Podstawy technologi JavaServer Faces

wg

http://docs.oracle.com/javaee/6/tutorial/doc/bnaph.html

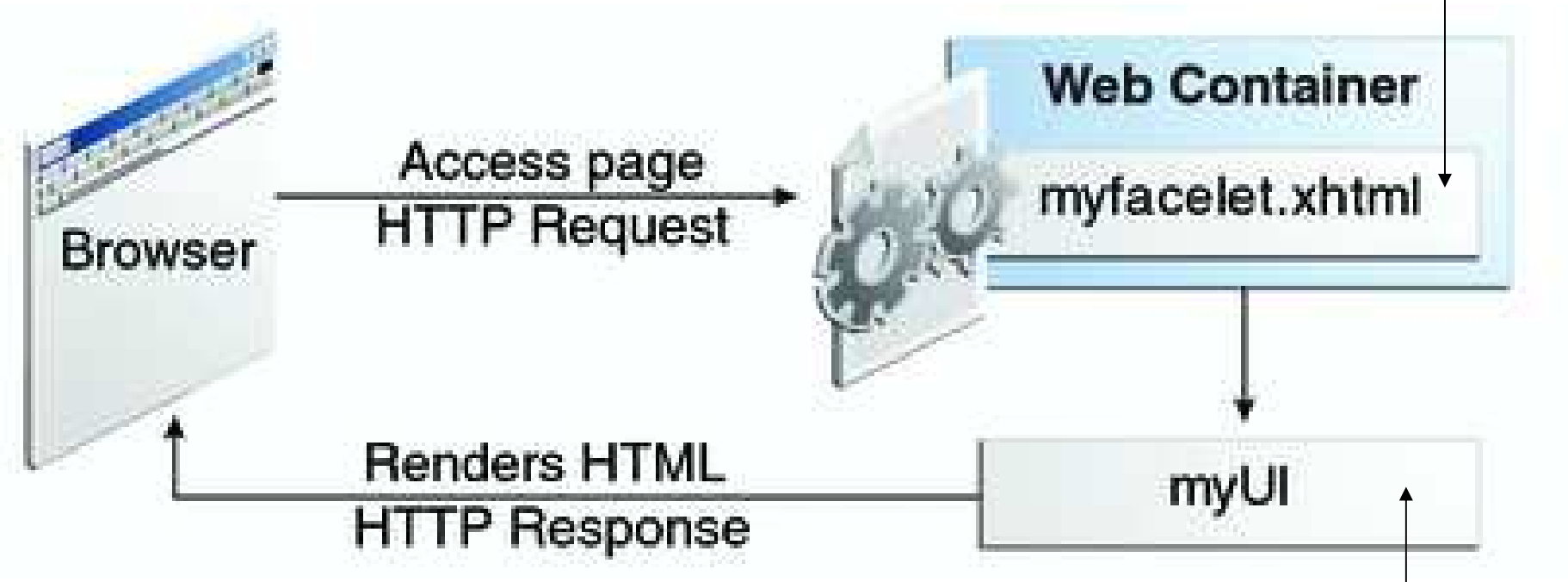
Programowanie komponentowe 3

Z czego składa się technologia JavaServer Faces ?

- interfejsu programowania aplikacji internetowej reprezentujący komponenty i zarządzanie ich stanem; obsługi zdarzeń, walidacji po stronie serwera, konwersji danych; nawigacji stron; wspieranie intenacjonalizacji i dostępności; zapewnienia rozszerzalność wszystkim tym elementom
- biblioteki znaczników do dodawania komponentów do stron internetowych i obiektów po stronie serwera

Interakcja żądanie - odpowiedź między warstwą klienta i warstwą internetową w typowej aplikacji JavaServer Faces

znaczniki komponentów JavaServer Faces, odniesienia do słuchaczy zdarzeń, walidatorów oraz konwerterów, komponentów JavaBeans, pozyskujące i przetwarzające dane specyficznie dla komponentów



Komponenty JavaServer Faces używane w widokach (UI)

Przebieg fazy: żądanie - odpowiedź

Strona www, myfacelet.xhtml, **jest zbudowana ze znaczników komponentów JavaServer Faces**.

Rola znaczników:

- łączą widok z **komponentami widoku** (reprezentowanymi przez myUI na rysunku), które są po stronie serwera, w warstwie internetowej
- łączą ze **słuchaczami zdarzeń, walidatorami oraz konwerterami**, które są reprezentowane przez komponenty
- łączą z **komponentami JavaBeans**, pozyskującymi i przetwarzającymi dane specyficznie dla komponentów

Skutkiem żądania z warstwy klienta (**Request**), widok jest renderowany jako odpowiedź (**Response**).
Renderowanie jest procesem, w którym na podstawie zawartości strony kontener internetowy tworzy strony typu HTML lub XHTML, które mogą być odczytywane przez warstwę klienta zawierającą przeglądarkę.

Technologia JavaServer Faces wspiera budowę aplikacji wielowarstwowej

- **Separacja zachowania i prezentacji** w aplikacji internetowej dzięki odwzorowaniu żądania HTTP na specyficzną dla komponentu obsługę zdarzeń oraz zarządzanie komponentami jako obiektami o określonym czasie życia (stateful) po stronie serwera
- **Separacja logiki biznesowej od prezentacji** pozwala programistom stron internetowych posługiwać się jedynie językiem znaczników bez konieczności używania języka proceduralnego Java
- **Możliwość zastosowania różnych implementacji języków skryptowych** dzięki używaniu API technologii JSF bezpośrednio przez API Java Servlet.

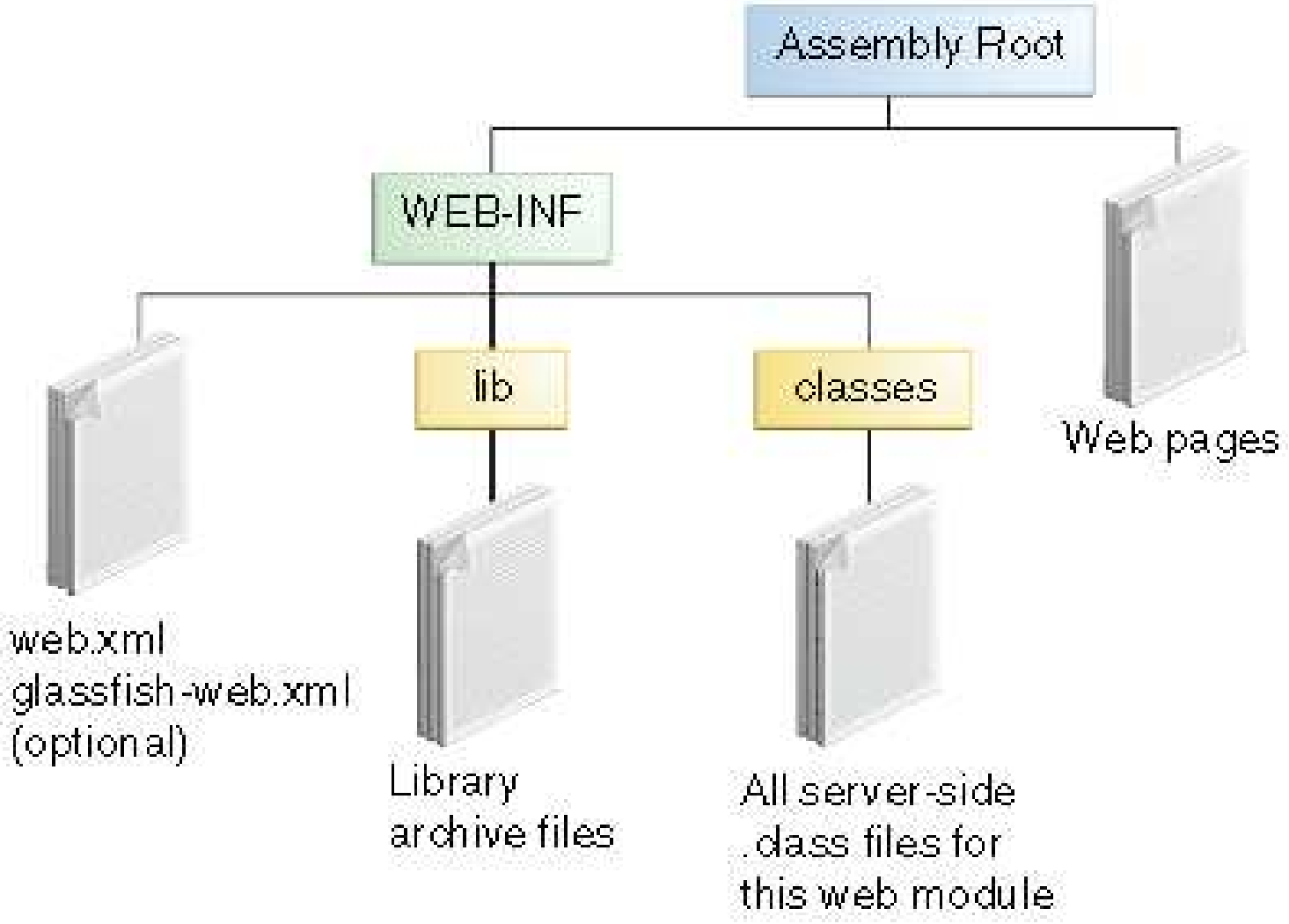
Zadania wykonywane podczas tworzenia aplikacji internetowej

- Utworzenie strony internetowej typu JSF
- Wstawienie komponentów do strony intenetowej za pomocą wstawienia ich znaczników
- Powiązanie komponentu na stronie internetowej do danych po stronie serwera
- Powiązanie zdarzeń generowanych przez komponenty do kodu aplikacji po stronie serwera
- Zachowanie i odtwarzanie stanu aplikacji podczas cyklu życia żądania wysłanego do serwera
- Ponowne wykorzystanie komponentów i rozszerzanie ich własności

Co zawiera aplikacja typu JavaServer Faces

- **Zbiór stron internetowych** zawierających znaczniki komponentów
- **Zbiór znaczników umożliwiających umieszczenie komponentów na stronie internetowej**
- **Zbiór obiektów typu Managed Bean,** które są obiektami zarządzanymi przez kontener internetowy, z minimalnymi wymaganiami. Wspierają one injekcję zasobów oraz akcje występujące w cyklu życia żądanie-odpowiedź
- **Deskryptor wdrożenia web.xml**
- **Opcjonalnie, pliki konfiguracji zasobów aplikacji np. faces-config.xml:** reguły nawigacji stron internetowych, konfiguracja ziaren oraz niestandardowych obiektów i komponentów,
- **Opcjonalnie, zbiór obiektów niestandardowych,** które zawierają komponenty niestandardowe, walidatory, konwertery lub słuchacze zdarzeń, tworzone przez programistów
- **Opcjonalnie, zbiór znaczników niestandardowych** reprezentujących obiekty niestandardowe na stronie

