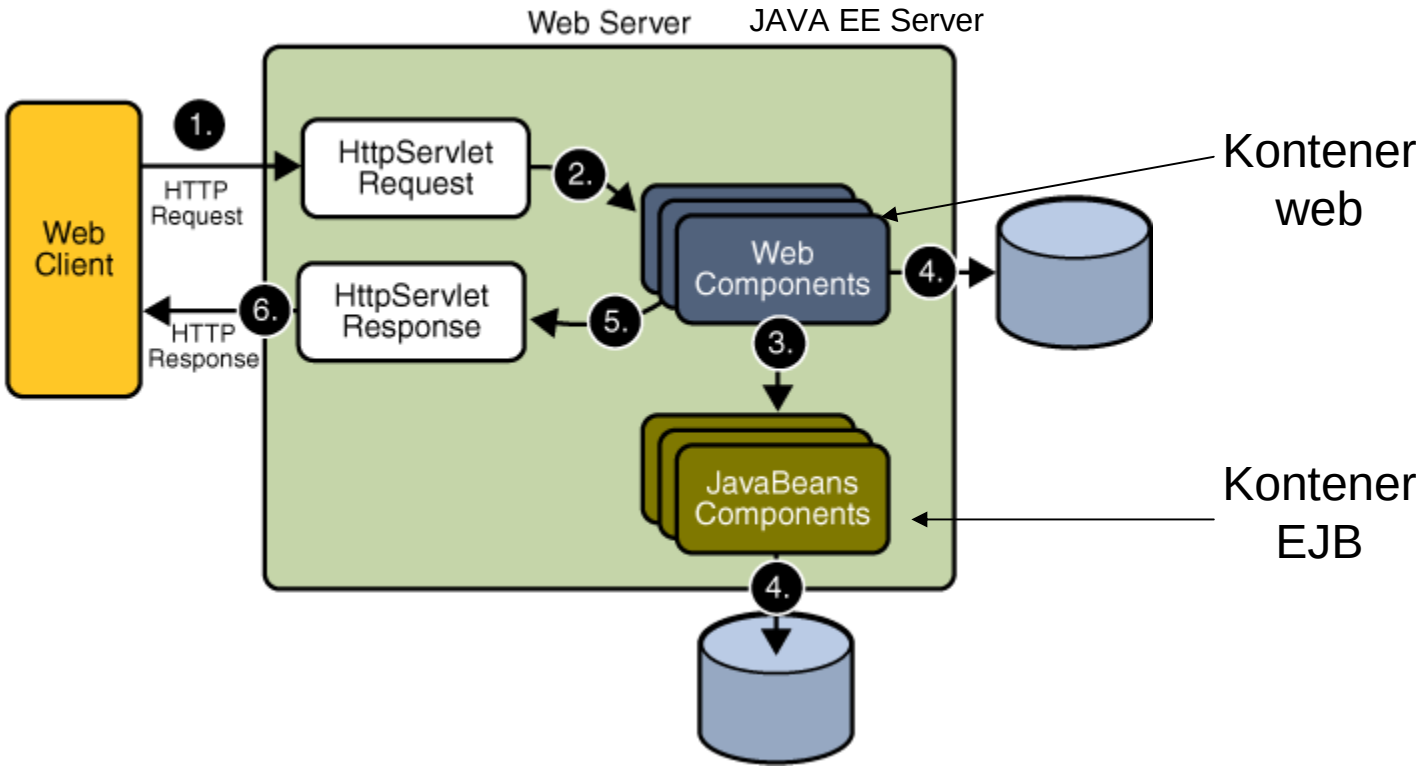
- **Uwierzytelnianie (Authentication)**: Uwierzytelnianie pozwalające komunikującym się encjom (np. klient i serwer)) na rozpoznanie się  w celu wykonania autoryzowanych funkcji
- **Autoryzacja lub kontrola dostępu (Authorization, or access control):** oznacza związki między zasobami systemu a zbiorem użytkowników lub programów w celu zachowania poprawnego działania systemu przez kontrolę dostępu do poszczególnych funkcji systemu.

