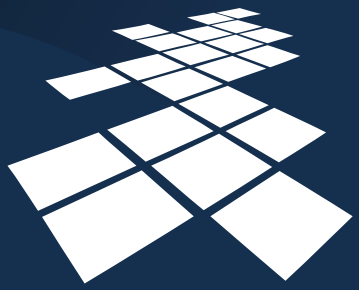
# Infrastruktura aplikacji WWW II

Wykład prowadzi:
Marek Wojciechowski



UCZELNIA
ONLINE


## Plan wykładu

- Infrastruktura w aplikacjach Java EE – JavaServer Faces (JSF)



Celem wykładu jest przedstawienie technologii JavaServer Faces (JSF), ułatwiającej tworzenie aplikacji Java EE poprzez dostarczenie infrastruktury dla stanowego, komponentowego interfejsu użytkownika.


## JavaServer Faces (JSF)

- Technologia, która ma ułatwić i ustandaryzować tworzenie interfejsu użytkownika w aplikacjach WWW na platformie Java EE
- Część specyfikacji Java EE
- Oparta o stanowe, konfigurowalne komponenty interfejsu użytkownika pracujące po stronie serwera
- Koncepcja podobna do Web Forms w ASP.NET
- Główne elementy technologii JSF:
  - API do reprezentowania komponentów interfejsu użytkownika i zarządzania nimi
  - 2 biblioteki znaczników dla JavaServer Pages (JSP)



JavaServer Faces (JSF) to technologia, która ma ułatwić i ustandaryzować tworzenie interfejsu użytkownika w aplikacjach WWW na platformie Java EE. Jest efektem pracy grupy ekspertów reprezentujących różne środowiska, w tym Apache Struts oraz firm takich jak Sun, Oracle i IBM. W przeciwieństwie do konkurencyjnych rozwiązań, JSF ma status standardu jako jedna z młodszych specyfikacji w ramach Java EE.

Technologia JSF jest oparta o stanowe, konfigurowalne komponenty interfejsu użytkownika pracujące po stronie serwera. Jest to więc koncepcja podobna do Web Forms dla ASP.NET. JSF w połączeniu z wspierającymi je środowiskami IDE stanowi próbę dorównania platformie .NET w łatwości tworzenia interfejsu użytkownika w aplikacjach WWW. Java EE od początku ma przewagę nad .NET jeśli chodzi o niezależność od konkretnego dostawcy oprogramowania i duży wybór narzędzi i serwerów, w tym darmowych, ale dotychczas nie dorównywała .NET w zakresie tworzenia interfejsu użytkownika.

Główne elementy technologii JSF to API do reprezentowania komponentów interfejsu użytkownika i zarządzania nimi oraz dwie biblioteki znaczników dla JavaServer Pages (JSP), służące do zagnieżdżania komponentów interfejsu w stronach JSP. JSF dostarcza zestaw standardowych, predefiniowanych komponentów i umożliwia tworzenie dodatkowych komponentów.

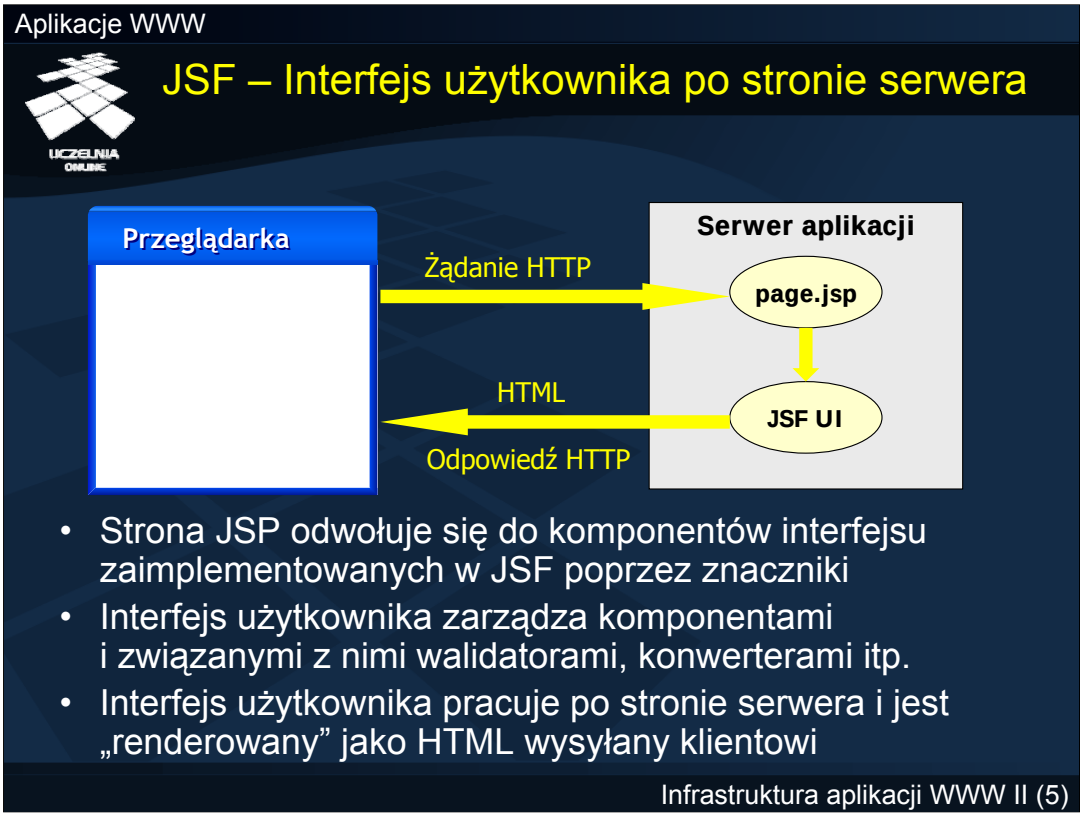

## JSF jako infrastruktura aplikacji WWW

- Zakres funkcjonalny:
  - stanowy, komponentowy interfejs użytkownika
  - obsługa nawigacji między stronami
  - walidacja danych
  - wsparcie dla aplikacji wielojęzycznych
- JSF jako implementacja architektury MVC:
  - model komponentów do tworzenia stron-widoków
  - gotowy, konfigurowalny kontroler: FacesServlet
  - nie obejmuje modelu, współpracuje z różnymi modelami

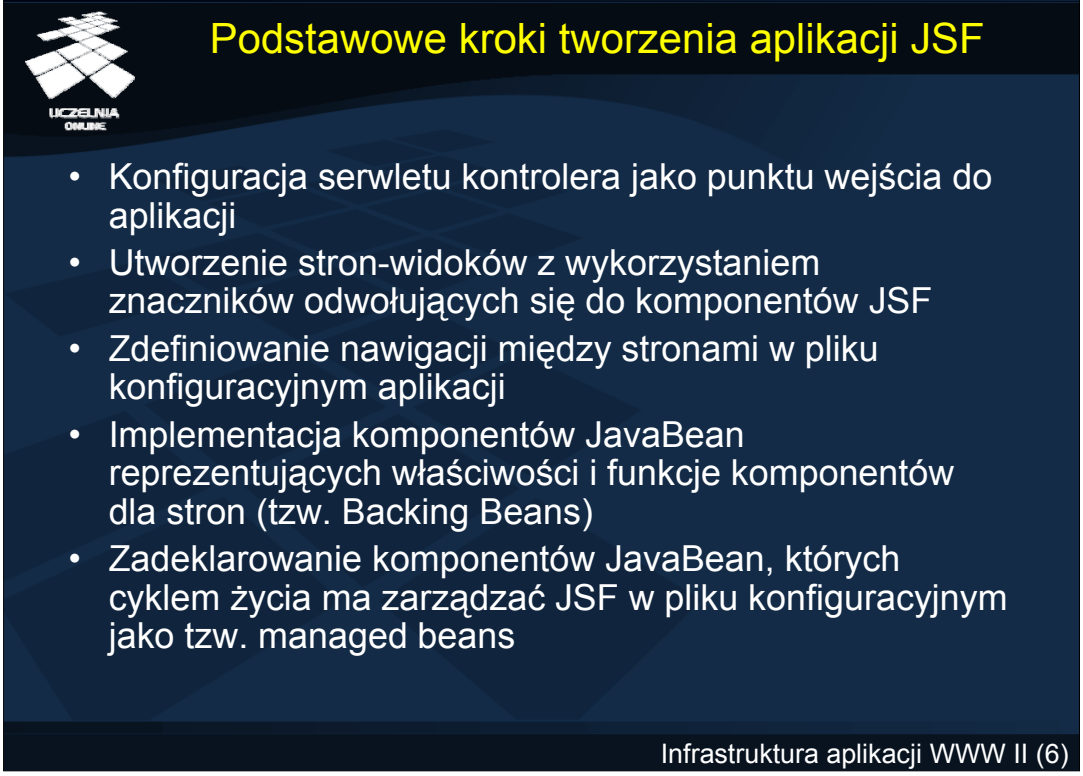


JSF zwiększa produktywność programisty oferując gotowe schematy rozwiązań problemów infrastruktury aplikacji WWW. Głównym celem JSF jest zapewnienie stanowego interfejsu użytkownika w oparciu o komponenty interfejsu, zarządzanie ich stanem i obsługę zdarzeń przez nie generowanych. Ponadto, JSF wspiera programistów w zakresie walidacji danych po stronie serwera, definiowania nawigacji między stronami i tworzenia aplikacji wielojęzycznych.

Architektura JSF jest oparta o wzorzec Model-View-Controller (MVC). Główny nacisk JSF kładzie na dostarczenie modelu komponentów do tworzenia stron-widoków. JSF dostarcza też gotową implementację kontrolera aplikacji WWW w postaci konfigurowalnego serwletu FacesServlet. Skupiając się na interfejsie użytkownika, JSF nie wspiera implementacji modelu i nie wymaga też konkretnego sposobu implementacji modelu. Udostępnia jedynie mechanizmy wiążące obiekty modelu z pozostałymi komponentami aplikacji.




W JSF interfejs użytkownika związany ze stroną aplikacji jest reprezentowany w postaci drzewa komponentów po stronie serwera. Interfejs użytkownika zarządza komponentami i związanymi z nimi walidatorami, konwerterami, itp. Obsługa zdarzeń zachodzących w interfejsie użytkownika odbywa się po stronie serwera, dlatego też każdorazowa zmiana stanu interfejsu wymagająca obsługi zdarzeń pociąga za sobą konieczność wysłania żądania do serwera i przeładowania strony w przeglądarce. Interfejs użytkownika pracujący po stronie serwera jest „renderowany” jako HTML i wysyłany klientowi w ramach odpowiedzi HTTP.




Tworzenie aplikacji JSF wymaga realizacji pięciu zadań, kolejność ich wykonania jest w dużym stopniu dowolna. Te zadania to:

1. Konfiguracja serwletu kontrolera JSF jako punktu wejścia do aplikacji.

2. Utworzenie stron-widoków, typowo jako stron JSP wykorzystujących specjalne znaczniki odwołujące się do komponentów JSF.

3. Zdefiniowanie nawigacji między stronami w pliku konfiguracyjnym aplikacji.

4. Implementacja komponentów JavaBean reprezentujących właściwości i funkcje komponentów dla poszczególnych stron (tzw. Backing Beans).

5. Zadeklarowanie komponentów JavaBean, których cyklem życia ma zarządzać JSF w pliku konfiguracyjnym jako tzw. managed beans. Konfigurowane w ten sposób są komponenty Backing Beans oraz inne, które mają być automatycznie dostępne przy uruchomieniu strony.