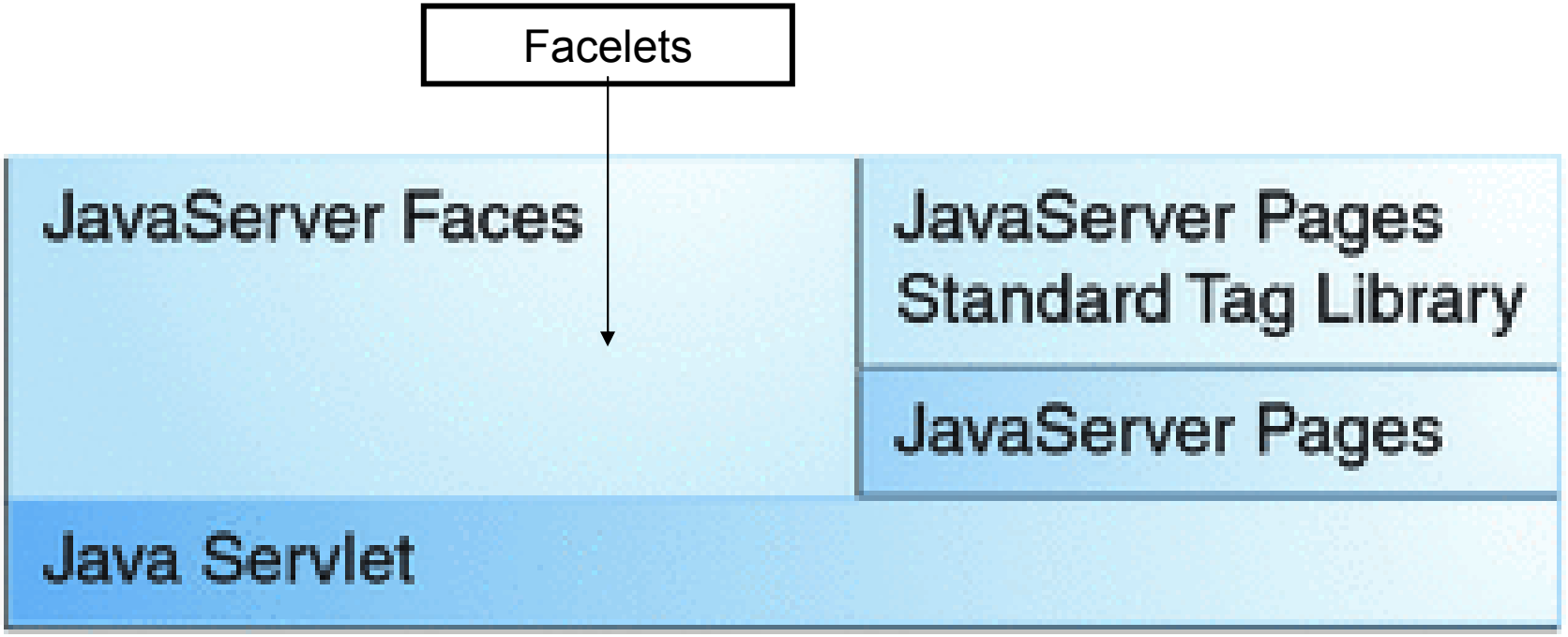
Struktura modułu internetowego



Język znaczników Facelets

- Użycie XHTML do tworzenia strony internetowej
- Korzystanie z biblioteki znaczników Facelets, JavaServer Faces i JSTL
- Korzystanie z języka Expression Language (EL)
- **Elementy wspierające budowę dużej aplikacji:**
  - Stosowanie szablonów komponentów i stron umożliwia wielokrotne używanie kodu
  - Funkcjonalne rozszerzanie właściwości komponentów i obiektów po stronie serwera
  - Krótki czas kompilacji
  - Walidacja wyrażeń języka EL podczas kompilacji
  - Wysoka wydajność renderowania stron aplikacji

Warstwowa struktura API



Cechy języka znaczników Facelets

- **Wielożywalność kodu** dzięki zastosowaniu szablonów i możliwości tworzenia komponentów kompozytowych
- **Zastowanie adnotacji JavaServer Faces** umożliwia rejestrowanie komponentów typu Managed Bean jako zasobów aplikacji JSF. Dzięki niejawnej nawigacji między stronami ograniczono jawne definiowanie nawigacji między stronami
- **JavaSever Faces dostarcza bogatą architekturę** do zarządzania komponentami, przetwarzania danych komponentów, walidacji danych użytkownika aplikacji oraz obsługi zdarzeń

Opis znaczników obsługiwanych przez Facelets

- Znaczniki typu UI
- Znaczniki do tworzenia szablonów

Biblioteki znaczników obsługiwanych przez Facelets

Biblioteki znaczników	URI	Prefiks	Przykład	Zawartość
JavaServer Faces Facelets Tag Library	http://java.sun.com /jsf/facelets	ui:	ui:component ui:insert	Znaczniki szablonów
JavaServer Faces HTML Tag Library	http://java.sun.com /jsf/html	h:	h:head h:body h:outputText h:inputText	Znaczniki komponentów JavaServer Faces dla wszystkich obiektów komponentów UI
JavaServer Faces Core Tag Library	http://java.sun.com /jsf/core	f:	f:actionListener f:attribute	Znaczniki niestandardowych akcji JavaServer Faces, które są niezależne od narzędzia renderowania
JSTL Core Tag Library	http://java.sun.com /jsp/jstl/core	c:	f:actionListener f:attribute	JSTL 1.2 Core Tags
JSTL Functions Tag Library	http://java.sun.com /jsp/jstl/functions	fn:	fn:toUpperCase fn:toLowerCase	JSTL 1.2 Functions Tags

Opis znaczników obsługiwanych przez Facelets (tutorial EE 6)

- Znaczniki typu UI

Przegląd znaczników JSF (UI) (xmlns:h="http://java.sun.com/jsf/html")

Znacznik	Funkcja	Renderowany jako	Widok
h:column	Reprezentuje kolumnę danych w komponencie danych	Kolumna w tabeli HTML	Kolumna w tabeli
h:commandButton	Przesyła dane z formularza do aplikacji	Element HTML <input type=*type*> gdzie wartość type może być: "submit", "reset", lub "image"	Przycisk
h:commandLink	Łączy z inną stroną lub z innym miejscem danej strony	Element HTML <a href>	Hyperlink
h:dataTable	Reprezentuje widok zbioru złożonych danych	Element HTML <table>	Tablica modyfikowana dynamicznie
h:form	Reprezentuje formularz danych wejściowych, które mogą być przesłane razem do aplikacji	Element HTML <form>	Brak widoku

Znacznik	Funkcja	Renderowany jako	Widok
h:graphicImage	Wyświetla obraz	Element HTML <img>	Obraz
h:inputHidden	Pozwala autorowi strony na ukrycie elementu strony	Element HTML <input type="hidden">	Brak widoku
h:inputSecret	Pozwala użytkownikowi wprowadzić łańcuch bez pokazania jego zawartości	Element HTML <input type="password">	Pole tekstowe zawierające znaki maskujące aktualną zawartość wprowadzonego łańcucha
h:inputText	Pozwala wprowadzać dane	Element HTML <input type="text">	Pole tekstowe
h:inputTextarea	Pozwala wprowadzać łańcuch wielowierszowy	Element HTML <textarea>	Pole wielowierszowe
h:message	Wyświetla komunikat	Znacznik HTML jeśli zastosowano styl	Łańcuch tekstu
h:messages	Wyświetla komunikaty	Zbiór znaczników HTML , jeśli zastosowano styl	Łańcuch tekstu

Znacznik	Funkcja	Renderowany jako	Widok
h:outputFormat	Wyświetla komunikat	Zwykły tekst	Zwykły tekst
h:outputLabel	Wyświetla zagnieżdżony komponent jako etykieta podanego pola wejściowego	Element HTML <label>	Zwykły tekst
h:outputLink	Łączy z inna stroną lub położeniem na stronie bez generowania zdarzenia	Element HTML <a>	Hyperlink
h:outputText	Wyświetla linię tekstu	Zwykły tekst	Zwykły tekst
h:panelGrid	Wyświetla tabelę	Elementy HTML: <table> z <tr> i <td>	Tabela
h:panelGroup	Grupuje komponenty za pomocą jednego nadrzędnego komponentu	Element HTML lub	Wiersz tabeli
h:selectBooleanCheckbox	Pozwala zmienić wartość za pomocą wyboru typu Boolean	Element. HTML <input type="checkbox">	Przycisk wyboru

Znacznik	Funkcja	Renderowany	Widok
h:selectManyCheckbox	Wyświetla zbiór przycisków wyboru, gdzie można dokonać wielokrotnego wyboru	Zbiór elementów HTML <input> typu przycisk wyboru	Zbiór przycisków wyboru
h:selectManyListbox	Wyświetla listę, gdzie można dokonać wyboru wielu pozycji listy	Element HTML <select>	lista
h:selectManyMenu	Wyświetla listę menu, gdzie można dokonać wyboru wielu elementów menu	Element HTML <select>	Przewijany widok typu ComboBox
h:selectOneListbox	Wyświetla listę, gdzie można dokonać wyboru jednej pozycji listy	Element HTML <select>	lista
h:selectOneMenu	Wyświetla listę menu, gdzie można dokonać wyboru jednego elementu menu	Element HTML <select>	Przewijany widok typu ComboBox
h:selectOneRadio	Wyświetla zbiór przycisków, gdzie można dokonać wyboru jednego przycisku	Element HTML <input type="radio">	Zbiór przycisków typu „radio”

Najczęściej występujące atrybuty w znacznikach komponentów

Atrybut	Opis
binding	Identyfikuje właściwość obiektu (atrybut i jego metody typu set i get lub same metody) i binduje do niej komponent
id	Identyfikuje komponent
immediate	Jeśli ma wartość true, zdarzenia, walidacje i konwersje związane z komponetem powinny się odbywać, kiedy rozpoczyna się obsługa fazy żądania
rendered	Specyfikuje warunek renderowania komponentu. Jeśli warunek nie jest spełniony, komponent nie jest renderowany.
style	Specyfikuje Kaskadowy arkusz stylu (CSS) znacznika np. **style="border:solid 1px"**
styleClass	Specyfikuje nazwę klasę CSS, która zawiera definicję stylów np. klasa **.center_content** z pliku **cssLayout.css**
value	Specyfikuje wartość komponentu w postaci wyrażenia

Atrybut id

Używany w przypadku powiązania z innym komponentem lub klasą po stronie sewera. W przypadku braku deklaracji takiego atrybutu, implementcja JSF generuje automatycznie atrybut id.

Atrybut immediate

Komponenty wejściowe i komponenty generujące polecenia (implementujące interfejs javax.faces.component.ActionSource np. przyciski, hyperlinki) przy wartości atrybutu true mogą obsługiwać zdarzenia, walidację i konwersję, na początku cyklu życia JSF. Należy ustawić we wszystkich komponentach ten atrybut na true, jeśli są powiązane logicznie podczas obsługi tych zdarzeń.

Atrybut rendered

Wartość tego atrybutu decyduje o możliwości umieszczenia widoku danego komponentu na stronie zwracanej do przeglądarki.

Atrybuty style i styleClass

Umożliwiają specyfikację kaskadowego arkusza stylu (http://www.w3.org/Style/CSS/.)