  **Autoryzacja** opiera się na **identyfikacji** i **uwierzytelnianiu**.

  **Identyfikacja** jest procesem umożliwiającym rozpoznanie encji przez system

  **Uwierzytelnianie** jest  procesem umożliwiającym weryfikację użytkownika, urządzenia lub encji w systemie komputerowym,  zazwyczaj jako początkowy warunek umożliwiający dostęp do zasobów systemu
- **Integralność danych (Data integrity)**: zabezpieczenie przed niekontrolowanych dostępem do danych (np. dane otrzymane w systemie mogą być zmieniane tylko przez autoryzowanych użytkowników dzięki zastosowaniu podpisu cyfrowego)
- **Poufność danych (Confidentiality** or **Data Privacy)**: pewność, że dane są udostępnione tylko użytkownikom o autoryzowanym dostępie. Dane są szyfrowane podczas transmisji.
- **Zapobieganie unikaniu odpowiedzialności (Non-repudiation)**: Usługa, która stanowi dowód pochodzenia i integralność danych (oznacza pewność, że dane zostały przesłane przez stronę, która nie może później wyprzeć się autorstwa tych danych oraz odbiorca nie może wyprzeć się, że odebrał te dane). Usługa ta jest zapewniana za pomocą podpisu cyfrowego danych, mechanizmu znaczników potwierdzania nadawania i odbioru danych oraz znaczników czasu zawierających datę i czas powstania danych.
- **Jakość usług (Quality of Service (QOS))**: System i dane muszą być dostępne w momencie, kiedy są potrzebne
- **Kontrola (Auditing)**: Oznacza metody utrzymywania kontroli mechanizmów bezpieczeństwa w celu poprawy ich efektów

# 4. Java EE - Poziomy implementacji mechanizmów bezpieczeństwa




Mechanizmy bezpieczeństwa Java EE są oddzielone od systemu operacyjnego

# 5. Poziomy zabezpieczeń

## 5.1. Zabezpieczenia na poziomie aplikacji – realizowana na poziomie kontenerów:

**Zalety:**

- Bezpieczeństwo jest dedykowane aplikacji.
- Bezpieczeństwo jest reprezentowane przez „ziarna” powiązane z parametrami aplikacji.

**Wady:**

- Aplikacja jest zależna od atrybutów mechanizmu bezpieczeństwa różnych dla różnych typów aplikacji
- Nie można zabezpieczyć różnych protokołów
- Nie można zabezpieczyć danych

# 5.2. Zabezpieczenia na poziomie protokołu transportowego

**Protokół *HTTP z użyciem protokołu SSL (Secure Sockets Layer) nazywany jest protokołem HTTPS.* Bezpieczeństwo protokołu transportu jest mechanizmem bezpieczeństwa typu „point-to-point", który określa uwierzytelnianie, integralność wiadomości i jej poufność.**

**Fazy ustalania bezpieczeństwa poziomu transportu:**

- Klient i serwer www uzgadniają właściwy algorytm kryptograficzny.
- Następuje uwierzytelnienie serwera www (czasem również klienta) za pomocą cyfrowego certyfikatu oraz wysłanie klucza prywatnego zw. kluczem sesji (wygenerowanego przez przeglądarkę) przy użyciu szyfrowania kluczem publicznym serwera.
- Wykorzystuje się szyfrowanie symetryczne wymienianych wiadomości za pomocą klucza prywatnego, który ma teraz serwer i przeglądarka.

**Zalety:**

- Relatywnie prosty, zrozumiały i standardowy
- Stosowany do ciała wiadomości i załączników

**Wady:**

- Duże powiązanie z poziomem protokołu transportu wiadomości
- „Wszystko lub nic" z punktu widzenia bezpieczeństwa. Nie można szyfrować fragmentów wiadomości.
- Bezpieczeństwo jest zapewnione jedynie podczas transmisji wiadomości. Zabezpieczenia są usuwane automatycznie po odbiorze wiadomości

# 5.3. Zabezpieczenia na poziomie wiadomości

**Bezpieczeństwo informacji obejmuje wiadomości SOAP i/lub załączniki tej wiadomości.**

***Serwer aplikacji* i *Pakiet serwisu Usług Internetowych* (*Java Web Services Developer Pack* (*Java WSDP*)) wspierają mechanizmy bezpieczeństwa:**

**Zalety:**

- Zabezpieczenia wiadomości są również obecne po jej odebraniu.
- Jest reprezentowana przez „ziarna”. Można dlatego zabezpieczać części wiadomości i/lub załączniki używając XWSS (XML and *Web Services Security* )
- Może być łączona z innymi zabezpieczeniami.
- Jest niezależna od środowiska aplikacji i protokołu transportowego

**Wady :** jest to złożony mechanizm obciążający wydajność procesu

# 6. Rodzaje mechanizmów bezpieczeństwa w kontenerach

- **Deklaratywne mechanizmy bezpieczeństwa** – deklarowane za pomocą tzw. „*deployment descriptors" (deskryptory aplikacji np.* ***web.xml*** dla aplikacji typu ***web*** *)*. Deskryptory jako zewnętrzny element aplikacji zawierają informację specyfikującą role bezpieczeństwa i wymagania dostępu są mapowane w role specyficzne dla środowiska oraz użytkowników i polisy bezpieczeństwa.