## Relacje JSF z JSP i Struts

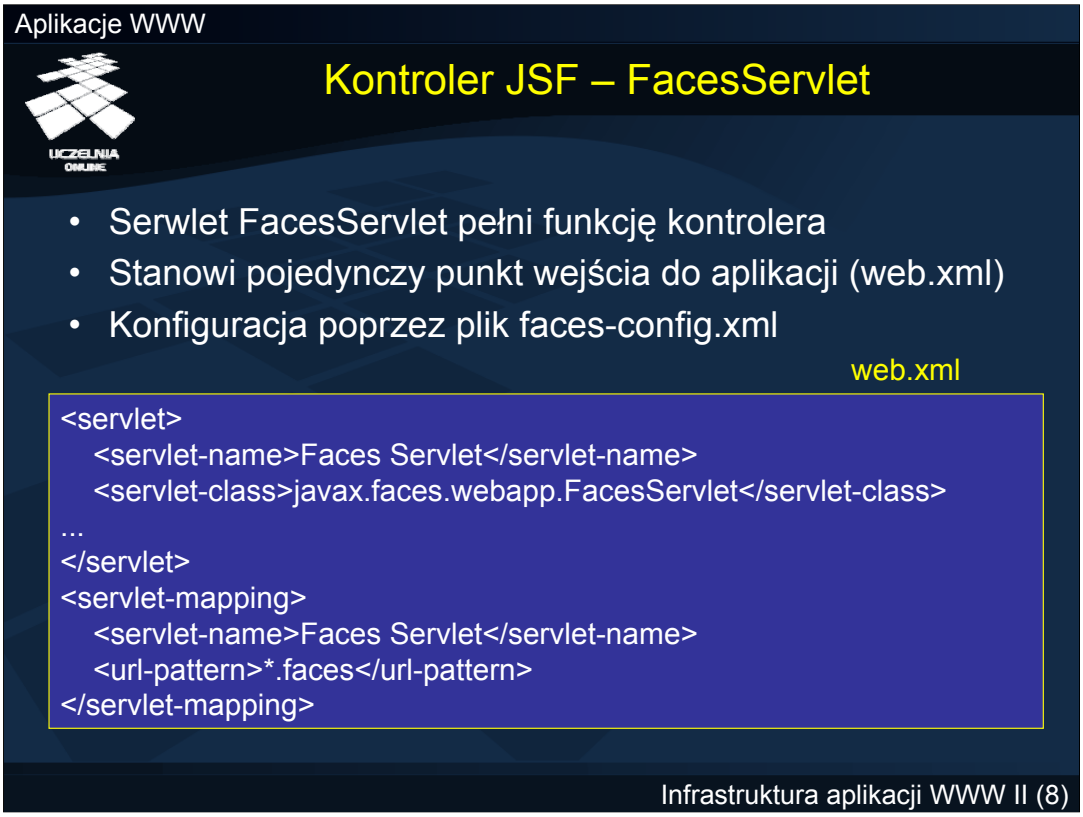
- JSF nie wymaga JSP jako technologii widoku, ale JSP jest domyślną technologią do renderowania komponentów JSF
- Funkcjonalność JSF pokrywa się ze Struts w zakresie obsługi nawigacji i walidacji
  - kontroler i oprogramowywanie nawigacji uproszczone w porównaniu ze Struts
  - nacisk w JSF na kompenenty interfejsu użytkownika, nie będące przedmiotem Struts
  - mniejsze możliwości standardowych walidatorów w JSF niż w Struts, ale są dodatkowe biblioteki



JSF nie wymaga JSP jako technologii widoku, ale JSP jest domyślną i najlepiej wspieraną technologią do renderowania komponentów JSF. Każda implementacja JSF musi posiadać wsparcie dla JSP. Definiowane przez specyfikację biblioteki znaczników JSF dla JSP ułatwiają i standaryzują korzystanie z JSF w połączeniu z JSP.

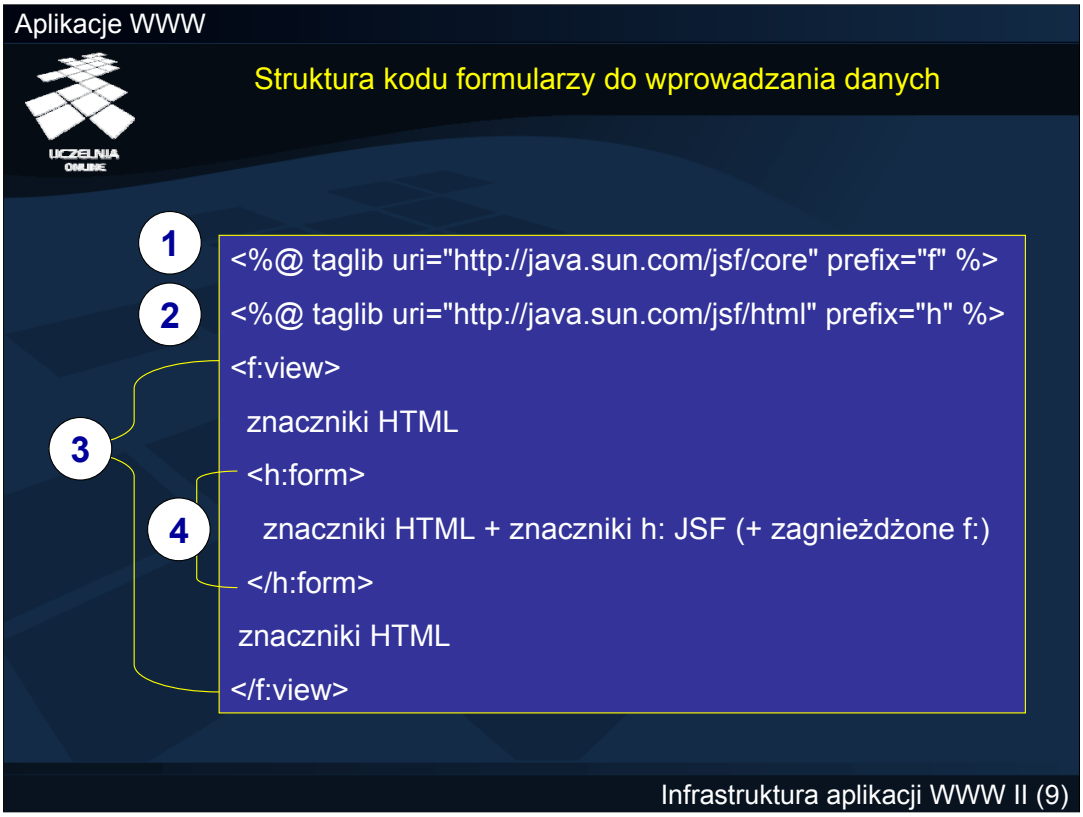
Funkcjonalność JSF pokrywa się ze Struts w zakresie obsługi nawigacji i walidacji. Kontroler JSF został uproszczony w porównaniu ze Struts, ale jego funkcjonalność jest wystarczająca. Technologia JSF jest zorientowana na stanowy, komponentowy interfejs użytkownika i w tym obszarze Struts nie stanowi dla niej konkurencji. Pewną wadą JSF są skromne możliwości standardowych walidatorów i brak w nich walidacji po stronie klienta. Problem ten jednak rozwiązują dodatkowe biblioteki komponentów.

JSF jako technologia oficjalna, nowsza i pokrywająca funkcjonalność Struts wydaje się dziś lepszym rozwiązaniem i powinna z czasem wyprzeć Struts. Struts i JSF mogą też być używane w połączeniu ze sobą: Struts do obsługi nawigacji, a JSF do implementacji stanowych stron-widoków.




Serwlet FacesServlet (pakiet javax.faces.webapp) pełni w aplikacji JSF funkcję kontrolera i jest bardzo podobny do ActionServlet ze Struts. Dzięki odpowiedniej konfiguracji w pliku web.xml (pokazanej na slajdzie) stanowi on pojedynczy punkt wejścia do całego systemu aplikacyjnego. Zwyczajowo, serwlet FacesServlet reaguje na wszystkie żądania HTTP, których adres URL posiada rozszerzenie .faces. Takie odwzorowanie serwletu na ścieżki URL rekomenduje specyfikacja JSF. Alternatywne, popularne odwzorowanie, wykorzystywane np. w środowisku Oracle JDeveloper to <url-pattern>/faces/*</url-pattern>.

FacesServlet to gotowy, w pełni zaimplementowany uniwersalny kontroler. Dla konkretnej aplikacji należy go skonfigurować poprzez XML-owy plik konfiguracyjny – domyślnie WEB-INF/faces-config.xml. Plik ten zawiera m.in. reguły nawigacji i deklaracje komponentów Java Bean, których cyklem życia ma zarządzać JSF.




Strony-widoki JSF to najczęściej strony JSP wykorzystujące specjalne znaczniki z bibliotek znaczników JSF dla JSP. Strony te zawierają jeden formularz i mają ogólną strukturę kodu przedstawioną na slajdzie:

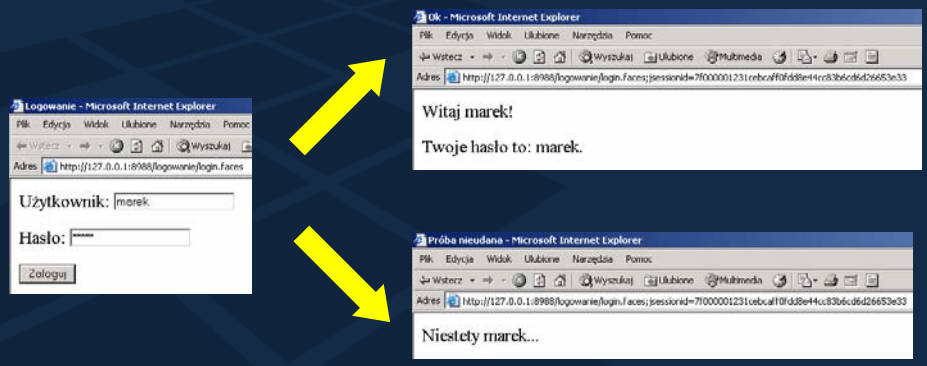
1. Deklaracja biblioteki znaczników reprezentujących elementy JSF, niezależne od technologii prezentacji (np. do walidacji, obsługi zdarzeń i wyróżniony znacznik do reprezentacji korzenia drzewa komponentów), zwyczajowy prefiks: „f”.

2. Deklaracja biblioteki znaczników reprezentujących znaczniki HTML (w tym elementy formularzy), zwyczajowy prefiks : „h”.

3. Strona zawierająca znaczniki JSF jest reprezentowana w postaci drzewa komponentów. Korzeniem drzewa jest komponent UIViewRoot, reprezentowany przez znacznik <f:view>. Dlatego wszystkie znaczniki JSF użyte na stronie JSP muszą być zawarte wewnątrz elementu <f:view></f:view>.

4. Znacznik <h:form> reprezentuje formularz HTML <FORM>. W generowanym przez ten znacznik formularzu atrybut ACTION automatycznie wskazuje na bieżącą stronę (strona wywołuje sama siebie), a atrybut METHOD przyjmuje wartość POST (obowiązkowo).

# Przykładowa aplikacja JSF



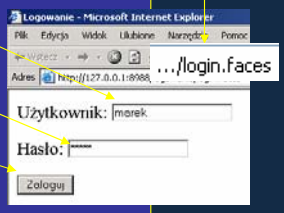



Na kolejnych slajdach tworzenie aplikacji JSF zostanie zilustrowane przykładową aplikacją obsługującą logowanie użytkownika do systemu. Aplikacja obejmuje 3 strony. Pierwsza strona prezentuje formularz logowania. Po weryfikacji nazwy użytkownika i hasła następuje przekierowanie do strony informującej o sukcesie lub niepowodzeniu logowania.