Atrybuty value i binding

Atrybuty te wiążą komponent z danymi obiektowymi

(1) Dodawanie wybranych znaczników do strony

Znaczniki html i body

Definicje strony xhtml w technologii JSF:

```
<!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">
    <h:head> Add a title
    </h:head>
    <h:body> Add Content
    </h:body>
</html>
```

Strona xhtml po renderowaniu

```
<!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">
    <head>
        <title>Add a title</title>
    </head>
      <body> Add Content
      </body>
   </html>
```

(2) Dodawanie wybranych znaczników do strony

Dodawanie komponentu Form

```
<h:form>
  inne znaczniki reprezentujące elementy fiormularz
</h:form>
```

Zastosowanie komponentów typu text

- Etykiety (labels)
- Pola tekstowe
- Obszary tekstowe
- Pole tekstowe hasła maskujące wprowadzane litery

(3) Dodawanie wybranych znaczników do strony

Znaczniki pól wejściowych textowych

- **h:inputHidden**
- **h:inputSecret**
- **h:inputText**
- **h:inputTextarea**

Atrybuty znaczników typu input	**Opis**
converter	Określa typ konwertera wartości wprowadzonej do komponentu lub wyprowadzania
converterMessage	Definicja wiadomości określającej błędy konwersji
dir	Specyfikuje kierunek wyświetlania tekstu LTR (od do prawej) i RTL (od p;rawej do lewej)
label	Specyfikacja nazwy komponentu używana w wiadomościach o błędach

(4) Dodawanie wybranych znaczników do strony

Atrybuty znaczników typu input cd.	**Opis**
lang	Specyfikuje kod związany z podanym językiem np. pl
required	Wartośc true oznacza konieczność wprowadzenia danej do pola komponentu (value)
requiredMessage	Definicja wiadomości określającej brak danej
validator	Identyfikuje metodę ziarna typu Managed Bean do obsługi walidacji danego komponentu (inny sposób: zagnieżdżenie znacznika **f:validator**)
validatorMessage	Definicja wiadomości określającej brak poprawnego wyniku walidacji
valueChangeListener	Identyfikuje metodę, która obsługuje zdarzenie wprowadzenia danej do komponentu

(5) Dodawanie wybranych znaczników do strony

Przykłady znaczników typu input wykorzystujących walidację

```
<h:inputText id="inputGuess"
  value="#{userNumberBean.userNumber}"
  required="true" size="3"
   disabled="#{userNumberBean.number eq userNumberBean.userNumber}"
   validator="#{userNumberBean.validateNumberRange}">
</h:inputText>
```

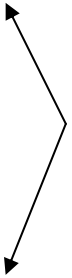
Metoda w klasie typu Managed Bean do walidacji danych wprowadzanych za pomocą h:inputText

```
public void validateNumberRange(FacesContext context,
                                UIComponent toValidate, Object value)
{ if (remainingGuesses <= 0)
  { FacesMessage message = new FacesMessage("No guesses left!");
    context.addMessage(toValidate.getClientId(context), message);
     ((UIInput) toValidate).setValid(false);  //wynik negatywny walidacji
        return; }
     int input = (Integer) value;
     if (input < minimum || input > maximum)
        { ((UIInput) toValidate).setValid(false);  //wynik negatywny walidacji
          FacesMessage message = new FacesMessage("Invalid guess");
          context.addMessage(toValidate.getClientId(context), message);
} }
```

(6) Dodawanie wybranych znaczników do strony

Przykłady znaczników typu input korzystających z obsługi zdarzeń

```
<h:inputText id="name" size="30"
   value="#{cashier.name}"
   required="true"
   valueChangeListener="#{cashier.processValueChange}">
</h:inputText>
```



Metoda w klasie typu Managed Bean do obsługi zdarzeń generowanych przez widoki komponentów stron JSF

```
public void processValueChange(ValueChangeEvent event)
                                    throws AbortProcessingException
  { if (null != event.getNewValue())
     { FacesContext.getCurrentInstance().getExternalContext().
          getSessionMap().put("name", event.getNewValue());
      }
   }
```

(7) Dodawanie wybranych znaczników do strony

Zastosowanie znacznika h:inputText do renderowania Text Field

```
<h:inputText
    id="name"
    label="Customer Name"
    size="30"
    value="#{cashier.name}"
    required="true"
    requiredMessage="#{bundle.ReqCustomerName}">
 <f:valueChangeListener type="dukesbookstore.listeners.NameChanged" />
</h:inputText>
```

Zastosowanie znacznika h:inputSecret do renderowania Text Field

Znacznik umożliwia maskowanie wprowadzanych znaków za pomocą wartości **false** atrybutu **redisplay**

```
<h:inputSecret redisplay="false"
  value="#{LoginBean.password}" />
```

(8) Dodawanie wybranych znaczników do strony

Znaczniki pól wyjściowych

- **h:outputFormat**
- **h:outputLabel**
- **h:outputLink**
- **h:outputText**

Zastosowanie znacznika h:outputLabel do renderowania Label (atrybut value komponentu h:outputText reprezentuje tekst komponentu **h:outputLabel** – można zastąpić go atrybutem **value** tego komponentu)

```
<h:selectBooleanCheckbox id="fanClub"
                         binding="#{cashier.specialOffer}" />
<h:outputLabel for="fanClub" binding="#{cashier.specialOfferText}" >
    <h:outputText id="fanClubLabel"
                  value="#{bundle.DukeFanClub}" />
</h:outputLabel>
```

czyli:

```
<h:selectBooleanCheckbox id="fanClub"
                         binding="#{cashier.specialOffer}" />
<h:outputLabel for="fanClub" binding="#{cashier.specialOfferText}" >
                value="#{bundle.DukeFanClub}" />.
```

(9) Dodawanie wybranych znaczników do strony

Zastosowanie znacznika h:outputLink do renderowanie Hyperlink

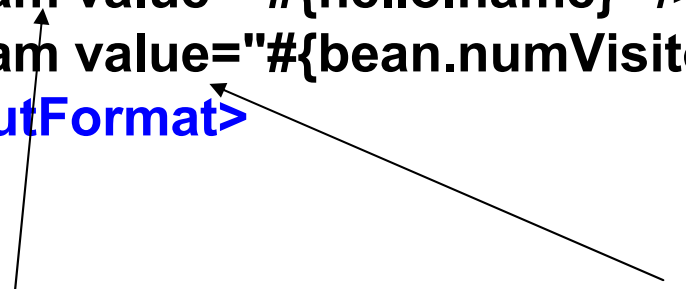
<h:outputLink value="javadocs"> **Documentation for this demo**
</h:outputLink>

Tekst zagnieżdżony wyświetla się jako tekst Hyperlinku na stronie.

Wyświetlanie wiadomości za pomocą znacznika h:outputFormat

Umożliwia wklejanie do komunikatu wartości atrybutów obiektów

<h:outputFormat
value="Hello, **{0}!** You are visitor number **{1}** to the page.">
<f:param value="#{hello.name}" />
<f:param value="#{bean.numVisitor}"/>
</h:outputFormat>



np

Hello, **Bill!** You are visitor number **4** to the page.

(10) Dodawanie wybranych znaczników do strony

Zastosowanie znaczników tworzących komponenty poleceń i nawigacji.

Znaczniki:

h:commandButton jest renderowany jako przycisk i znacznik

h:commandLink jako hyperlink.

Znaczniki te używają atrybuty:

- **action** – łańcuch określajcy wywoływaną metodę od obiektu typu Managed Bean. Metoda zwraca łańuch określający adres strony, która zostaje wywołana. Atrybut może zawierać bezpośrednio adres tej strony
- **actionListener** - wskazanie wywołanej metody od obiektu typu ziarno, obsługującej zdarzenie

Przykład 1

```
<h:commandButton id="powrot"
                 value="#{bundle['jsf.rezultat2.akcja']}"
                 action="/faces/index2"/>
```

Przykład 2

```
<h:commandLink
    id="Duke"
    action="bookstore"
    actionListener="#{actionBean.chooseBookFromLink}"
    value="#{bundle['jsf.dodaj_produkt2.akcja']}" />
```

Atrybut **action** zawiera adres strony, która zostanie wywołana, a jednocześnie zostanie wykonana metoda **chooseBookFromLink** od obiektu typu **actionBean** (atrybut **actionListener**).

Przykład 3 (renderuje JavaScript!)

```
<h:commandLink id="Duke" action="bookstore">
   <f:actionListener
        type="dukesbookstore.listeners.LinkBookChangeListener" />
  <h:outputText value="#{bundle.Book201}"/>
</h:commandLink>
```

Po renderowaniu

```
<a id="_idt16:Duke" href="#"
      onclick="mojarra.jsfcljs(document.getElementById('j_idt16'),
            { 'j_idt16:Duke‘ : 'j_idt16:Duke‘ },‘ '); return false; ”‘>
   My Early Years: Growing Up on Star7, by Duke</a>
```

(11) Dodawanie wybranych znaczników do strony

Dodawanie grafiki za pomocą znacznika h:graphicImage

```
<h:graphicImage id="mapImage" url="/resources/images/book_all.jpg"/>
  lub
<h:graphicImage id="mapImage" name="book_all.jpg"
                library="images" alt="#{bundle.ChooseBook}"
                usemap="#bookMap" />
  lub
<h:graphicImage value="#{resource['images:wave.med.gif']}"/>
```

Równoważna definicja za pomocą arkusza stylu css:

```
header { position: relative; height: 150px; background: #fff
        url(#{resource['img:top-background.jpg']}) repeat-x;
```

Znaczniki określające ułożenie elementów

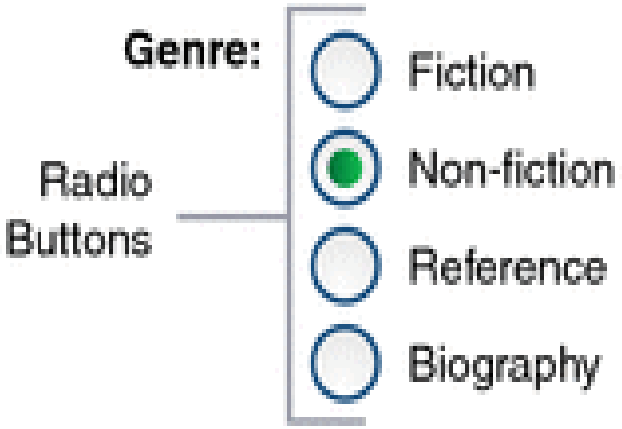
h:panelGrid – atrybuty: columns, columnClasses, footerClass, headerClass, panelClass, rowClasses – wyświetla tabelę
layout – wyświetla wiersze tabeli

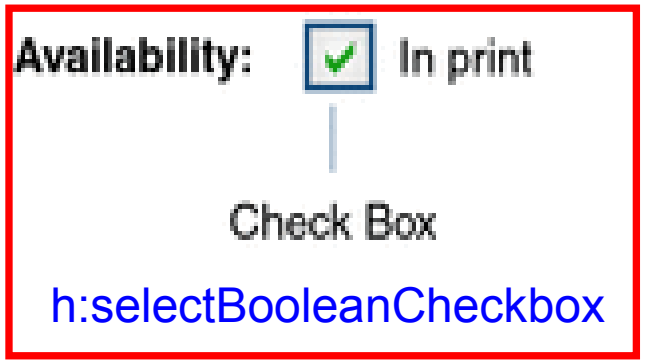
h:panelGroup

```
<h:panelGrid columns="2" headerClass="list-header"
    styleClass="list-background" rowClasses="list-row-even, list-row-odd"
   summary="#{bundle.CustomerInfo}" title="#{bundle.Checkout}">
   <f:facet name="header">
      <h:outputText value="#{bundle.Checkout}"/>
   </f:facet>
   <h:outputLabel for="name" value="#{bundle.Name}" />
   <h:inputText id="name" size="30" value="#{cashier.name}"
      required="true" requiredMessage="#{bundle.ReqCustomerName}">
         <f:valueChangeListener
                  type="dukesbookstore.listeners.NameChanged" />
   </h:inputText>
   <h:message styleClass="error-message" for="name"/>
   <h:outputLabel for="ccno" value="#{bundle.CCNumber}"/>
   <h:inputText id="ccno" size="19" value="#{cashier.creditCardNumber}"
          required="true" requiredMessage="#{bundle.ReqCreditCard}" >
         <f:converter converterId="ccno"/>
         <f:validateRegex
              pattern="\d{16}|\d{4} \d{4} \d{4} \d{4}|\d{4}-\d{4}-\d{4}-\d{4}" />
    </h:inputText>
    <h:message styleClass="error-message" for="ccno"/> ...
</h:panelGrid>
```