- ***Programowe mechanizmy bezpieczeństwa*** - są osadzone w aplikacji i służą do podejmowanie decyzji o bezpieczeństwie. Uzupełniają deklaratywne mechanizmy bezpieczeństwa – lepiej wyrażają model bezpieczeństwa aplikacji. API mechanizmów programowych:
  - metody interfejsu EJBContext
  - metody interfejsu HttpServletRequest. Metody te pozwalają na podejmowanie decyzji biznesowych opartych na rolach bezpieczeństwa nadawcy lub zdalnego odbiorcy

- **Adnotacje lub metadane** są używane do specyfikowania informacji wewnątrz pliku z kodem klasy. Kiedy aplikacja jest uruchamiana, informacja ta jest używana lub pokrywana przez deskryptor aplikacji.

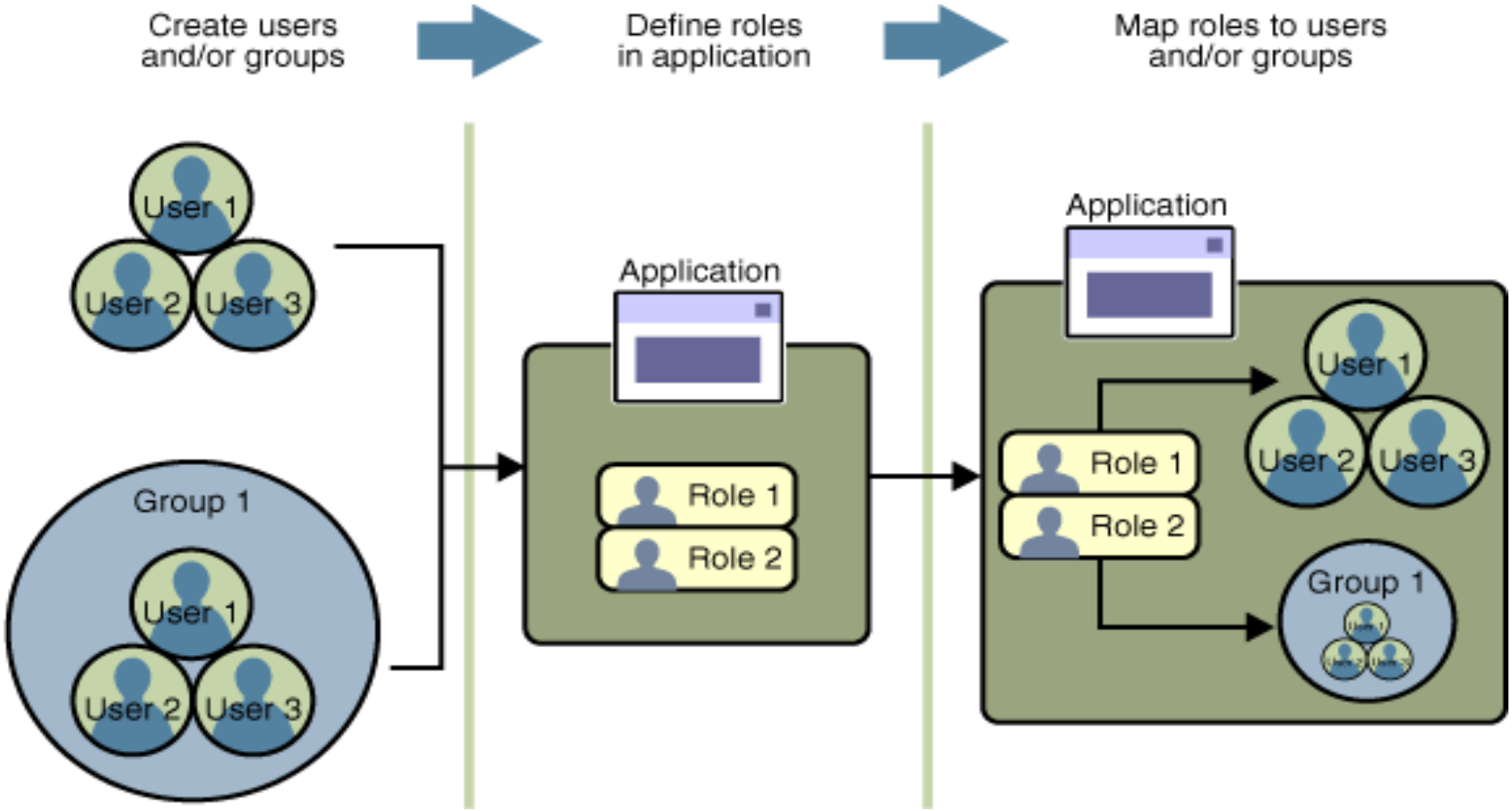
Np.

```
@DeclareRoles(„klient") public class Page1 extends AbstractPageBean
  {  //... }
```