# Formularz logowania

login.jsp

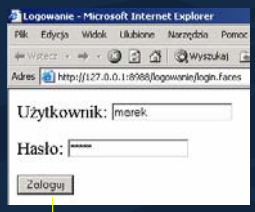
```
<%@ page contentType="text/html;charset=windows-1250"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<f:view><html>
  <head>...<title>Logowanie</title></head>
  <body><h:form>
   <p>Użytkownik: <h:inputText value="#{loginBean.username}"
                               id="username"/>
   <p>Hasło: <h:inputSecret value="#{loginBean.password}"
                            id="password"/>
   <p><h:commandButton value="Zaloguj"
                       id="submitButton"
                       action="#{loginBean.register}"/>
     </h:form></body>
</html></f:view>
```

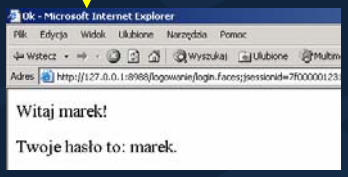





Slajd pokazuje kod źródłowy formularza logowania. Formularz zawiera pole tekstowe <h:inputText>, pole z hasłem <h:inputSecret> i przycisk <h:commandButton>. Znaczenie wyrażeń stanowiących wartości atrybutów znaczników JSF w kodzie strony zostanie wyjaśnione na kolejnych slajdach. Strona zapisana jest w pliku login.jsp. W adresie URL w przeglądarce nazwa pliku ma rozszerzenie .faces, dzięki czemu żądanie wyświetlenia strony będzie kierowane do serwletu kontrolera FacesServlet zgodnie z rekomendowanym sposobem odwzorowania adresów URL w pliku web.xml.

# Nawigacja statyczna




login.jsp

```
<h:commandButton value="Zaloguj"
                 id="submitButton"
                 action="success"/>
```

faces-config.xml

```
<navigation-rule>
  <from-view-id>/login.jsp</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/ok.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```



Dla każdej strony-widoku, możliwe dla niej przypadki nawigacji są definiowane w pliku faces-config.xml w formie tekstowych etykiet związanych z docelowymi widokami.

W najprostszym przypadku nawigacja między stronami JSF ma charakter statyczny tzn. zatwierdzenie formularza zawsze prowadzi do tej samej strony. Konfiguracja nawigacji statycznej sprowadza się do zaszycia etykiety związanej z nawigacją na stałe w definicji przycisku zatwierdzającego formularz. W przykładzie na slajdzie z formularza logowania nawigacja zawsze prowadzi do strony informującej o sukcesie. Jedyna możliwa nawigacja ma etykietę „success”, która jest na sztywno podana jako wartość atrybutu ACTION komponentu przycisku.

W rzeczywistości nawigacja ma najczęściej charakter warunkowy i wybór docelowego widoku zależy od danych wprowadzonych do formularza. Wtedy konieczna jest konfiguracja nawigacji dynamicznej, która będzie omówiona w dalszej części wykładu.

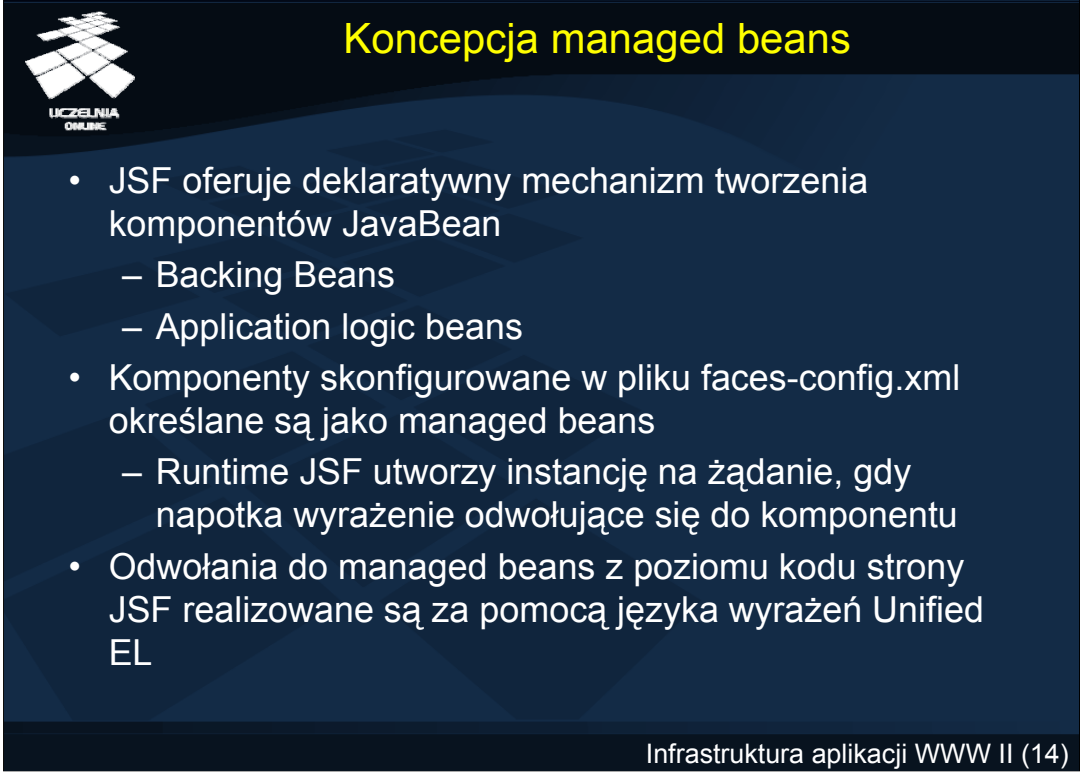

## Komponenty Backing Beans

- Komponenty Backing Beans definiują właściwości i metody powiązane z komponentami interfejsu użytkownika
  - każda właściwość powiązania z wartością lub instancją komponentu
  - dodatkowo backing bean może zawierać metody:
    - do walidacji danych komponentu
    - do obsługi zdarzeń generowanych przez komponent
    - do przetwarzania związanego z nawigacją
- Ich klasy muszą spełniać reguły Java Beans



Komponenty Backing Beans definiują właściwości i metody powiązane z komponentami interfejsu użytkownika tworzącymi strony-widoki. Każda właściwość powiązana jest z wartością lub instancją komponentu. Dodatkowo, Backing Bean może zawierać metody do walidacji danych komponentu, do obsługi zdarzeń generowanych przez komponent i do przetwarzania związanego z nawigacją.

Klasy Backing Beans muszą spełniać reguły JavaBeans tj. bezargumentowy konstruktor, brak publicznych pól, dostęp do właściwości poprzez metody setXXX/getXXX (ewentualnie isXXX zamiast getXXX dla właściwości typu logicznego).




JSF oferuje deklaratywny mechanizm tworzenia komponentów JavaBean. Mechanizm ten dotyczy dwóch rodzajów komponentów:

(a) Backing Beans – komponenty o zasięgu żądania (request), związane z konkretną stroną JSF;

(b) Application logic beans – komponenty niepełniące roli Backing Beans, zawierające kod i właściwości niezwiązane z konkretną stroną.

Komponenty skonfigurowane w pliku faces-config.xml określane są jako managed beans, czyli komponenty zarządzane. JSF utworzy instancję komponentu zarządzanego na żądanie, gdy napotka wyrażenie odwołujące się do komponentu, a instancja komponentu w żądanym zasięgu nie będzie dostępna.

Odwołania do managed beans z poziomu kodu strony JSF realizowane są za pomocą zunifikowanego języka wyrażeń Unified Expression Language (EL).

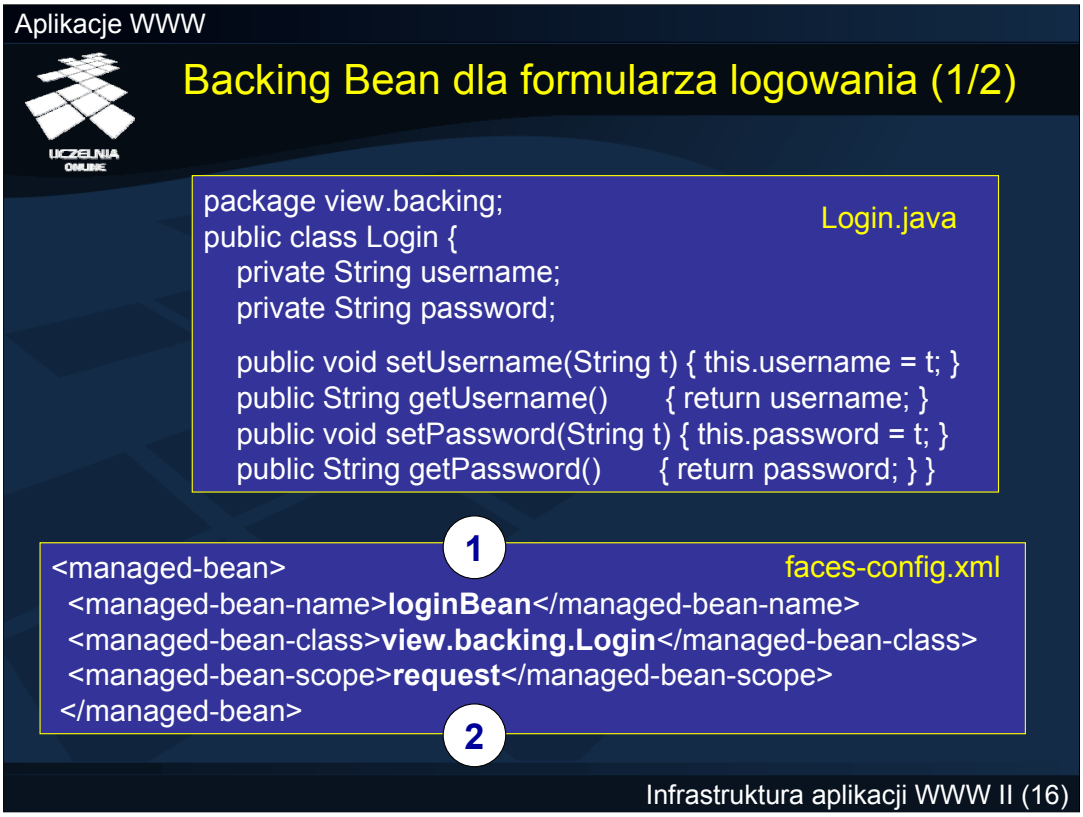

## Unified Expression Language

- Połączenie języka wyrażeń (EL) JSP z językiem wyrażeń opracowanym dla pierwszych wersji JSF
- JSP wykorzystuje natychmiastowe wartościowanie wyrażeń
  - notacja ${...}
  - tylko odczyt danych (wyrażenia read-only)
  - dostęp do właściwości obiektów
- JSF wykorzystuje odroczone wartościowanie wyrażeń
  - notacja #{...}
  - odczyt i zapis danych
  - dostęp do właściwości i metod obiektów



W JSP od wersji 2.0 dostępny jest język wyrażeń ułatwiający dostęp do właściwości obiektów dostępnych z poziomu strony JSP. Dla technologii JSF opracowano identyczny składniowo język, ale dodatkowo umożliwiający zapis danych i wywoływanie metod na rzecz obiektów dostępnych ze strony. Ze względu na inne znaczenie tych samych wyrażeń, dla języka wyrażeń JSF ustalono inny prefiks: # zamiast $. Wyrażenia z prefiksem # na stronach JSF używane są tylko w atrybutach znaczników JSF i służą do dostępu do managed beans i obiektów w zasięgach request, session i application.

W specyfikacji Java EE 5 oba języki połączone zostały w jeden zunifikowany język Unified Expression Language (Unified EL) z zachowaniem odmiennych prefiksów zależnych od zastosowania wyrażeń. Notacja ${...} oznacza wyrażenia wartościowane natychmiastowo i jest wykorzystywana w JSP do odczytu właściwości obiektów. Notacja #{...} oznacza wyrażenia wartościowane z opóźnieniem i jest wykorzystywana w JSF do odczytu i ustawiania właściwości obiektów oraz wywoływania metod na rzecz obiektów.




Na slajdzie pokazano przykład komponentu Backing Bean dla strony logowania. Kod źródłowy klasy komponentu został zawarty w pliku Login.java. Jest to klasyczna klasa JavaBean, określana również jako POJO (ang. Plain Old Java Object), zawierająca właściwości reprezentujące zawartość pól formularza i metody setXXX/getXXX dostępu do nich. Klasa Backing Bean nie musi dziedziczyć z żadnej konkretnej klasy bibliotecznej (inaczej niż przy tworzeniu klasy dla Form Bean w Struts).

Aby instancja komponentu była tworzona automatycznie przy odwołaniu do strony, komponent musi być skonfigurowany jako managed bean w pliku konfiguracyjnym faces-config.xml, co ilustruje fragment zawartości pliku u dołu slajdu. Komponentowi została nadana nazwa logiczna „loginBean” (1) i zasięg żądania (2). Zazwyczaj najbardziej odpowiednim zasięgiem dla Backing Beans jest zasięg żądania, obejmujący bieżącą stronę i stronę, do której nastąpi przekierowanie w wyniku nawigacji.


# Backing bean dla formularza logowania (2/2)

login.jsp