Komponenty wyświetlające komponenty wyboru jednej opcji

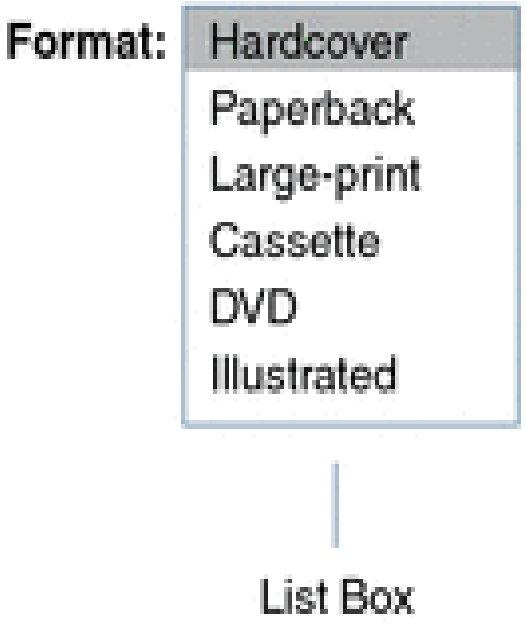
h:selectOneRadio





h:selectBooleanCheckbox

h:selectOneMenu



h:selectOneListbox

(12) Dodawanie wybranych znaczników do strony

h:selectBooleanCheckbox - wyświetlany stan typu Boolean jako check Box

h:selectOneRadio – wyświetlany jako zbiór radio buttons

h:selectOneListbox – wyświetlany jako nieprzewijana lista typu list box

h:selectOneMenu – wyświetlana jako przewijana lista typu drop-down

(13) Zastosowanie zagnieżdżonych znaczników wyboru ocji f:selectItem w komponentach wyboru

Zalety f:selectItem

- dane z listy są definiowane z danych podanych na stronie
- niewiele kodu należy umieścić w ziarnie związanym z komponentem

Przykład wyświetlania rezultatów wyboru (ComboBox, drop-down list)

```
<h:selectOneMenu id="shippingOption" required="true"
                 value="#{cashier.shippingOption}">
  <f:selectItem itemValue="2" itemLabel="#{bundle.QuickShip}"/>
  <f:selectItem itemValue="5" itemLabel="#{bundle.NormalShip}"/>
  <f:selectItem itemValue="7" itemLabel="#{bundle.SaverShip}"/>
</h:selectOneMenu>
```

Atrybut **value** jest zbindowany z właściwościa ziarna, która przechowuje aktualnie wybraną pozycję reprezentowaną przez **itemValue** lub pierwszą, jeśli nie dokonano wyboru. Atrybut **itemLabel** służy do wyświetlania pozycji wyboru.

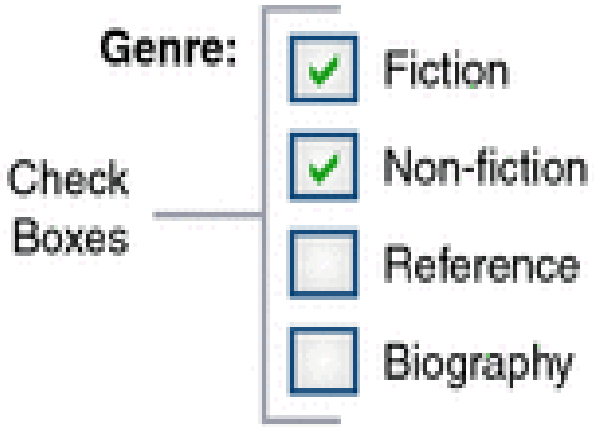
Pozostałe znaczniki definuje się podobnie.

Komponenty wyświetlające komponenty wyboru wielu opcji

h:selectManyCheckbox – wyświetlany jako zbiór check box

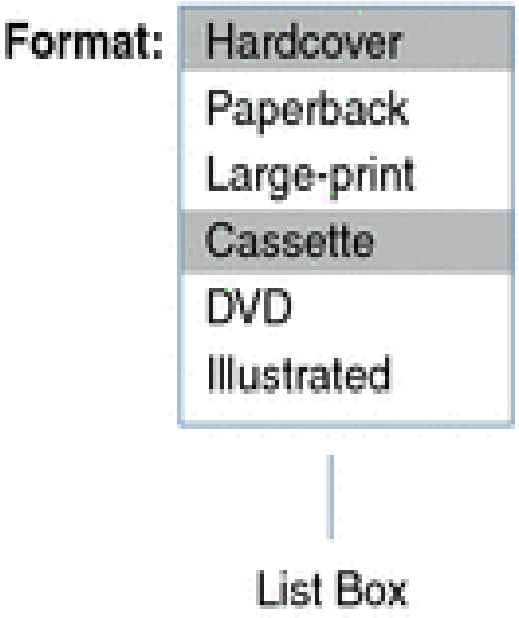
h:selectManyListbox - wyświetlany jako drop-down menu

h:selectManyMenu – wyświetlany jako list box



h:selectManyCheckbox

h:selectManyListbox



h:selectManyMenu

(14) Zastosowanie zagnieżdżonych znaczników wyboru opcji f:selectItems w komponentach wyboru

Znaczniki reprezentują komponenty zagnieżdżane w innych komponentach służących do wyboru jednego (**f:selectItem**) lub kilku elementów (**f:selectItems**).

Zalety f:selectItems:

- są reprezentowane przez różne typy pojemników: Array, Map oraz Collection, zawierających elementy jako zwykłe obiekty Javy (PoJo – Plain Old Java Object)
- można łączyć listy różnych komponentów w jeden komponent
- wartości komponentu mogą być generowane dynamicznie podczas działania programu

(15) Wyświetlanie rezultatów uzyskanych z komponentów wyboru pozycji

1. Po zastosowaniu wyboru za pomocą h:selectBooleanCheckbox

```
<h:selectBooleanCheckbox id="fanClub"
                         binding="#{cashier.specialOffer}" />
 <h:outputLabel for="fanClub"
                binding="#{cashier.specialOfferText}"
                value="#{bundle.DukeFanClub}" />
<h:outputText value="#{bundle.DukeFanClubThanks}”
              rendered="#{cashier.specialOffer.isSelected()}"/>
```

Pole wyboru jest wyświetlane przez znacznik **h:selectBooleanCheckbox.**
Tekst jest wyświetlany przez znacznik **h:outputLabel** (atrybut **value**)

Wyświetlany jest komunikat w znaczniku **h:outputText**, gdy dokonano wyboru (atrybut **rendered**)

2. Po zastosowaniu znacznika h:selectManyCheckbox

```
<h:selectManyCheckbox  id="newslettercheckbox"
                                    layout="pageDirection"
                                    value="#{cashier.newsletters}">
          <f:selectItems value="#{cashier.newsletterItems}"/>
 </h:selectManyCheckbox>
 <h:outputText value="#{bundle.NewsletterThanks}"
                      rendered="#{!empty cashier.newsletters}"/>
  <ul>
   <ui:repeat value="#{cashier.newsletters}" var="nli">
      <li><h:outputText value="#{nli}" /></li>
    </ui:repeat>
 </ul>
```

Elementy kolekcji **cashier.newsletterItems** są generowane programowo.

Blok wyboru jest wyświetlany przez znacznik **h:selectManyCheckbox**
Atrybut **value** znacznika **h:selectManyCheckbox** jest zbindowany z właściwością ziarna, która przechowuje aktualnie wybrane pozycje ze zbioru reprezentowanego przez **f:selectItems value** lub pierwszą, jeśli nie dokonano wyboru. Wybrane pozycje są wyświetlane w znaczniku **ul:repeat** oraz komunikat w znaczniku **h:outputText**, gdy zbiór wybranych pozycji nie jest pusty (atrybut **rendered**).

(16) Zastosowanie komponentu h:dataTable

Komponennt pozwala wyświetlać dane powiązane relacyjnie w postaci tabeli. Wspiera on wyświetlanie kolekcji obiektów reprezentujących dane aplikacji (atrybut **value,** gdzie atrybut **var** deklaruje obiekt tej kolekcji). Znacznik **h:column** reprezentuje kolumnę tabeli z danymi, uzyskanymi w wyniku iteracji po każdym rekordzie danych (atrybuty obiektu, elementy tablicy itp. deklarowanych w **var**) w źródle danych, które są wyświetlane w wierszach tabeli.

```
<h:dataTable id="items"
   captionStyle="font-weight:bold"
   columnClasses="list-column-center, list-column-left, list-column-right, list-
         column-center"
   footerClass="list-footer"
   headerClass="list-header"
   rowClasses="list-row-even, list-row-odd"
   styleClass="list-background"
   summary="#{bundle.ShoppingCart}"
   value="#{cart.items}"
   border="1"
   var="item">
```

Atrybut **value** – odniesienie do zbioru danych, gdzie każda z danych jest deklarowana za pomocą atrybutu **var.**

```
<h:column>
  <f:facet name="header">
    <h:outputText value="#{bundle.ItemQuantity}" />
  </f:facet>
  <h:inputText id="quantity"
        size="4"
        value="#{item.quantity}"
        title="#{bundle.ItemQuantity}">
     <f:validateLongRange minimum="1"/>
   </h:inputText>
    <h:message for="quantity"/>
</h:column>
 <h:column>
    <f:facet name="header">
       <h:outputText value="#{bundle.ItemTitle}"/>
    </f:facet>
    <h:commandLink action="#{showcart.details}">
         <h:outputText value="#{item.title}"/>
    </h:commandLink>
</h:column>
```

atrybut **var** deklaruje rekord danych, gdzie jego składowe sa prezentowane w poszczególnych kolumnach każdego wiersza tabeli

Kolumna z przyciskami do usuwania wiersza za pomoca metody details

```
……..
<f:facet name="footer"
  <h:panelGroup>
      <h:outputText value="#{bundle.Subtotal}"/>
      <h:outputText value="#{cart.total}" />
            <f:convertNumber currencySymbol="$" type="currency" />
       </h:outputText>
   </h:panelGroup>
 </f:facet>
 <f:facet name="caption">
      <h:outputText value="#{bundle.Caption}"/>
 </f:facet>
</h:dataTable>
```

W tabeli wyświetlane są dane książek w księgarni: liczba kupionych książek w kartach płatniczych, ceny i przyciski pozwalające na usunięcie książek z karty płatniczej.

Dane wyświetlane w komponencie dataTable

- Lista ziaren (beans)
- Tablica ziaren
- Pojedyncze ziarno
- Obiekt typu javax.faces.model.DataModel
- Obiekt java.sql.ResultSet
- Obiekt javax.servlet.jsp.jstl.sql.Result
- Obiekt javax.sql.RowSet.

Komponent może wyświetlić wszystkie dane lub ich podzakres określając granice za pomocą atrybutów first i rows

Atrybuty opcjonalne	Zdefiniowane style	Przykłady stylów
captionClass	Tytuł tabeli	
columnClasses	Kolumny tabeli	list-column-center i list-column-right
footerClass	Stopka	
headerClass	Nagłówek	
rowClasses	Wiersze	
styleClass	Wygląd całej tabeli	