# 7. Rodzaje mechanizmów bezpieczeństwa
## w serwerze aplikacji

- Dodawanie, usuwanie i modyfikowanie autoryzowanych użytkowników
- Dodawanie, usuwanie i modyfikowanie **baz użytkowników i grup** lub danych użytkownika
- Konfiguracja bezpieczeństwa HTTP i IIOP
- Konfiguracja bezpieczeństwa połączeń JMX.
- Definiowanie interfejsu dla narzędzi autoryzacji z wykorzystaniem *Java Authorization Contract for Containers* (JACC).
- Definiowanie kontraktów bezpieczeństwa między Serwerem Aplikacji a modułami polis bezpieczeństwa – wykorzystanie *Java Authorization Contract for Containers* (JACC) (specyfikuje narzędzia autoryzacji: konfiguracje, sposób użycia w kontroli dostępu)
- Użycie dołączanych modułów kontroli
- Ustawianie i zmiana uprawnień dla aplikacji

# 8. Bazy użytkowników i grup, Użytkownik, Grupa, Rola



# 8.1. Praca z bazami użytkowników, użytkownikami, grupami i rolami

Często należy zabezpieczyć zasoby przed niepożądanym dostępem za strony użytkowników.

Autoryzacja pozwala kontrolować dostęp do zasobów chronionych

- Programista pisze kod żądający od użytkownika nazwy i hasła
- Programista oznajmia, jak **ustawić bezpieczeństwo uruchomianej aplikacji za pomocą deskryptora aplikacji** (bezpośrednio lub pośrednio za pomocą adnotacji)
- Administrator serwera **ustawia autoryzowanych użytkowników i grupy na serwerze aplikacji**.
- Programista **mapuje role bezpieczeństwa aplikacji na użytkowników, grupy i zleceniodawców zdefiniowanych na serwerze aplikacji.**

# 8.2. Baza użytkowników na serwerze aplikacji

**W aplikacjach www baza danych użytkowników i grup które umożliwiają identyfikację właściwych użytkowników aplikacji www lub grup aplikacji www i są kontrowane przez te sama polisę uwierzytelniającą.**

- **Serwer aplikacji dla aplikacji Java EE** posiada usługę uwierzytelniania za pomocą baz użytkowników. Istnieją następujące typy baz: admin-realm, file, bazy certyfikatów

- **W bazie typu *file* (plik o nazwie keyfile)** serwer przechowuje listy uwierzytelniające użytkowników, zarządzanych za pomocą narzędzia Admin Console W bazie typu ***file*** usługa uwierzytelniania przez serwer polega na sprawdzeniu wszystkich użytkowników w bazie typu *file* oprócz tych, którzy używają protokołów HTTPS i certyfikatów.

- **W bazie certyfikatów (certificate realm)** serwer przechowuje listy uwierzytelniające użytkowników internetowych. Kiedy używana jest baza certyfikatów, serwer używa certyfikatów razem z protokołem HTTPS do uwierzytelniania użytkowników internetowych. Do weryfikacji serwer używa certyfikat X.509. Pole *name* tego certyfikatu jest używane jako principal name.

- **Baza typu *admin-realm* (plik o nazwie admin-keyfile)** przechowuje listy uwierzytelniające administratorów*, zarządzane przez narzędzie Admin Console*.