```
<%@ page contentType="text/html;charset=windows-1250"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<f:view>
 <html><head>...<title>Logowanie</title></head>
  <body><h:form>
   <p>Użytkownik: <h:inputText value="#{loginBean.username}"
                               id="username"/>
   <p>Hasło: <h:inputSecret value="#{loginBean.password}"
                            id="password"/>
  ...
  </h:form></body></html>
</f:view>
```



Slajd pokazuje sposób związania komponentów interfejsu zawartych w formularzu logowania z właściwościami komponentu Backing Bean. Powiązanie zostało zrealizowane przez wartość (atrybut VALUE znacznika komponentu). Wyrażenia języka wyrażeń wskazujące właściwości komponentu Backing Bean odwołują się do jego logicznej nazwy zdefiniowanej w faces-config.xml.


## Sposoby wiązania Backing Beans z komponentami interfejsu

- Każda z właściwości komponentu Backing Bean powiązana jest z wartością lub instancją komponentu
- Powiązanie właściwości z wartością komponentu:
  - np. <h:inputText id="uName" value="#{UserBean.userName}" />
  - zalecane w większości sytuacji
- Powiązanie właściwości z instancją komponentu:
  - np. <h:inputText binding="#{UserBean.userNameComponent}" />
  - wymagane gdy istnieje potrzeba programowego modyfikowania właściwości komponentu



Backing Bean to komponent JavaBean, powiązany z komponentami interfejsu używanymi na stronie. Każda z właściwości komponentu Backing Bean powiązana jest z wartością lub instancją komponentu.

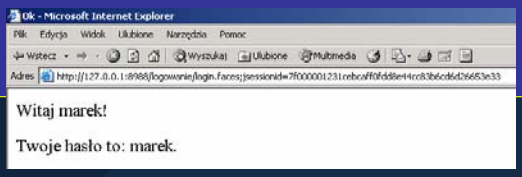
W większości sytuacji zalecane jest wiązanie właściwości z wartością komponentu poprzez atrybut VALUE znacznika komponentu. Właściwość wtedy jest wartością standardowego typu języka Java: String, Integer, int, double, ... Zaletą wiązania przez wartość są automatyczne konwersje typów danych i niezależność klasy komponentu Backing Bean od JSF API.

Drugi sposób to wiązanie właściwości Backing Bean z instancją komponentu interfejsu poprzez atrybut BINDING znacznika komponentu. Właściwość jest wtedy instancją jednej z podklas klasy bibliotecznej JSF UIInput. Sposób ten pozwala na programowe modyfikowanie właściwości komponentu.

## Przekazywanie wartości między stronami poprzez Backing Beans

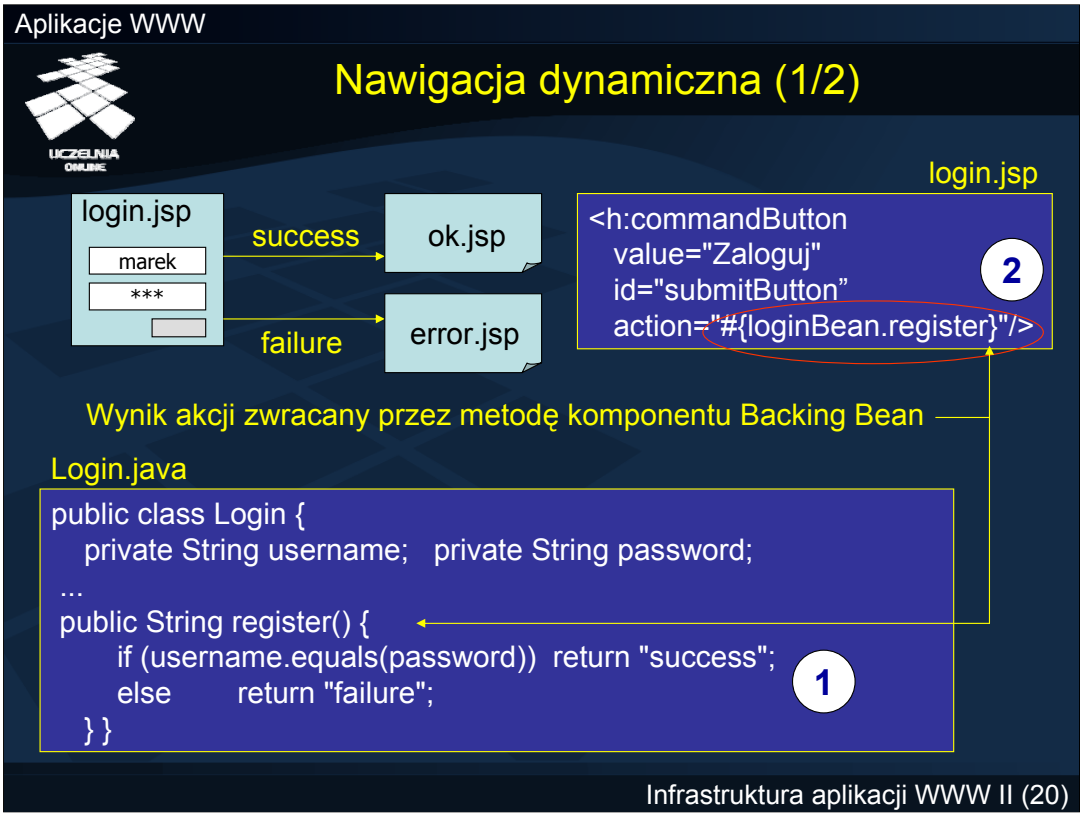
ok.jsp

```
<%@ page contentType="text/html;charset=windows-1250"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<f:view>
 <html> <head><title>Ok</title></head>
  <body><h:form>
   <p>Witaj <h:outputText value="#{loginBean.username}"/>!
   <p>Twoje hasło to: <h:outputText value="#{loginBean.password}"/>.
  </h:form></body>
  </html>
</f:view>
```






Komponenty Backing Beans mogą być wykorzystane do przekazywania danych z formularzy do stron wynikowych. W JSF zatwierdzenie formularza powoduje jego ponowne wywołanie. Żądanie ponownego wywołania formularza jest kierowane do serwletu kontrolera, który w wypadku pozytywnej walidacji danych dokona przekierowania do odpowiedniej strony wynikowej. Typowy dla Backing Beans zasięg request pozwala na odczyt parametrów z formularza zapisanych w komponencie Backing Bean z poziomu strony wynikowej. W ten właśnie sposób przedstawiona na slajdzie strona, informująca o sukcesie logowania, odczytuje wprowadzone w formularzu nazwę i hasło użytkownika. Wyświetlenie wartości wyrażenia w generowanym dokumencie HTML jest realizowane znacznikiem <h:outputText>.




Slajd pokazuje implementację nawigacji dynamicznej dla przykładowej aplikacji obsługującej logowanie. Tym razem, po zatwierdzeniu formularza nastąpi przejście do strony ok.jsp gdy logowanie się powiedzie lub do error.jsp w przeciwnym wypadku. Z dwoma warunkami nawigacji zostały związane etykiety „success” i „failure”.

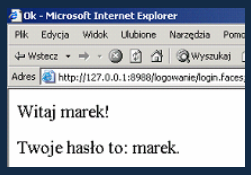
W celu implementacji dynamicznej nawigacji należy utworzyć metodę akcji, która w zależności od wprowadzonych do formularza danych zwróci jedną z etykiet nawigacji. Rozwiązanie to bardzo przypomina mechanizm znany ze Struts, ale w JSF metoda akcji nie jest tworzona w specjalnej klasie akcji, tylko w zwykłej klasie, najczęściej klasie komponentu Backing Bean związanego ze stroną. Etykiety zwracane przez metodę akcji w JSF to zwykłe obiekty String. W przykładzie na slajdzie (1) utworzona została metoda akcji register(), zwracająca „success” gdy hasło jest takie samo jak nazwa użytkownika, a „failure” w przeciwnym wypadku.

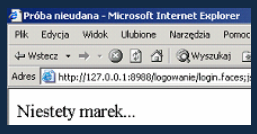
W definicji przycisku w formularzu, wartość etykiety nawigacji w atrybucie ACTION nie może już być podana statycznie. Jest ona tym razem zdefiniowana w postaci wyrażenia języka wyrażeń, wywołującego metodę akcji komponentu Backing Bean – w naszym przykładzie (2) metodę register() komponentu loginBean.

# Nawigacja dynamiczna (2/2)

faces-config.xml

```
...
<navigation-rule>
  <from-view-id>/login.jsp</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/ok.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>failure</from-outcome>
    <to-view-id>/error.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
...
```






Etykiety nawigacji zwracane przez metodę akcji związaną z formularzem muszą zostać zdefiniowane jako reguła nawigacji i związane z widokami, do których prowadzą, w pliku konfiguracyjnym faces-config.xml, co ilustruje niniejszy slajd. Jeśli metoda akcji zwróci wartość nieuwzględnioną w regule nawigacji (lub null), nastąpi ponowne wywołanie bieżącej strony.


JSF oferuje kilka możliwości w zakresie walidacji danych. Te możliwości to:

1. Walidacja ręczna, rozumiana jako walidacja w metodzie akcji komponentu Backing Bean. W tym wypadku twórca aplikacji koduje logikę walidacji, wymusza ponowne wyświetlenie formularza poprzez zwrócenie null w metodzie akcji oraz oprogramowuje wyświetlanie komunikatów o błędach walidacji.

2. Niejawna walidacja automatyczna, obejmująca wykorzystanie atrybutu REQUIRED znaczników komponentów i automatyczną walidację w ramach konwersji danych gdy wykorzystywane są w Backing Beans właściwości typów prostych np. int, double. W przypadku niewypełnienia obowiązkowego pola lub wprowadzenia wartości, dla której konwersja wartości się nie powiedzie, automatycznie nastąpi powrót do formularza wprowadzania danych. Komunikaty o błędach umieszcza się w formularzu za pomocą znacznika <h:message>.

3. Jawna walidacja automatyczna, poprzez użycie predefiniowanych walidatorów: walidatora długości <f:validateLength> lub walidatorów przedziałowych <f:validateDoubleRange>, <f:validateLongRange>. W przypadku błędu walidacji automatycznie nastąpi powrót do formularza wprowadzania danych. Komunikaty o błędach umieszcza się w formularzu za pomocą znacznika <h:message>.

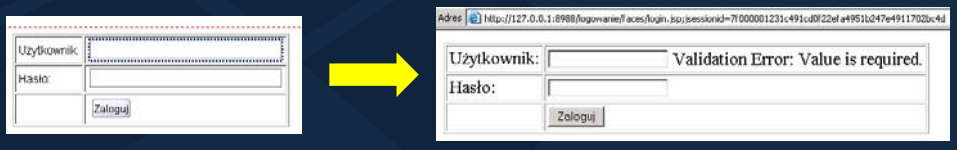
4. Walidatory aplikacji (ang. custom validators), implementujące interfejs Validator, rejestrowane w faces-config.xml.

# Walidacja w JSF – Przykład 1

- Obowiązkowa nazwa użytkownika

login.jsp