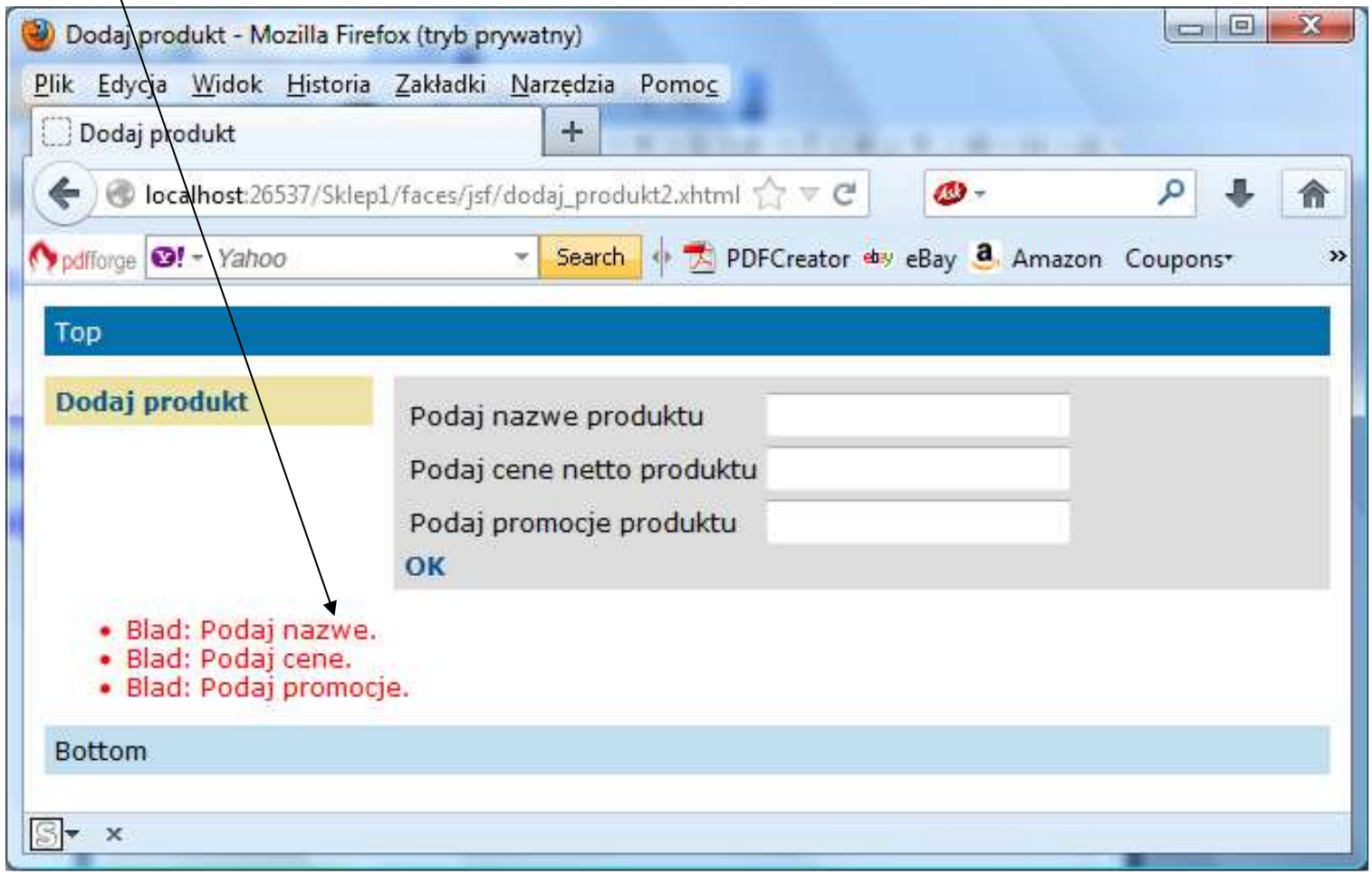
(17) Wyświetlanie wiadomości o błędach konwersji i walidacji za pomocą znaczników h:message i h:messages

```
<p> <h:inputText id="userNo"
                 title="Type a number from 0 to 10:"
                 value="#{userNumberBean.userNumber}">
    <f:validateLongRange minimum="#{userNumberBean.minimum}"
                         maximum="#{userNumberBean.maximum}"/>
  </h:inputText>
  <h:commandButton id="submit" value="Submit"  action="response"/> </p>
 <h:message showSummary="true" showDetail="false"
            style="color: #d20005;
            font-family: 'New Century Schoolbook', serif;
            font-style: oblique;
            text-decoration: overline"
            id="errors1"
            for="userNo"/>
```

Wiadomośc o błędach za pomoca znacznika **h:message** wyświetla się za przyciskiem Submit na stronie i dotyczy wszystkich błędów jednego komponentu.

Znacznik **h:messages** wyświetla błędy wszystkich komponentów na stronie.

```
<h:panelGroup id="messagePanel" layout="block">
        <h:messages errorStyle="color: red" infoStyle="color: green" />
</h:panelGroup>
```



(18) Tworzenie odniesień typu URL za pomocą znaczników h:button i h:link

Znaczniki h:commandLink oraz h:commandButton pozwalają na prostszą definicję odniesień typu URL (zastosowanie żądań typu **POST**) – są używane do przesyłania bloków danych do serwera.

Znaczniki **h:button i h:link** pozwalają na definicję odniesienia za pomocą kilku atrybutów typu **name** po znaku **?** i zakończone znakiem separatora **&;** (zastosowanie żądań typu **GET**).

Przykład:

```
<h:link outcome="somepage" value="Message" />
```

jest renderowana na znacznik html <a>:

```
<a href="/simplebookmark/faces/somepage.xhtml">
    Message</a>
```

(19) Uzywanie parametrów widoków do konfigurowania odniesienia URL

```
<h:body>
  <h:form>
    <h:graphicImage url="duke.waving.gif" alt="Duke waving his hand"/>
    <h2>Hello, #{hello.name}!</h2>
    <p>I've made your
      <h:link outcome="personal" value="personal greeting page!"
              includeViewParams="true">
          <f:param name="Result" value="#{hello.name}"/>
      </h:link>
    </p>
    <h:commandButton id="back" value="Back" action="index" />
  </h:form>
</h:body>
```

Efekt:

http://localhost:8080/bookmarks/faces/personal.xhtml?Result=Timmy

(20) Uzywanie parametrów widoków do konfigurowania odniesienia URL

Równoważna postać deklarowania widoczności wartości parametrów wyświetlanych w adresie URL na stronie internetowej

```
<f:metadata>
        <f:viewParam name="Result" value="#{hello.name}" />
</f:metadata>
```

Teraz można odwołać się wartości właściwości ziarna **hello**

```
<h:outputText value="Howdy, #{hello.name}!" />
```

Efekt:

http://localhost:8080/bookmarks/faces/personal.xhtml?Result=Timmy

Typy wyświetlanych parametrów w adresie URL na stronie:

- Komponent
- Parametry przypadków nawigacji
- Parametry typu **f:viewParam**

(21) Relokacja zasobów za pomocą znaczników h:outputScript i h:outputStylesheet

Przykład 1

```
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html">
 <h:head id="head">
    <title>Resource Relocation</title>
 </h:head>
 <h:body id="body">
    <h:form id="form">
      <h:outputScript name="hello.js"/>
      <h:outputStylesheet name="hello.css"/>
     </h:form>
 </h:body>
</html>
```

Postać strony po renderowaniu:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
     <title>Resource Relocation</title>
     <link type="text/css" rel="stylesheet"
           href="/context-root/faces/javax.faces.resource/hello.css"/>
   </head>
   <body>
      <form id="form" name="form" method="post" action="..." enctype="...">
          <script type="text/javascript"
                  src="/context-root/faces/javax.faces.resource/hello.js">
           </script>
       </form>
    </body>
</html>
```

Przykład 2

```
<html xmlns="http://www.w3.org/1999/xhtml"
        xmlns:h="http://java.sun.com/jsf/html">
  <h:head id="head">
      <title>Resource Relocation</title>
  </h:head>
   <h:body id="body">
          <h:form id="form">
              <h:outputScript name="hello.js" target="#{param.location}"/>
              <h:outputStylesheet name="hello.css"/>
           </h:form>
   </h:body>
</html>
```

Postać strony po renderowaniu:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
        <title>Resource Relocation</title>
        <link type="text/css" rel="stylesheet"
              href="/context-root/faces/javax.faces.resource/hello.css"/>
        <script type="text/javascript"
             src="/context-root/faces/javax.faces.resource/hello.js">
        </script>
  </head>
  <body>
      <form id="form" name="form" method="post" action="..." enctype="...">
        </form>
  </body>
</html>
```

Uwaga: użycie adnotacji @ResourceDependency zwalnia programistów stron JSF do definiowania lokacji zasobów

Opis znaczników obsługiwanych przez Facelets

- Znaczniki typu UI
- Znaczniki do tworzenia szablonów

Szablony JavaServer Faces

- Pozwalają w łatwy sposób rozszerzać interfejs użytkownika
- Zapewniają wielożywalność elementów interfejsu użytkownika
- Zapobiegają budowanie podobnie skonstruowanych stron
- Ułatwiają wprowadzanie standardów w budowie stron internetowych

Znacznik	Funkcje
ui:component	Definiuje, tworzy i umieszcza komponent w drzewie komponentów
ui:composition	Definiuje kompozycję strony, która opcjonalnie używa szablonu. Pomijana jest zawartość poza znacznikiem
ui:debug	Definiuje i tworzy komponent do debugowania, umieszczany w drzewie komponentów
ui:decorate	Podobny do znacznika **ui:composition**, jednak uwzględnia zawartość poza znacznikiem
ui:define	Definiuje zawartość wstawianą do strony za pomocą szablonu
ui:fragment	Podobny do znacznika **ui:component**, jednak uwzględnia zawartość poza znacznikiem
ui:include	Hermetyzuje i wielokrotnie używa zawartość na wielu stronach
ui:insert	Wstawia zawartość do szablonu
ui:param	Używana do wstawiania parametrów do dołączanego pliku
ui:repeat	Używany zamiast znaczników c:forEach lub h:dataTable.
ui:remove	Usuwa zawartość ze strony

Szablon template.xhtml

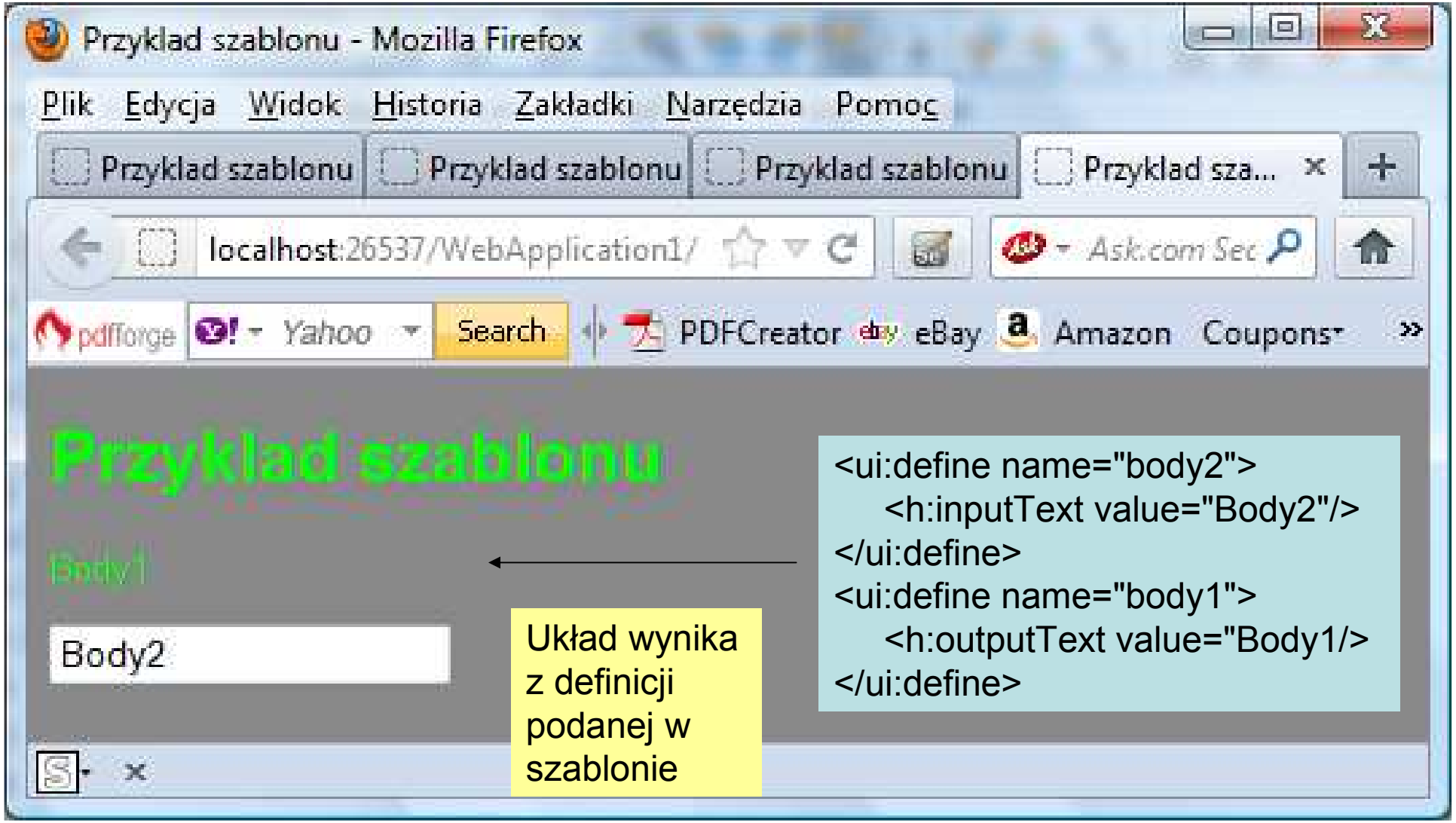
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">
   <h:head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
      <title><ui:insert name="title">Default Title</ui:insert></title>
<h:outputStylesheet name="css/jsf.css"/>
   </h:head>
   <h:body>
      <h1>
         <ui:insert name="title">Default Title</ui:insert>
      </h1>
      <p>
         <ui:insert name="body1">Default Body</ui:insert>
      </p>
      <p>
         <ui:insert name="body2">Default Body</ui:insert>
      </p>
   </h:body>
</html>
```

Przykład strony zbudowanej na szablonie template.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!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:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>Facelet Title</title>
  </h:head>
  <h:body>
    <ui:composition template="./template.xhtml">
      <ui:define name="title">
        Przyklad szablonu
      </ui:define>
      <ui:define name="body2">
        <h:inputText value="Body2"/>
      </ui:define>
      <ui:define name="body1">
            <h:outputText value="Body1/>
      </ui:define>
    </ui:composition>
    <h:outputText value="Body1"/>
  </h:body>
</html>
```