- **Użytkownik?**
  - Użytkownik jest indywidualną tożsamością zdefiniowaną w Serwerze aplikacji.
  - W aplikacji internetowej użytkownik ma zbiór ról związanych z tożsamością, która określa dostęp do zasobów aplikacji. Użytkownicy gromadzeni są w grupy.
  - Użytkownik Java EE jest podobny do użytkownika systemu operacyjnego. Może być nim człowiek. Jednak nie ma powiązania między systemem uwierzytelniania w systemie operacyjnym i aplikacji JavaEE.
- **Grupa?**
  - Grupa jest zbiorem uwierzytelnianych użytkowników zdefiniowanych w Serwerze Aplikacji.
  - Użytkownik Java EE zapisany w bazie **typu *file*** może należeć do grupy, natomiast użytkownik z bazy **typu *certificate*** nie może należeć do grupy. Grupa jest tworzona wg potrzeb związanych z typem pracy użytkownika np. *klient* w grupie *klienci*
  - Grupa jest związana z przechowywaniem jej na serwerze aplikacji, natomiast rola jest związana z aplikacją na serwerze aplikacji.
- **Rola?**
  - Rola jest abstrakcyjną nazwą związaną z pozwoleniem na dostęp do wyspecyfikowanych zasobów *w* aplikacji.

## 8.3.Definiowanie mechanizmów bezpieczeństwa w deskryptorze aplikacji : ról, kolekcji url, transportu danych

```
<security-constraint>
    <display-name>KlientConstraint</display-name>
    <web-resource-collection>
        <web-resource-name>Klient</web-resource-name>
        <description/>
        <url-pattern>/faces/Page1.jsp</url-pattern>
        <url-pattern>/faces/Tytuly.jsp</url-pattern>
        <url-pattern>/faces/Ksiazki.jsp</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
        <http-method>HEAD</http-method>
        <http-method>PUT</http-method>
        <http-method>OPTIONS</http-method>
        <http-method>TRACE</http-method>
        <http-method>DELETE</http-method>
    </web-resource-collection>
    <auth-constraint>
        <description/>
        <role-name>klient1</role-name>
    </auth-constraint>
    <user-data-constraint>
        <description/>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

```
<security-constraint>
    <display-name>AdministratorConstraint</display-name>
    <web-resource-collection>
        <web-resource-name>Administrator</web-resource-name>
        <description/>
        <url-pattern>/faces/*</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
        <http-method>HEAD</http-method>
        <http-method>PUT</http-method>
        <http-method>OPTIONS</http-method>
        <http-method>TRACE</http-method>
        <http-method>DELETE</http-method>
    </web-resource-collection>
    <auth-constraint>
        <description/>
        <role-name>administrator1</role-name>
    </auth-constraint>
    <user-data-constraint>
        <description/>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