```
...
<h:inputText
    binding="#{loginBean.inputText1}"
    id="inputText1" required="true"/>
<h:message for="inputText1" "/>
...
```





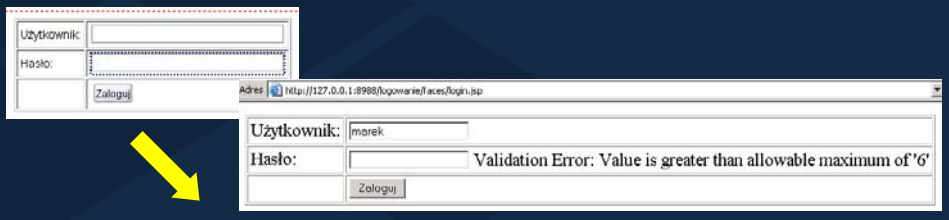

Slajd przedstawia przykład walidacji obowiązkowości wypełnienia pola formularza. W przykładowym formularzu logowania obowiązkowa ma być nazwa użytkownika. W celu wymuszenia obowiązkowości pola, w znaczniku komponentu tworzącego pole należy ustawić wartość „true” atrybutu REQUIRED. Aby po ponownym wyświetleniu błędnie wypełnionego formularza, obok pola z nazwą użytkownika pojawił się komunikat o błędzie, należy w odpowiednim miejscu kodu strony umieścić komponent <h:message>. Komponent <h:message> wyświetla ewentualny komunikat o błędzie związany z komponentem wskazanym atrybutem FOR.

# Walidacja w JSF – Przykład 2

- Hasło od 4 do 6 znaków

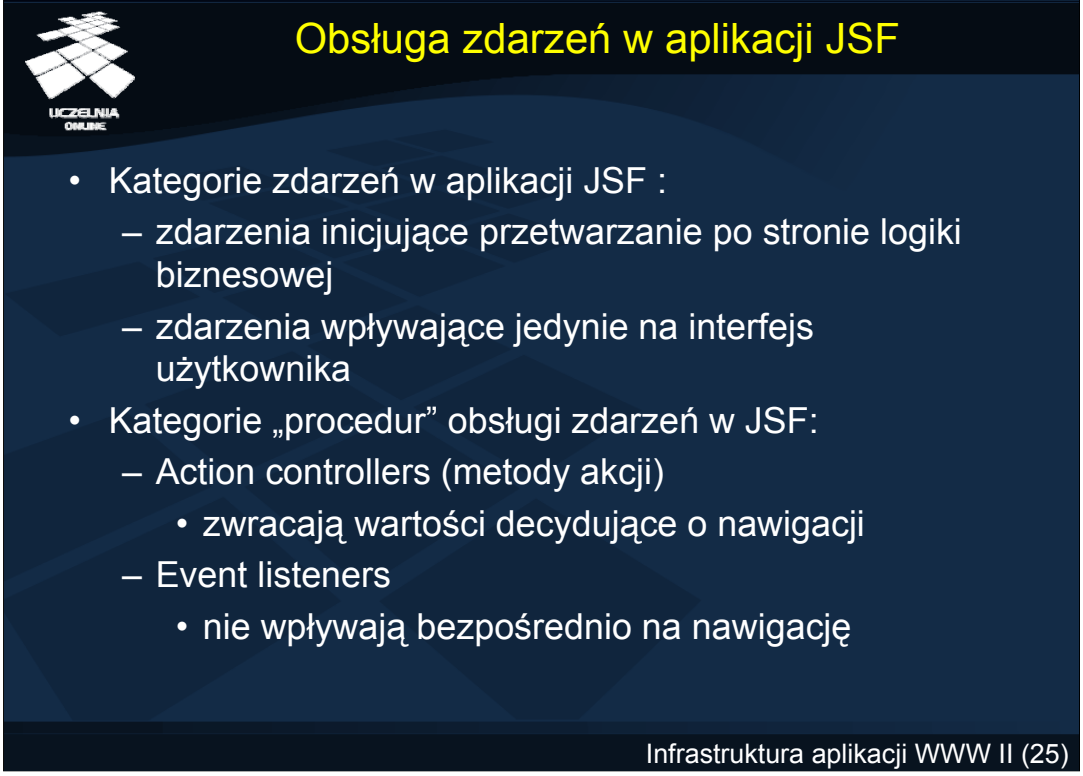
login.jsp

```
<h:inputSecret
   binding="#{loginBean.inputSecret1}"
   id="inputSecret1">
   <f:validateLength maximum="6" minimum="4"/>
</h:inputSecret>
<h:message for="inputSecret1" />
```





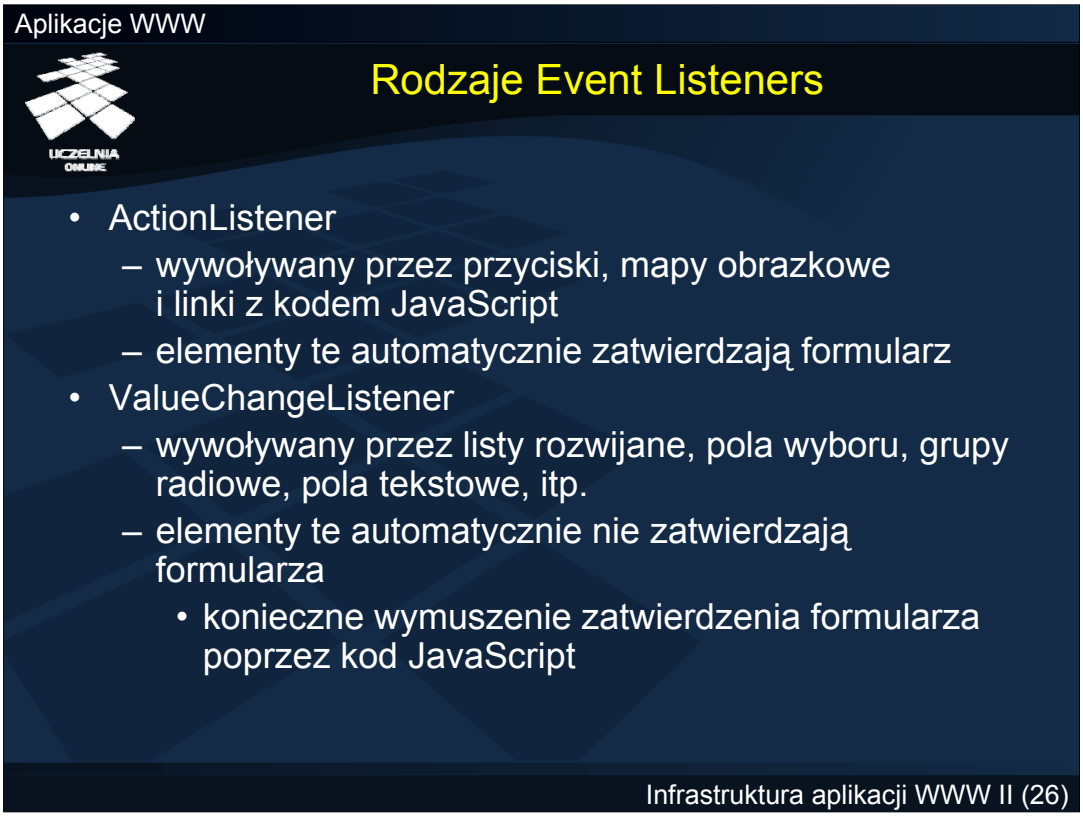

Slajd przedstawia przykład walidacji długości tekstu wprowadzonego do pola formularza. W przykładowym formularzu logowania hasło ma mieć od 4 do 6 znaków. W celu wymuszenia odpowiedniej długości hasła, wewnątrz elementu <h:inputSecret> zawarty został walidator <f:validateLength> z odpowiednio ustawionymi wartościami atrybutów MINIMUM i MAXIMUM. Dodany został również znacznik <h:message> wyświetlający ewentualny komunikat o błędzie związany z walidacją hasła.




Zdarzenia zachodzące w aplikacji JSF można podzielić na dwie kategorie: zdarzenia inicjujące przetwarzanie po stronie logiki biznesowej i zdarzenia wpływające jedynie na interfejs użytkownika. Kategoriom zdarzeń odpowiadają dwie kategorie „procedur” obsługi zdarzeń w JSF:

(a) action controllers (metody akcji) - uruchamiane po wypełnieniu komponentu Backing Bean danymi i po walidacji, zwracające wartości decydujące o nawigacji, inicjujące przetwarzanie po stronie logiki biznesowej.

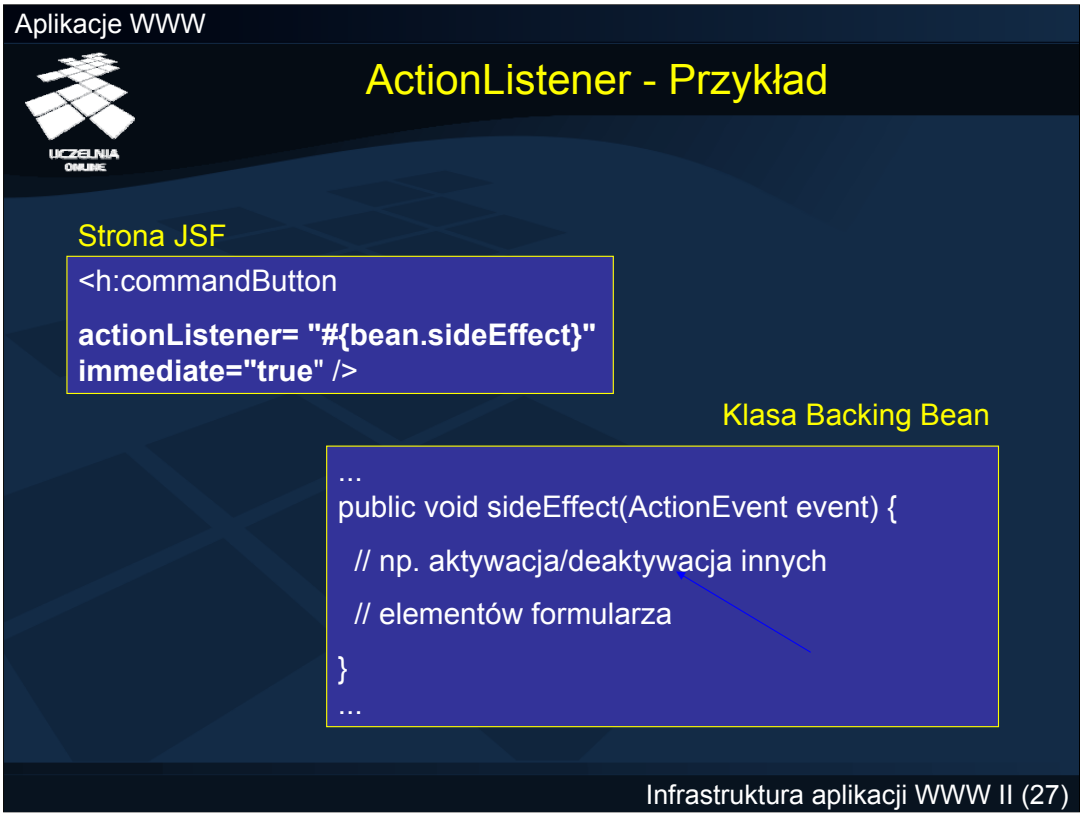
(b) event listeners (pozostałe metody obsługi zdarzeń z interfejsu) - często uruchamiane przed wypełnieniem komponentu Backing Bean danymi i z pominięciem walidacji, niewpływające bezpośrednio na nawigację.




Procedury obsługi zdarzeń event listeners, obsługujące zdarzenia niezwiązane z nawigacją, można podzielić na dwa podtypy:

(a) ActionListener - wywoływany przez przyciski, mapy obrazkowe i linki z kodem JavaScript, czyli elementy, które automatycznie powodują zatwierdzenie formularza;

(b) ValueChangeListener - wywoływany przez listy rozwijane, pola wyboru, grupy radiowe, pola tekstowe, itp. Ponieważ interakcja z tymi elementami formularza wg reguł HTML nie powoduje zatwierdzenia formularza, a jest to konieczne do obsługi zdarzenia, gdyż obsługa ma miejsce po stronie serwera, należy jawnie wymusić zatwierdzenie formularza poprzez kod JavaScript.




ActionListener to metoda obsługująca zdarzenie niepowodujące nawigacji, generowane przez komponent interfejsu powodujący zatwierdzenie formularza – najczęściej przycisk. Metoda taka zwyczajowo jest definiowana w komponencie Backing Bean związanym ze stroną. Metoda ActionListener jest wiązana z komponentem poprzez atrybut ACTIONLISTENER jego znacznika. Przypomnijmy, że metoda akcji, powodująca nawigację jest wskazywana poprzez atrybut ACTION.

Często dla komponentu akcji, np. przycisku, z którym związana jest metoda obsługi zdarzenia niepowodującego nawigacji ustawiana jest wartość „true” dla atrybutu IMMEDIATE (domyślnie „false”), oznaczająca, że obsługa zdarzenia generowanego przez komponent ma mieć miejsce przed walidacją danych wprowadzonych do formularza.

Ustawienie atrybutu IMMEDIATE dla przycisku jest też charakterystyczne dla jednego szczególnego zastosowania przycisków generujących zdarzenia nawigacyjne, a mianowicie przycisków służących do anulowania wprowadzonych zmian i powrotu do poprzedniego widoku (przycisk „Anuluj”). W tym wypadku walidacja danych, z których wprowadzenia użytkownik chce się wycofać nie ma sensu.

# ValueChangeListener – Przykład (1/2)

- Pole wyboru uaktywniające przycisk zatwierdzający formularz logowania


login.jsp

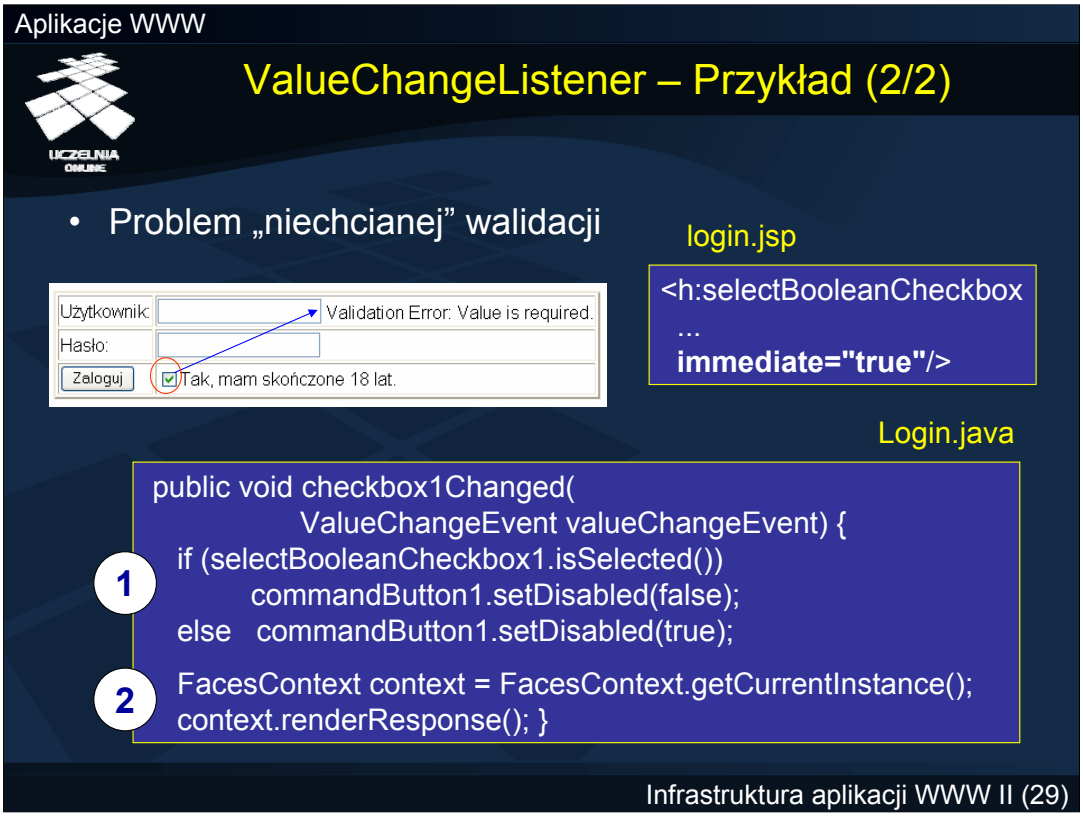
```
<h:commandButton value="Zaloguj"  ...  disabled=”true” />