Dyrektywy przestrzeni nazw udostępniających biblioteki użytych typów znaczników

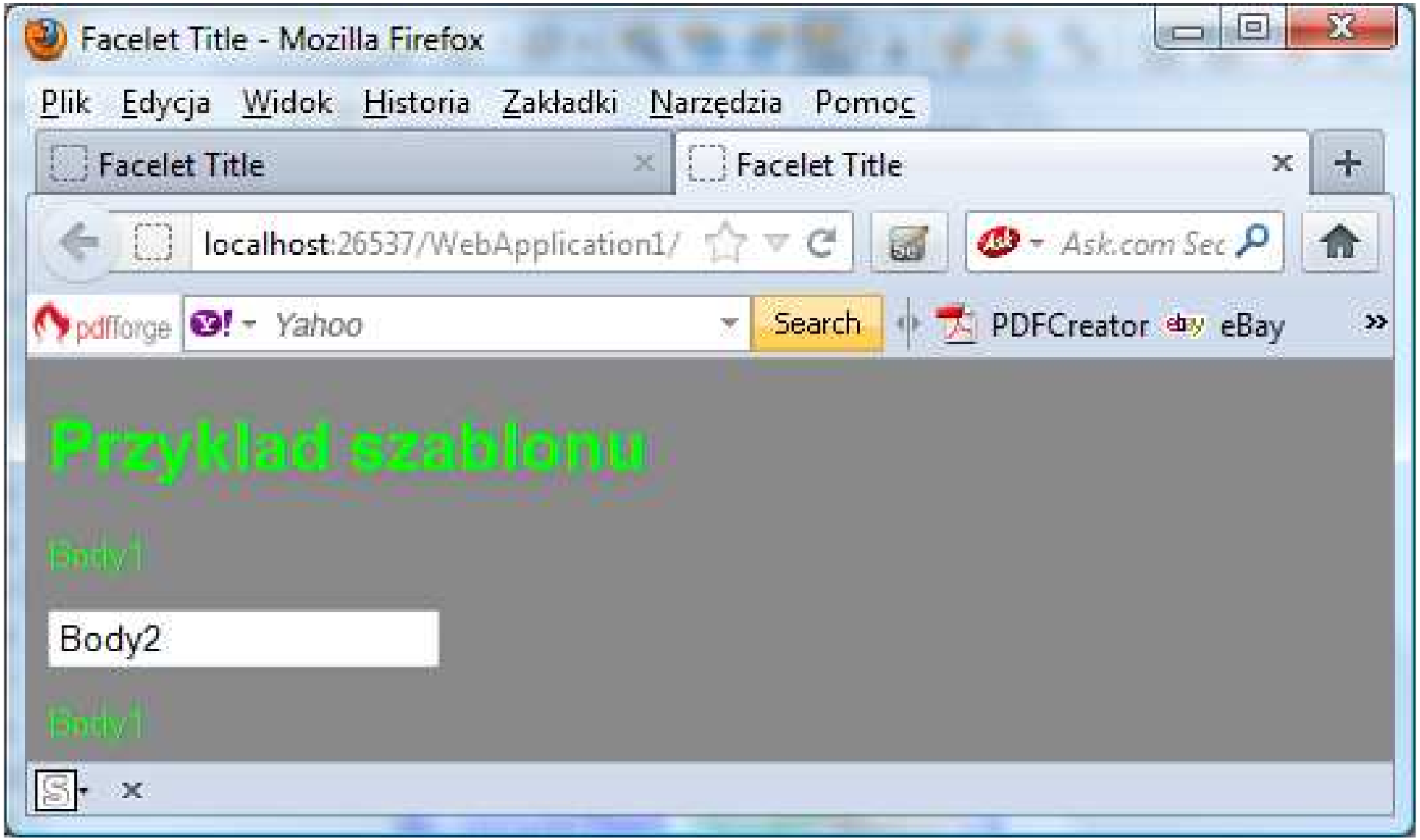
Strona wykonana za pomocą szablonu template.xhtml
brak komponentu <h:outputText value="Body1"/>
z powodu użycia znacznika ui:composition



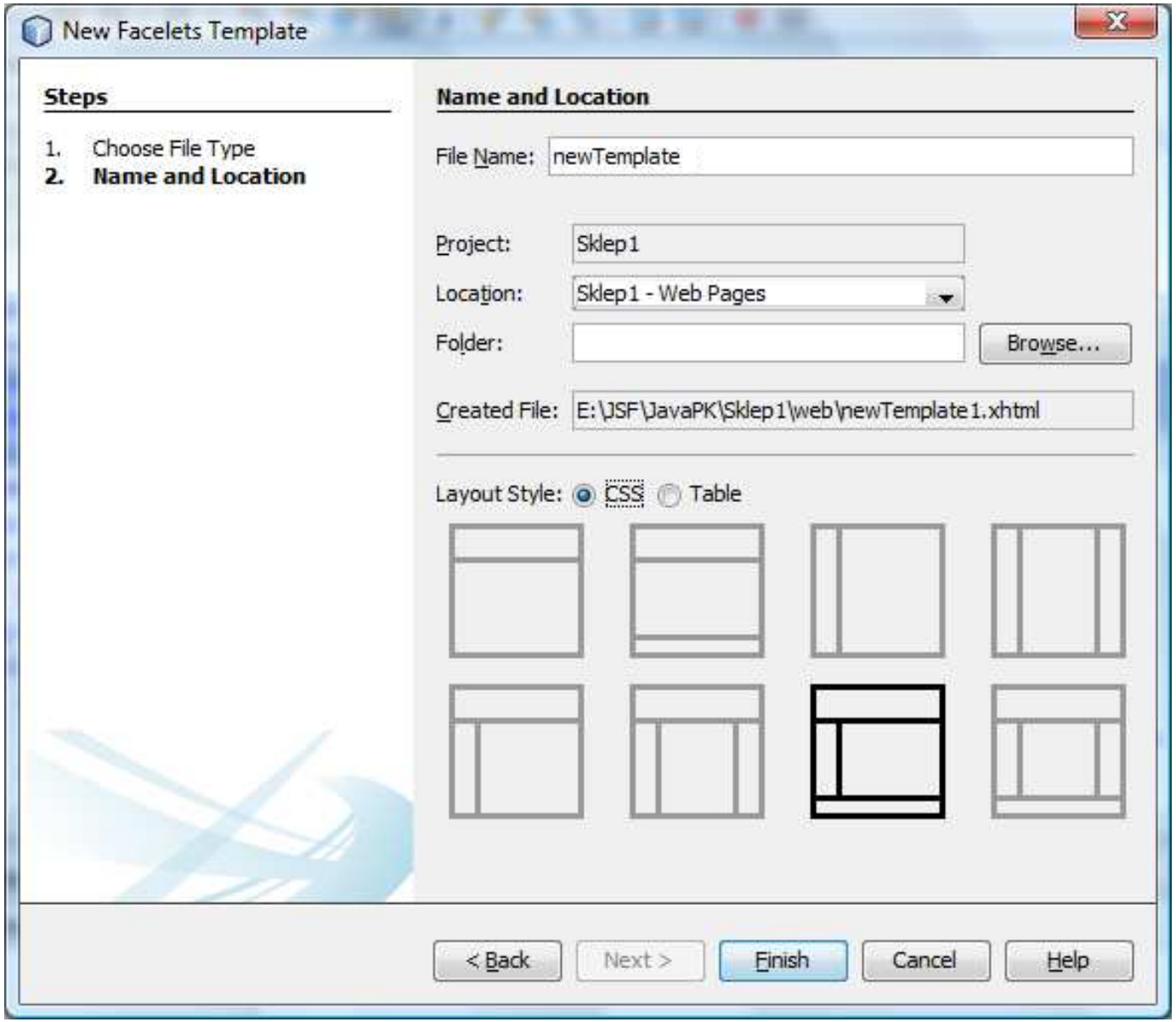
Przykład strony zbudowanej na szablonie template.xhtml ui:decorate

```
<?xml version='1.0' encoding='UTF-8' ?>
<!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:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>Facelet Title</title>
  </h:head>
  <h:body>
      <ui:decorate template="./template.xhtml">
        <ui:define name="title">
          Przyklad szablonu
        </ui:define>
        <ui:define name="body2">
          <h:inputText value="Body2"/>
        </ui:define>
        <ui:define name="body1">
          <h:outputText value="Body1"/>
        </ui:define>
      </ui:decorate>
      <h:outputText value="Body1"/>
  </h:body>
</html>
```

Strona wykonana za pomocą szablonu template.xhtml
pojawił się komponent <h:outputText value="Body1"/>
z powodu użycia znacznika ui:decorate



Szablony stron



Tworzenie szablonów – plik szablonu newTemplate.xhtml (po zmianach)

```
<?xml version='1.0' encoding='UTF-8' ?>
<!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:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <h:outputStylesheet name="css/default.css" />
        <h:outputStylesheet name="css/cssLayout.css"/>
        <title><ui:insert name="title">Facelets Template</ui:insert></title>
    </h:head>
    <h:body>
        <div id="top">
            <h:panelGroup>
                <ui:include src="./logo.xhtml" />
                <ui:insert name="top"></ui:insert>
            </h:panelGroup>
        </div>
```

```
<div>
    <div id="left">
        <h:link
outcome="/faces/jsf/dodaj_produkt2"
                    value="Dodaj produkt"/>
    </div>

    <div id="content" class="left_content">
        <ui:insert
name="content">Content</ui:insert>
    </div>
</div>
```