## Ustawianie sposobu uwierzytelniania użytkowników i ustawiania ról w deskryptorze aplikacji web.xml

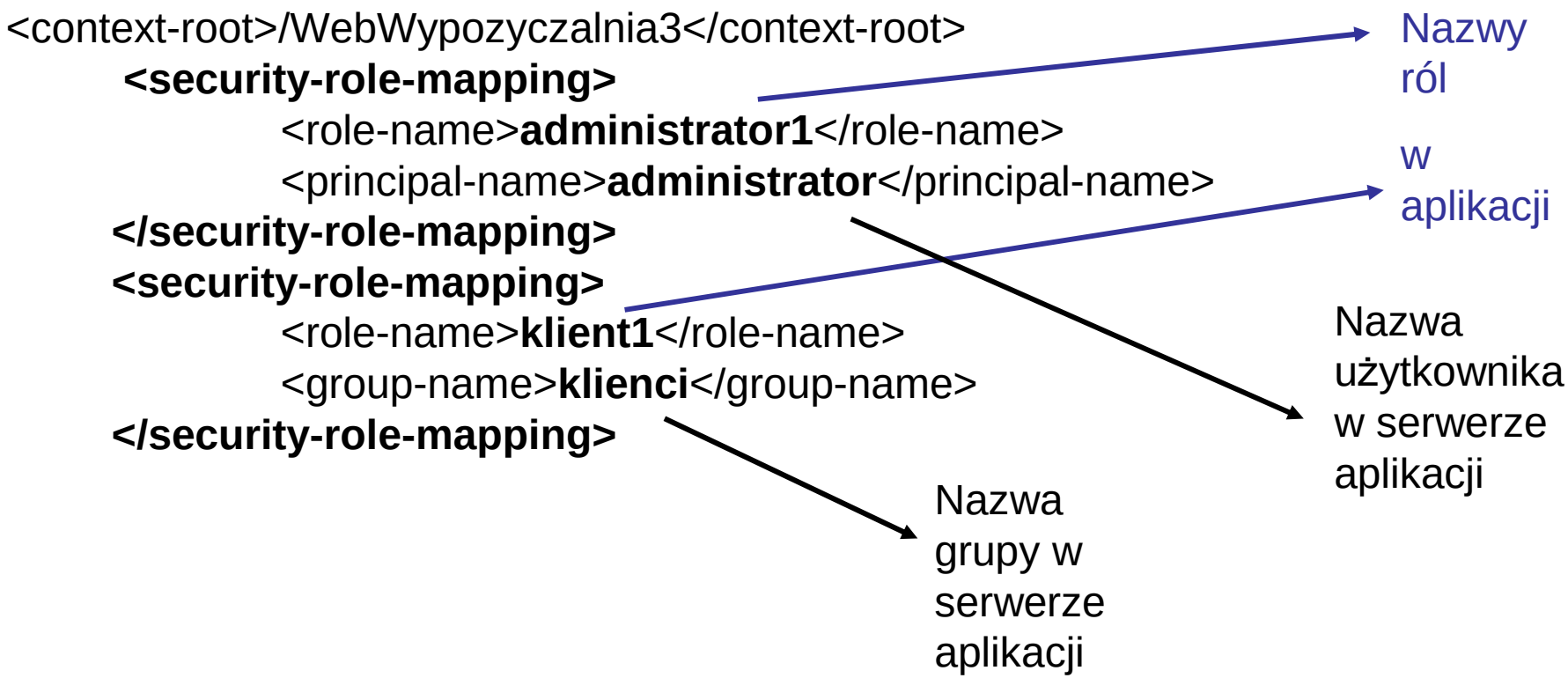
```
  <login-config>
    <auth-method>FORM</auth-method>
    <realm-name>file</realm-name>
    <form-login-config>
      <form-login-page>/logon.jsp</form-login-page>
      <form-error-page>/logonError.jsp</form-error-page>
    </form-login-config>
  </login-config>

  <security-role>
    <description/>
    <role-name> klient1 </role-name>
  </security-role>
  <security-role>
    <description/>
    <role-name> administrator1 </role-name>
  </security-role>
</web-app>
```

# 8.4. Mapowanie ról do użytkowników i grup

Podczas uruchamiania aplikacji Java EE korzysta się z mechanizmu mapowania ról nadanych w aplikacji do użytkowników lub grup w plikach typu **file** lub **admin-realm**. **Mapowane są role** do **użytkowników** i **grup** zdefiniowanych w serwerze aplikacji.

Wykonuje się to w deskryptorze serwera aplikacji (w zależności od typu aplikacji: sun-application.xml, **sun-web.xml**, or sun-ejb-jar.xml).

Należy zdefiniować w deskryptorze serwera mapowanie roli, zdefiniowanej w deskryptorze aplikacji **web.xml,** za pomocą nazwy *group* lub nazwy *principals* odpowiadające nazwie jednego lub wielu użytkownikom lub jednej lub wielu grupom w pliku typu ***file*** lub ***admin-realm*** *w serwerze aplikacji.*



# 9. Java SE - Implementacja mechanizmów bezpieczeństwa

- **Java Authentication and Authorization Service (JAAS)**

  APIs obsługujące usługi uwierzytelniania i autoryzacji dostępu użytkowników do zasobów systemu.

- **Java Generic Security Services (Java GSS-API)**

  Java GSS-API jest oparte na żetonach API używane do bezpiecznej wymiany komunikatów między aplikacjami. Dostarcza ona zunifikowany zestaw usług zapewniających mechanizmy bezpieczeństwa, włączając Kerberos.

- **Java Cryptography Extension (JCE)**

  JCE dostarcza framework i implementację mechanizmów szyfrowania, generowania kluczy, serializowania obiektów

- **Java Secure Sockets Extension (JSSE)**

  JSSE dostarcza framework i implementację protokołów SSL and TLS i zawiera funkcje szyfrowania, uwierzytelniania serwerów, integralności wiadomości i uwierzytelniania klientów podczas komunikacji przez Internet.

- **Simple Authentication and Security Layer (SASL)**

  SASL jest standardem internetowym (RFC 2222), który specyfikuje protokół do uwierzytelniania i zapewnienia bezpieczeństwa podczas komunikacji klient-serwer. SASL definiuje sposób uwierzytelniania przesyłanych danych bez specyfikowania zawartości danych.