<h:selectBooleanCheckbox
  binding="#{loginBean.selectBooleanCheckbox1}"
  id="selectBooleanCheckbox"
  valueChangeListener="#{loginBean.checkbox1Changed}"
  onchange="submit()"/>
```

1 2

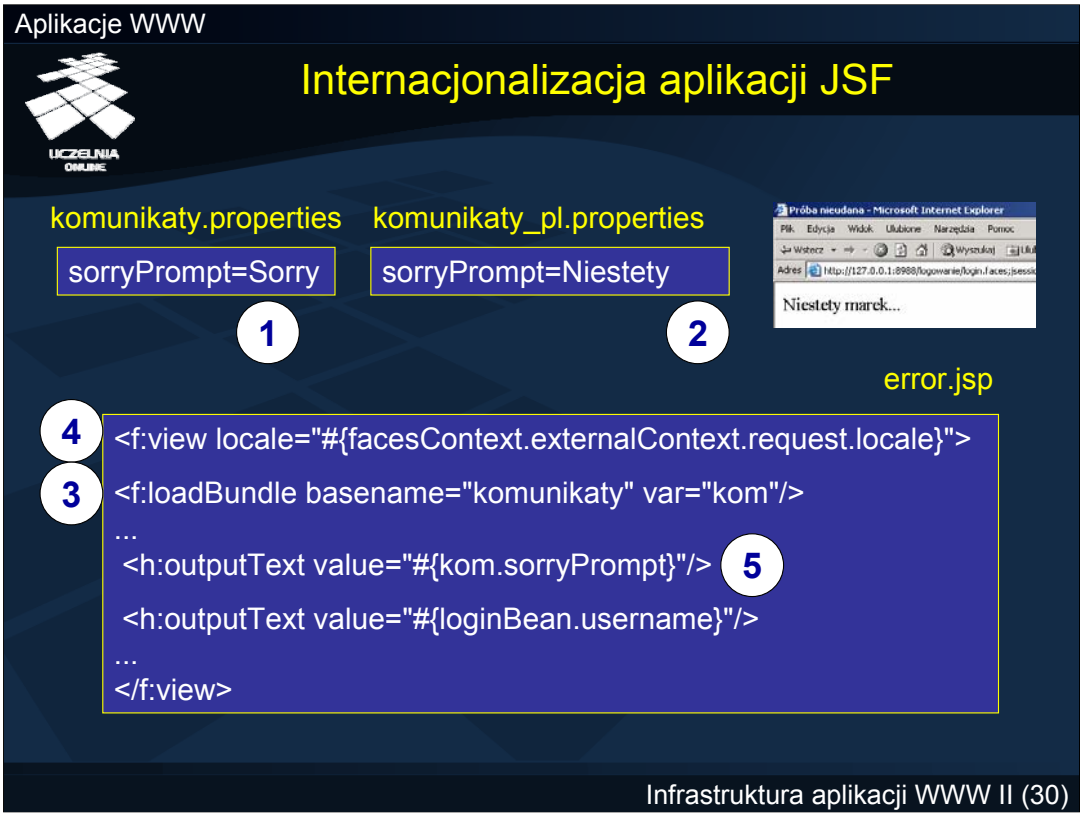


Slajd ilustruje obsługę zdarzeń generowanych przez komponenty, które automatycznie nie zatwierdzają formularza, na przykładzie rozbudowanego formularza logowania. Przycisk zatwierdzający formularz został uczyniony nieaktywnym poprzez ustawienie atrybutu DISABLED na „true” i zostało dodane pole wyboru, którego zaznaczenie aktywuje przycisk. Aktywacja przycisku odbywa się w metodzie zarejestrowanej jako ValueChangeListener dla pola wyboru (1) (kod metody na następnym slajdzie). Aby po zmianie stanu pola wyboru następowało uruchomienie metody obsługującej zdarzenie po stronie serwera, konieczne było sprawienie, aby zmiana stanu pola wyboru powodowała zatwierdzenie formularza. Efekt ten został osiągnięty poprzez kod JavaScript związany ze zdarzeniem onchange (2) (alternatywnie można było wykorzystać zdarzenie onclick).




Działanie metody obsługującej zdarzenie zmiany stanu przycisku polega na programowym modyfikowaniu wartości atrybutu DISABLED przycisku metodą setDisabled() zależnie od stanu pola wyboru (1). Ze względu na sposób przetwarzania żądań przez JSF konieczne jest jednak dodanie na końcu ciała metody kodu rozwiązującego problem „niechcianej” walidacji zilustrowany na slajdzie.

Konieczność zatwierdzenia formularza w celu wywołania metody obsługującej zdarzenie zmiany stanu pola wyboru stanowi problem, gdy z pozostałymi komponentami do wprowadzania danych zostały związane walidatory, tak jak w naszym przykładzie. Nie można wymagać od użytkownika wypełnienia wszystkich pól formularza przed zmianą stanu pola wyboru, który logicznie nie kończy pracy z formularzem. Problem niechcianej walidacji należy rozwiązać podobnie jak w poprzednim przykładzie dla przycisku, poprzez ustawienie dla pola wyboru atrybutu IMMEDIATE. Ustawienie atrybutu IMMEDIATE dla komponentu do wprowadzania danych, takiego jak pole wyboru powoduje, że walidacja danych w komponencie i obsługa zdarzeń przez niego generowanych nastąpi przed walidacją pozostałych komponentów formularza. Aby walidacja pozostałych komponentów w ogóle nie została przeprowadzona, należy poprzez odwołanie do kontekstu JSF w metodzie ValueChangeListener metodą renderResponse() skrócić normalne przetwarzanie żądania, pomijając walidację i przechodząc do fazy generacji wynikowego dokumentu dla przeglądarki (2).



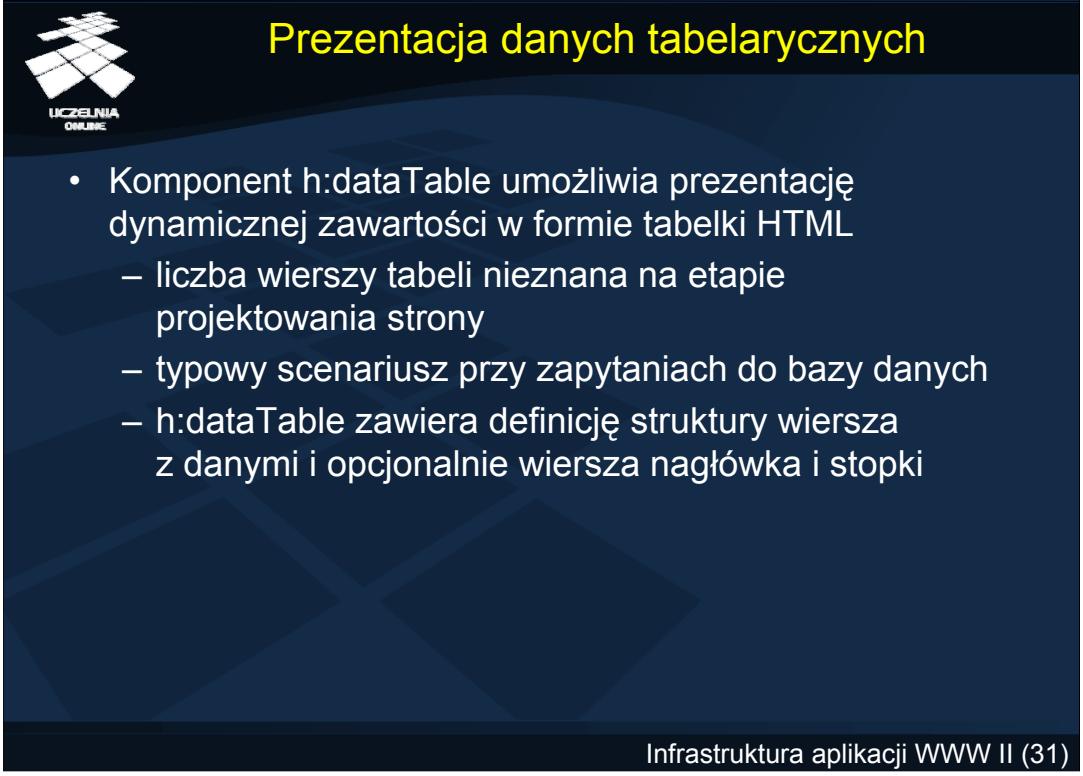

Internacjonalizacja aplikacji JSF jest realizowana poprzez (1) umieszczenie wszystkich komunikatów w zbiorze zasobów (ang. resource bundle) i (2) przygotowanie wersji plików .properties dla wszystkich przewidzianych języków. Pliki .properties to pliki tekstowe zawierające pary klucz=wartość, wiążące klucz komunikatu z jego treścią. Pliki mają kodowanie ASCII, więc polskie znaki muszą być zapisane w postaci kodów Unicode (\uXXXX).

Do ładowania plików .properties na stronach JSF służy znacznik <f:loadBundle> (3), wskazujący nazwę zbioru zasobów (bez kodu języka i rozszerzenia) i wiążący z nim nazwę zmiennej, która będzie reprezentować na stronie zbiór komunikatów.

Odpowiednia wersja językowa komunikatów będzie wybrana na podstawie obowiązujących ustawień lokalizacji. Lokalizację można jawnie ustawić dla strony znacznikiem <f:view> z atrybutem LOCALE. Jako wartość tego atrybutu można podać, tak jak w przykładzie na slajdzie, wyrażenie odczytujące ustawienia lokalizacji zawarte w żądaniu HTTP (4).

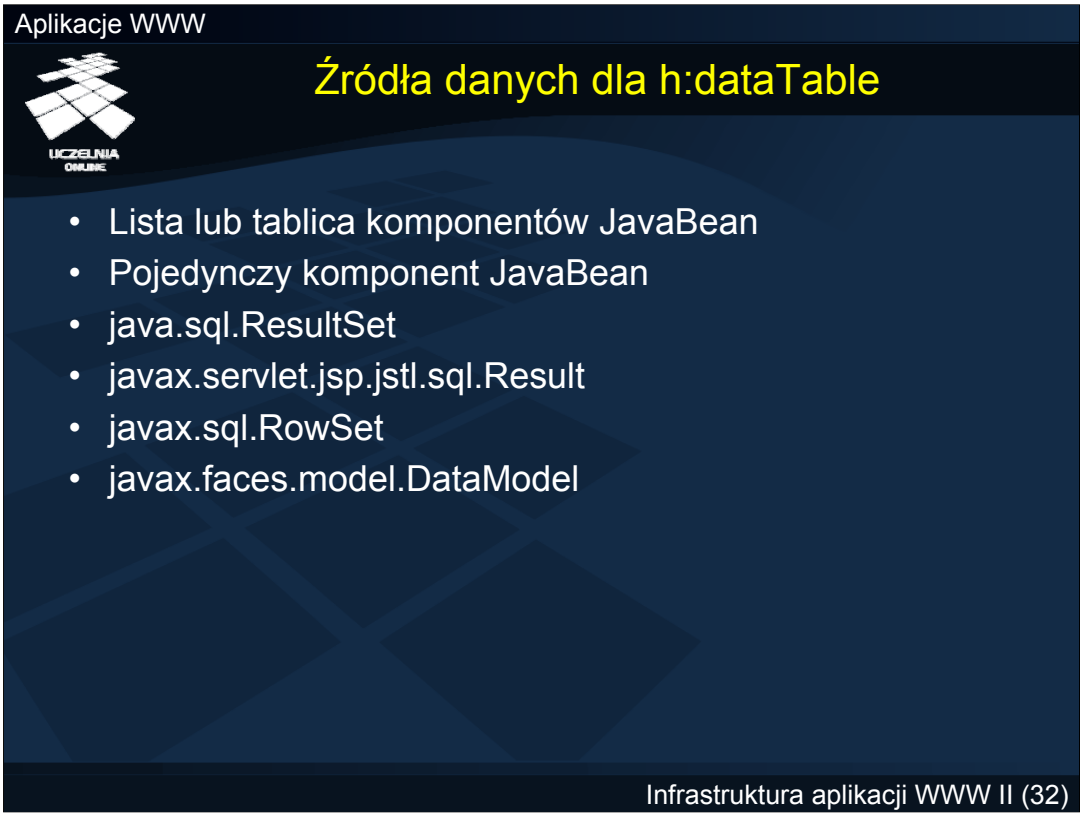
Do wyświetlania komunikatów ze zbiorów zasobów na stronie można wykorzystać znacznik <h:outputText> (5).

Oprócz internacjonalizacji komunikatów aplikacji, możliwa jest też podmiana i internacjonalizacja komunikatów ze standardowych walidatorów, które posiadają predefiniowane klucze. Muszą one być zdefiniowane w wyróżnionym zbiorze zasobów zarejestrowanym w pliku faces-config.xml.




Wśród standardowych komponentów JSF znajduje się komponent reprezentowany w JSP przez znacznik h:dataTable, służący do prezentacji dynamicznej zawartości w formie tabelki HTML. Komponent ten przydaje się w sytuacjach gdy na etapie projektowania liczba elementów, które mają być przedstawione w wierszach tabeli nie jest znana. Taki scenariusz jest typowy dla zapytań do baz danych. Komponent h:dataTable jest więc wygodnym sposobem prezentacji wyników zapytań.

Umieszczając na stronie komponent h:dataTable należy zdefiniować strukturę jednego wiersza z danymi, która będzie powielona odpowiednią liczbę razy w trakcie przetwarzania strony. Opcjonalnie można zdefiniować wiersz nagłówka i stopki tabeli oraz wskazać ile wierszy i począwszy od którego ma być wyświetlonych. Ta ostatnia możliwość ułatwia paginację wyników zapytania.