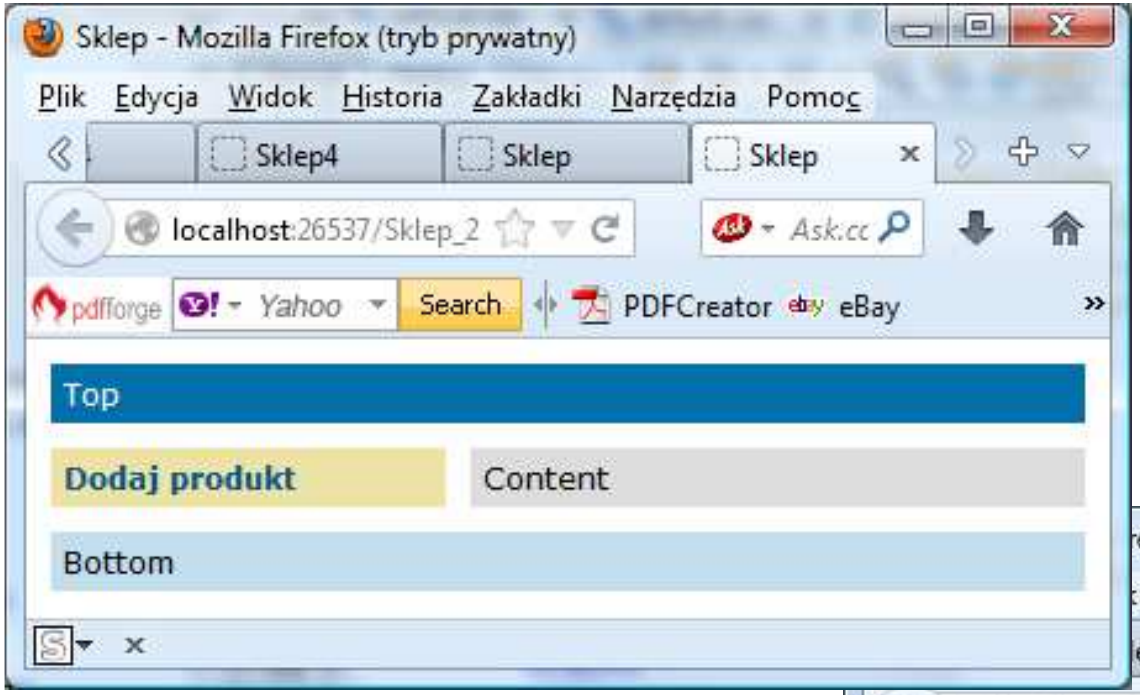
Plik logo.xhtml użyty w szablonie stron za pomocą znacznika
<ui:include src="./logo.xhtml" />

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets">
   <ui:composition>
      <div align="left" style="width: 100%">
         <h:graphicImage value="/resources/obrazy/Dock.jpg"
                         width="59" height="47"
                         title="Molo">
          </h:graphicImage>
      </div>
   </ui:composition>
</html>
```

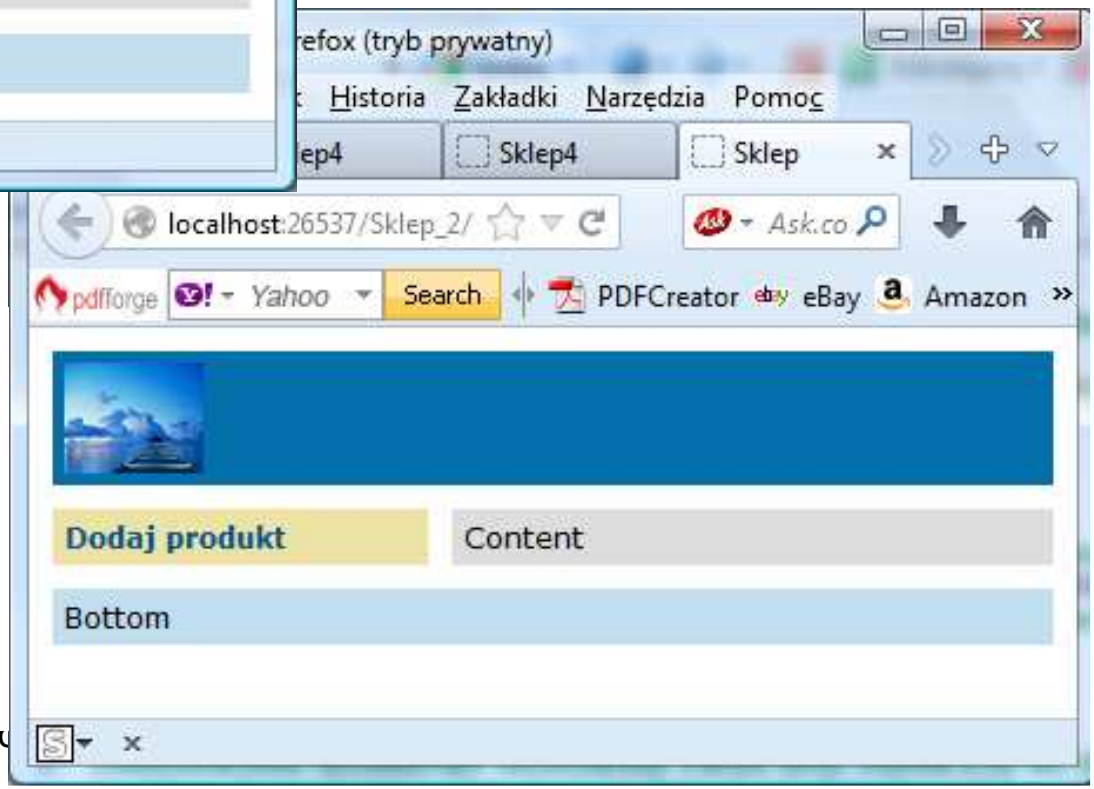
Strona startowa index2.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!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:ui="http://java.sun.com/jsf/facelets">
<body>
      <ui:composition template="./newTemplate.xhtml">

          <ui:define name="title">
             Sklep
          </ui:define>

        </ui:composition>
</body>
</html>
```

Widok stron startowej index2.xhtml – różne wersje szablonu




```
<div id="top">
  <h:panelGroup>
    <ui:insert name="top">
     Top</ui:insert>
  </h:panelGroup>
 </div>
```



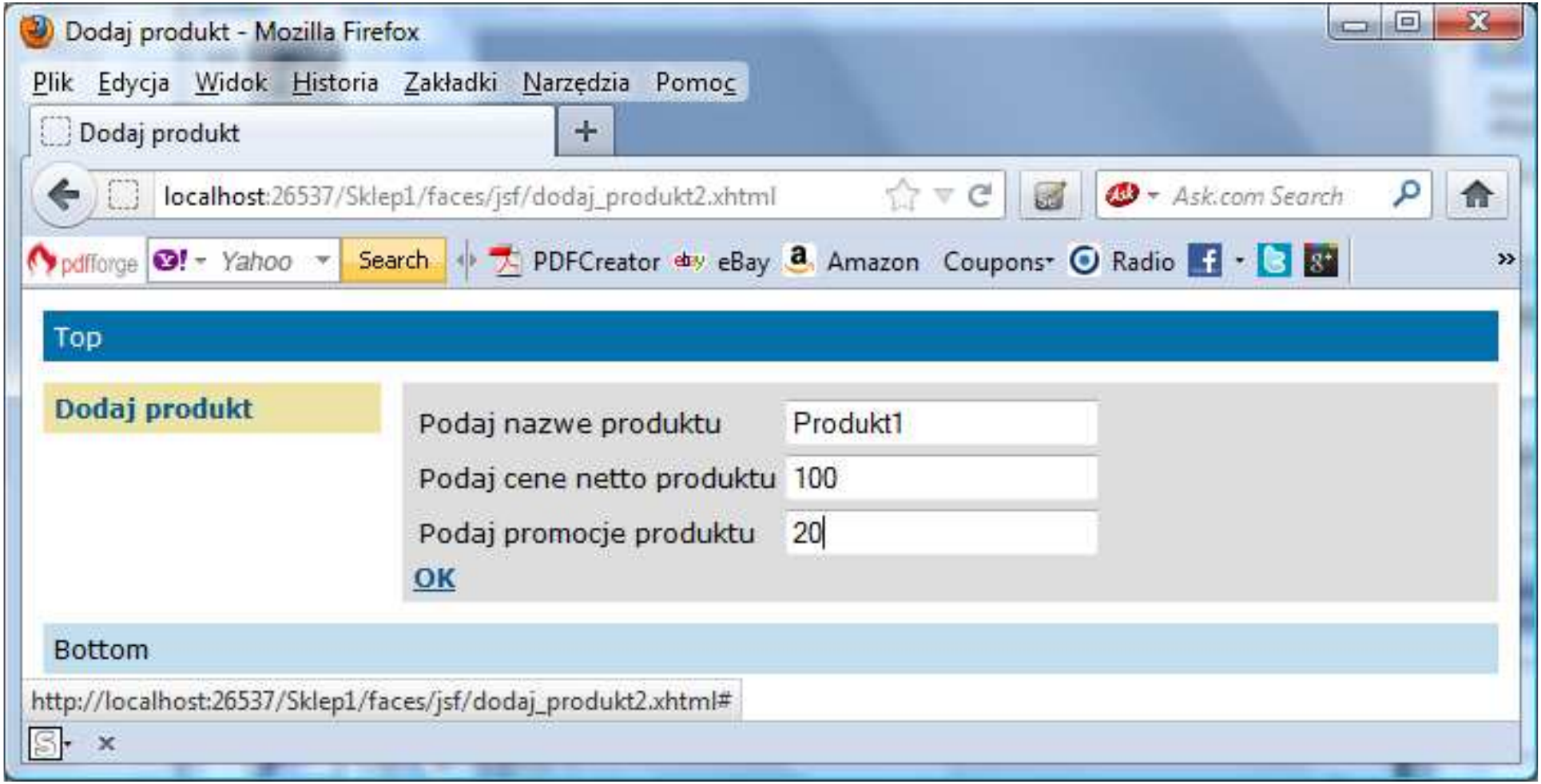

```
<div id="top">
  <h:panelGroup>
   <ui:include src="./logo.xhtml" />
   <ui:insert name="top">
    </ui:insert>
  </h:panelGroup>
 </div>
```

Widok strony dodaj_produkt2.xhtml po kliknięciu na link Dodaj produkt

```
<?xml version='1.0' encoding='UTF-8' ?>
<!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:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html">

 <body>
   <ui:composition template="./../newTemplate.xhtml">
       <ui:define name="title">
          Dodaj produkt
        </ui:define>

       <ui:define name="content">
         <h:form><h:panelGrid columns="2">
            <h:outputLabel value="Podaj nazwe produktu"  for="nazwa" />
            <h:inputText
                    id="nazwa"
                    title="Podaj nazwe:"
                    value="#{managed_produkt.nazwa}"
                    required="true"
                    requiredMessage="Blad: Podaj nazwe." >
            </h:inputText>
```

```
<h:outputLabel value="Podaj cene netto produktu" for="cena" />
        <h:inputText
                id="cena"
                title="Podaj cene:"
                value="#{managed_produkt.cena}"
                required="true"
                requiredMessage="Blad: Podaj cene." >
        </h:inputText>
        <h:outputLabel value="Podaj promocje produktu" for="promocja" />
        <h:inputText
                id="promocja"
                title="Podaj promocje:"
                value="#{managed_produkt.promocja}"
                required="true"
                requiredMessage="Blad: Podaj promocje." >
        </h:inputText>
      </h:panelGrid>

      <h:commandLink  action="#{managed_produkt.dodaj_produkt}" value="OK" />

    </h:form>
   </ui:define>
  </ui:composition>
</body>
</html>
```