Źródłem danych dla komponentu h:dataTable może być:

(a) lista lub tablica komponentów JavaBean;

(b) pojedynczy komponent JavaBean;

(c) zbiór wynikowy JDBC java.sql.ResultSet

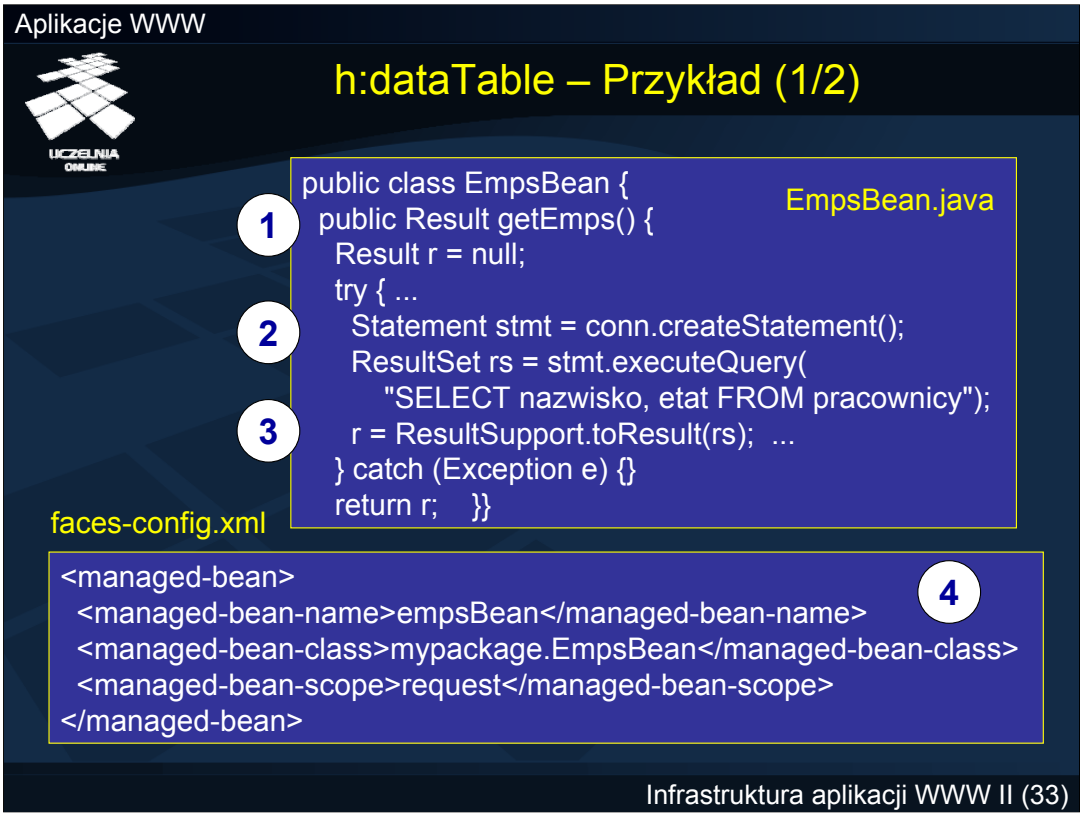
(d) zbiór wynikowy javax.servlet.jsp.jstl.sql.Result

(e) zbiór rekordów javax.sql.RowSet;

(f) obiekt javax.faces.model.DataModel.

Komentarza wymagają trzy ostatnie możliwości. Obiekty Result i RowSet to alternatywa dla ResultSet. Oba reprezentują zbiór rekordów odczytanych z bazy danych i oba w przeciwieństwie do ResultSet mogą zachować dane po zamknięciu połączenia z bazą danych.

Ostatnia możliwość to jawne dostarczenie gotowego obiektu jednego z podtypów DataModel. Tak naprawdę wszystkie źródła danych dla h:dataTable muszą być jawnie lub niejawnie opakowane w DataModel. Dla wcześniej wymienionych możliwości dzieje się to niejawnie.




Ten i następny slajd pokazują przykład wykorzystania komponentu h:dataTable do prezentacji wyników zapytania do bazy danych w formie tabelki HTML. Zapytanie odczytuje nazwiska i etaty pracowników z tabeli PRACOWNICY. Niniejszy slajd pokazuje klasę komponentu JavaBean, której zadaniem jest udostępnienie obiektu Result z wynikami zapytania i fragment pliku faces-config.xml rejestrujący obiekt tej klasy jako zarządzany komponent JavaBean.

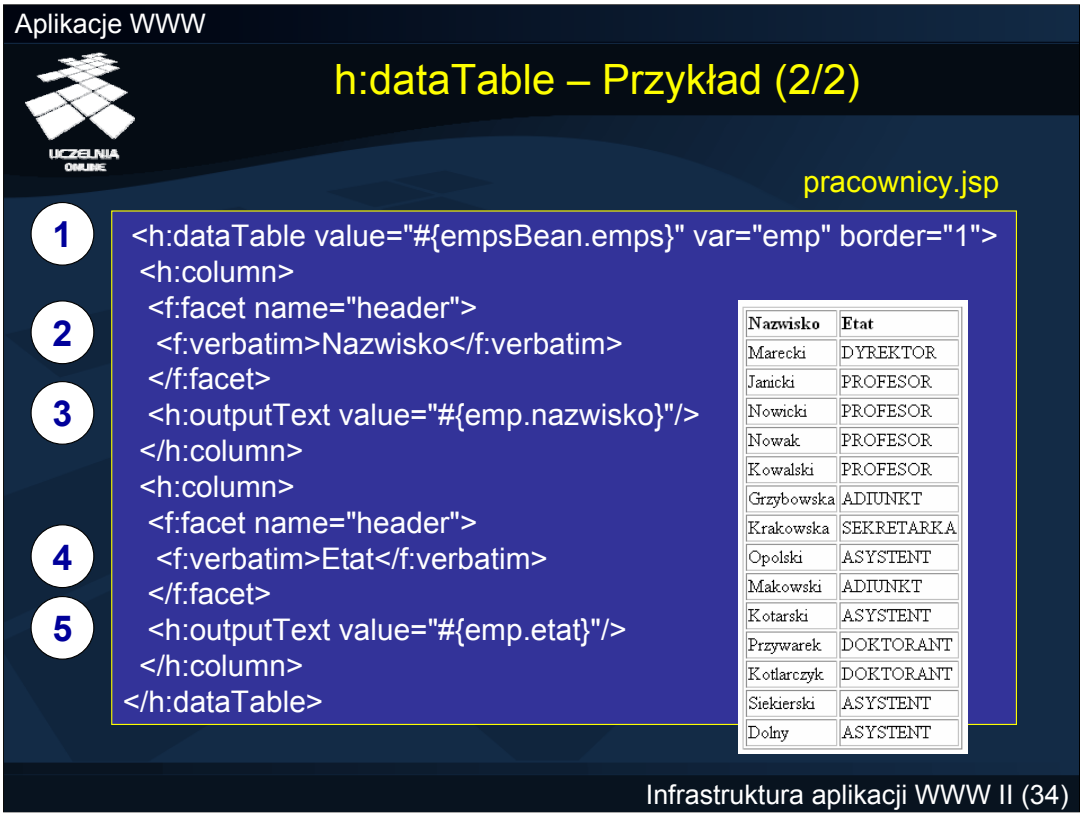
Znaczenie wyróżnionych fragmentów kodu jest następujące:

1. Obiekt Result z wynikami zapytania zwraca metoda klasy EmpBean o nazwie getEmps(), dzięki temu obiekt JavaBean tej klasy będzie udostępniał wyniki zapytania poprzez właściwość „emps”.

2. Wykonanie polecenia odczytującego nazwiska i etaty pracowników poprzez JDBC. Kod otwierający i zamykający połączenie z bazą danych został na slajdzie zastąpiony wielokropkami.

3. Wynik zapytania uzyskany w postaci obiektu ResultSet jest konwertowany do Result za pomocą pomocniczej klasy bibliotecznej JSTL ResultSupport. Celem konwersji jest uzyskanie obiektu zawierającego wyniki zapytania, niewymagającego utrzymywania otwartego połączenia z bazą danych.

4. Rejestracja obiektu klasy EmpsBean jako zarządzanego przez JSF komponentu JavaBean o nazwie „empsBean”. Dzięki temu, JSF utworzy obiekt udostępniający dane z bazy danych, gdy zostanie uruchomiona strona z niego korzystająca.




Slajd pokazuje stronę JSP wykorzystującą komponent h:dataTable do wyświetlenia listy nazwisk i etatów pracowników pobranych z bazy danych za pośrednictwem zarządzanego obiektu JavaBean. Znaczenie wyróżnionych fragmentów kodu jest następujące:

1. Jako źródło danych dla komponentu h:dataTable wskazana jest właściwość „emps” komponentu JavaBean „empsBean”. Kolejne elementy kolekcji obiektów zwrócone przez źródło danych będą reprezentowane przez zmienną „emp”.

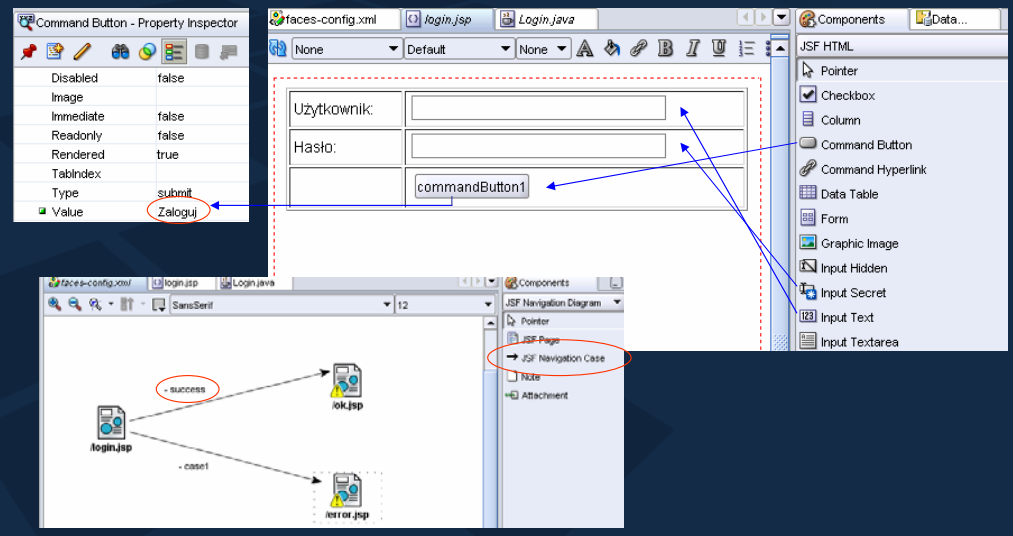
2. Definicja nagłówka pierwszej kolumny, zawierającej nazwiska, za pomocą znacznika f:facet. Zawartość inna niż znaczniki JSF musi być ujęta w element f:verbatim, aby była wyświetlona we właściwym miejscu.

3. Wskazanie właściwości „nazwisko” elementu prezentowanej kolekcji jako wartości w kolumnie.

4. Definicja nagłówka drugiej kolumny, zawierającej etaty, za pomocą znacznika f:facet.

5. Wskazanie właściwości „etat” elementu prezentowanej kolekcji jako wartości w kolumnie.

# JSF w środowisku IDE (1/2)





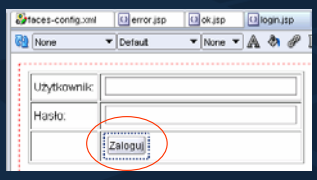

Podobnie jak Web Forms w ASP.NET, technologia JSF została opracowana z myślą o tworzeniu aplikacji w trybie wizualno-zdarzeniowym w zintegrowanych środowiskach typu IDE. Oczywiście tworzenie aplikacji JSF jest możliwe za pomocą edytora tekstowego i kompilatora Java wywoływanego z linii poleceń. Rolą środowiska IDE jest zwiększenie produktywności programisty poprzez wizualną edycję pliku konfiguracyjnego aplikacji i stron widoków oraz generację szkieletów klas i metod.

Inaczej niż w przypadku ASP.NET, dla JSF i ogólnie dla języka Java nie można wskazać jednego oficjalnego środowiska IDE, używanego przez większość twórców aplikacji. Wybór między środowiskami takimi jak Net Beans, Eclipse, czy Oracle JDeveloper zależy od indywidualnych preferencji i przyzwyczajeń.

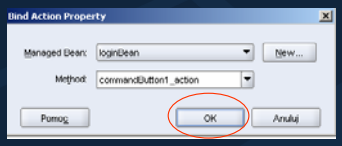
Slajd ilustruje wsparcie dla wizualnego tworzenia aplikacji JSF na przykładzie środowiska Oracle JDeveloper 10g 10.1.3. U góry pokazany jest sposób wizualnej edycji stron JSF poprzez umieszczanie poszczególnych komponentów na stronie techniką drag-and-drop. Właściwości poszczególnych komponentów można następnie zmodyfikować poprzez paletę właściwości. Środowisko na bieżąco uaktualnia kod źródłowy strony i projektant zawsze ma możliwość przełączenia się w tryb edycji kodu.

U dołu pokazany jest sposób wizualnej edycji reguł nawigacji i przepływu sterowania między stronami. Na bieżąco uaktualniana jest zawartość pliku konfiguracyjnego faces-config.xml.

# JSF w środowisku IDE (2/2)




Dwuklik
na przycisku




Login.java

```
...
public String commandButton1_action()
{
        // Add event code here...
        return null;
}
...
```



Slajd ilustruje wsparcie środowiska Oracle JDeveloper dla implementacji procedur obsługi zdarzeń zachodzących w wyniku interakcji użytkownika z interfejsem aplikacji JSF. W przypadku oprogramowywania domyślnego zdarzenia dla komponentu, jak w przykładzie na slajdzie - zdarzenia naciśnięcia przycisku, wystarczy podwójne kliknięcie na komponencie w wizualnym edytorze strony. Następnie środowisko poprosi o potwierdzenie lub zmianę zaproponowanej nazwy metody obsługującej zdarzenie i utworzy szkielet metody w klasie Backing Bean związanej ze stroną.


## Podsumowanie

- JSF to technologia tworzenia interfejsu użytkownika pracującego po stronie serwera w aplikacjach Java EE
- Nadzieja dla programistów Java EE na dorównanie .NET w zakresie prostoty tworzenia interfejsu użytkownika
- W pewnym stopniu alternatywa dla Struts
- Oparta o architekturę MVC oraz stanowe, rozszerzalne i konfigurowalne komponenty interfejsu
- Podstawowym środowiskiem do renderowania komponentów jest JSP



JSF to technologia tworzenia interfejsu użytkownika pracującego po stronie serwera w aplikacjach Java EE. JSF kładzie nacisk na stanowy interfejs użytkownika, ale wspiera twórców aplikacji również w zakresie obsługi nawigacji, walidacji danych i tworzenia aplikacji wielojęzycznych. JSF daje programistom Java EE nadzieję na dorównanie .NET w zakresie prostoty tworzenia interfejsu użytkownika.

W pewnym stopniu JSF stanowi alternatywę dla Struts, ale JSF i Struts mogą też współpracować. Podobnie jak Struts, JSF są oparte o architekturę MVC. Nową jakością w JSF są stanowe, rozszerzalne i konfigurowalne komponenty interfejsu. JSF z założenia ma wspierać różne technologie wizualizacji komponentów, ale podstawowym, w sposób szczególny wspieranym środowiskiem do renderowania komponentów jest JSP.


## Materiały dodatkowe

- The Java EE 5 Tutorial, http://java.sun.com/javaee/5/docs/tutorial/doc/



•The Java EE 5 Tutorial, http://java.sun.com/javaee/5/docs/tutorial/doc/