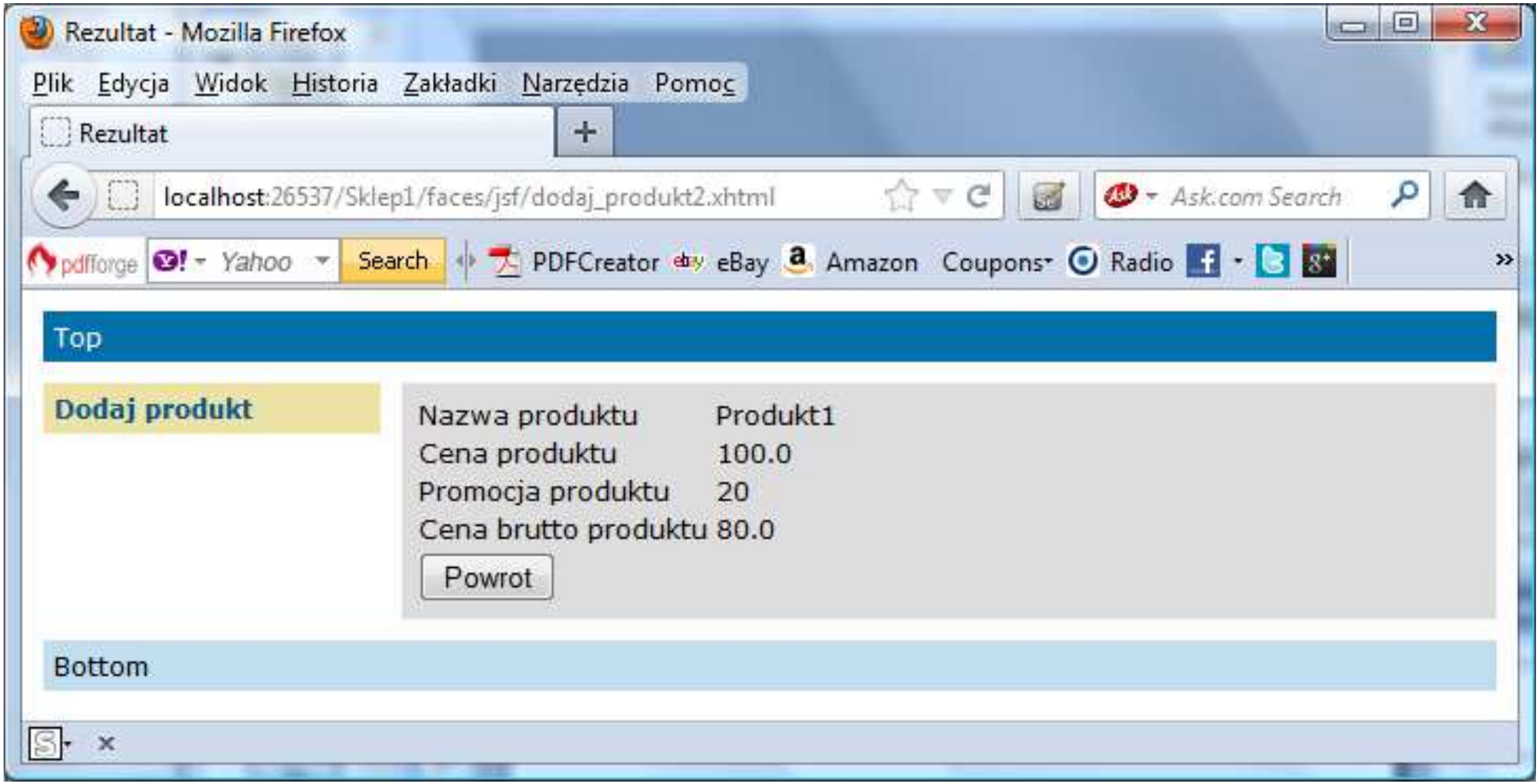
Widok strony dodaj_produkt2.xhtml po kliknięciu na link Dodaj produkt



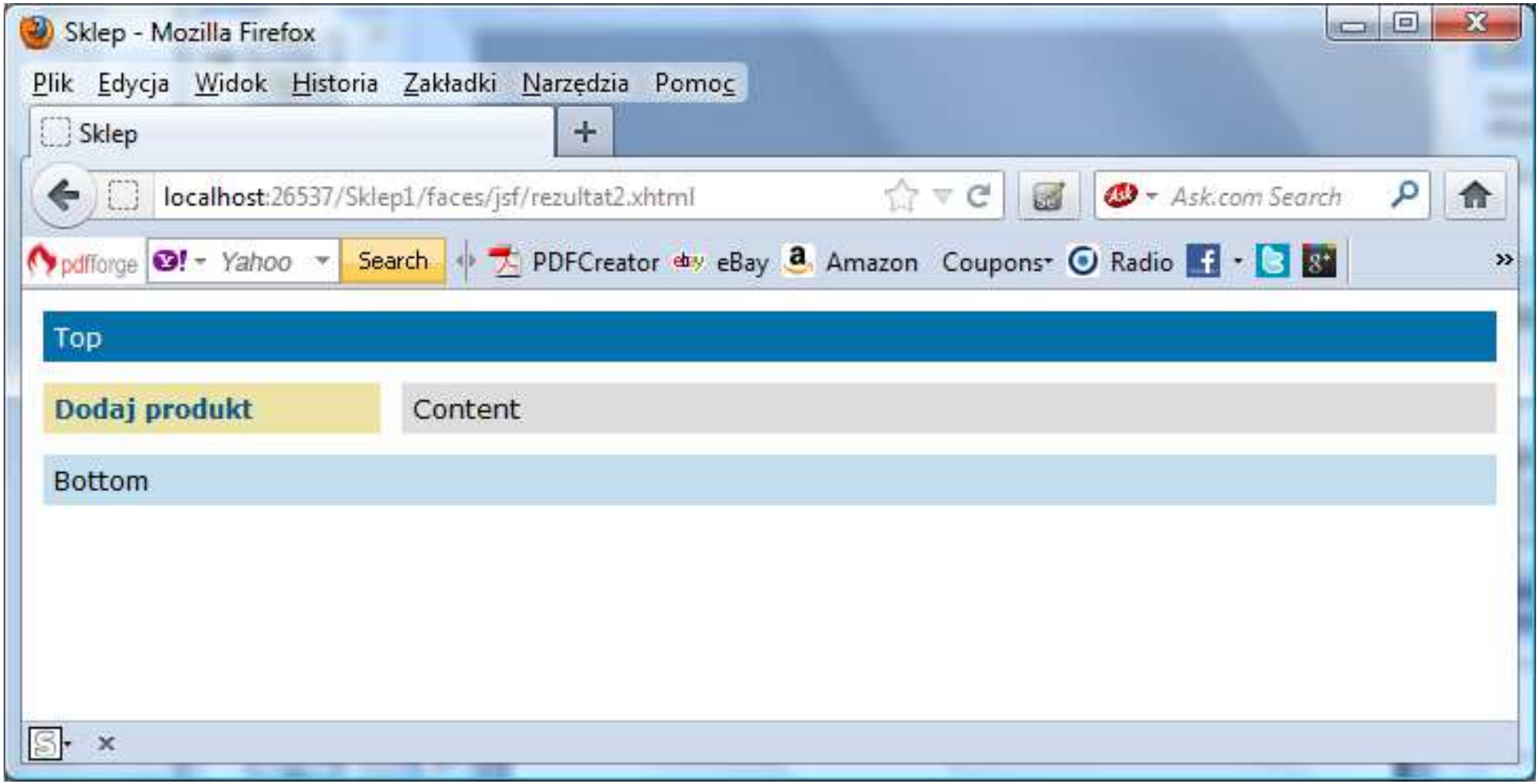
Strona rezultat2.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!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:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html">
  <body>
   <ui:composition  template="./../newTemplate.xhtml">
      <ui:define name="title">
          Rezultat
      </ui:define>
      <ui:define name="content">
        <h:form> <h:panelGrid columns="2">
          <h:outputLabel value="Nazwa produktu"  for="nazwa" />
          <h:outputText id="nazwa" value="#{managed_produkt.nazwa}"/>
          <h:outputLabel value="Cena produktu"  for="cena" />
          <h:outputText id="cena" value="#{managed_produkt.cena}"/>
          <h:outputLabel value="Promocja produktu"  for="promocja" />
          <h:outputText id="promocja" value="#{managed_produkt.promocja}"/>
          <h:outputLabel value="Cena brutto produktu"  for="brutto" />
          <h:outputText id="brutto" value="#{managed_produkt.cena_brutto}" />
          <h:commandButton id="powrot" value="Powrot" action="/faces/index2"/>
        </h:panelGrid></h:form>
      </ui:define>
   </ui:composition>
 </body>
</html>
```

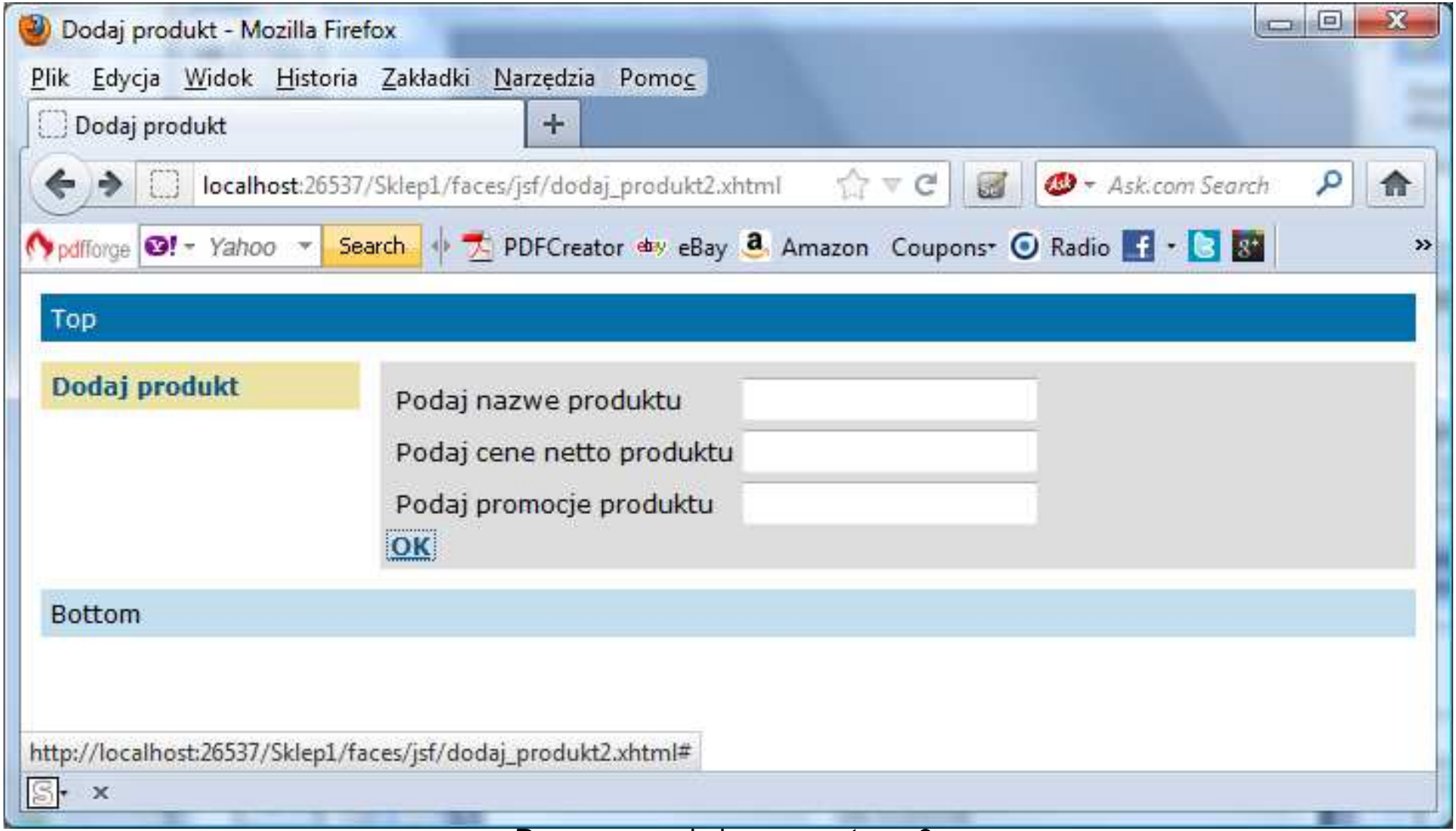
Widok strony rezultat2.xhtml po kliknięciu na przycisk OK



Widok strony index2.xhtml po kliknięciu na przycisk Powrot



Widok strony dodaj_produkt2.xhtml po kliknięciu na link Dodaj produkt



Widok strony dodaj_produkt2.xhtml po kliknięciu na przycisk OK., gdy formularz nie został wypełniony