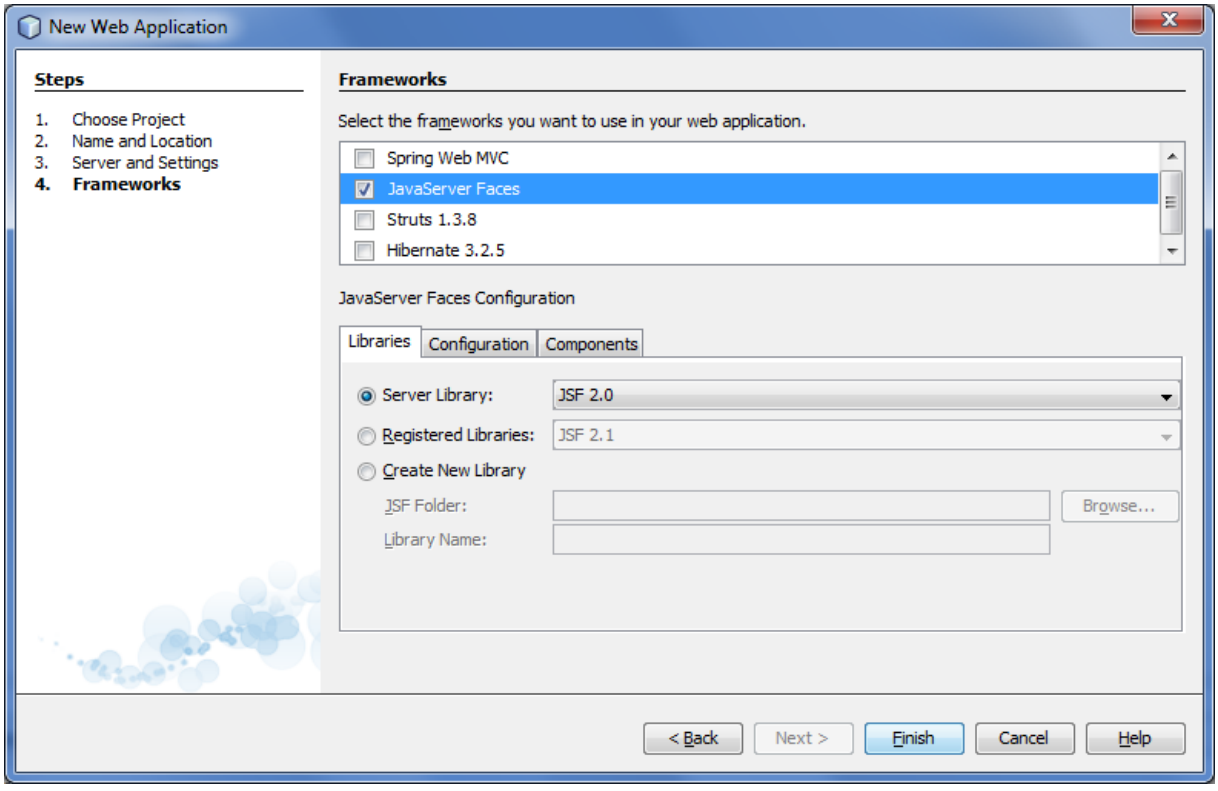
JavaServer Faces

Celem ćwiczenia jest przygotowanie prostej aplikacji wykorzystującej technologię JavaServer Faces w wersji 2.0. Aplikacja umożliwi sprawdzenie poprawności zalogowania się i w zależności od wyniku walidacji wyświetli stosowny dokument. Ćwiczenie obrazuje wykorzystanie podstawowych elementów JSF: komponentów typu *backing bean*, deklaratywnego zarządzania nawigacją w serwisie, obsługi wielojęzyczności i walidacji poprawności danych wprowadzanych przez formularz. Do wykonania ćwiczenia potrzebne jest środowisko NetBeans 7.0.

1. Uruchom program NetBeans i utwórz nowy projekt. Wybierz File→New Project... i z listy kategorii wybierz Java Web, z listy projektów wybierz Web Application. Kliknij przycisk Next >, jako nazwę projektu podaj **JSFLab**. Upewnij się, że projekt będzie oznaczony jako główny projekt (Set as Main Project). Kliknij przycisk Next >. Wybierz zgodność ze standardem Java EE 6, pozostaw zaproponowaną ścieżkę kontekstu i upewnij się, że jako serwer aplikacji wybrany jest GlassFish v3. Kliknij przycisk Next >. W kroku wyboru frameworków zaznacz tylko JavaServer Faces. W sekcji konfiguracji JSF obejrzyj odwzorowanie URL dla serwletu kontrolera i zapoznaj się z dostępnymi opcjami na liście wyboru języka strony. Pozostaw wybraną opcję Facelets i kliknij przycisk Finish.



2. Przejdź do edycji utworzonej wraz z projektem strony `index.xhtml`.
   a) Zmień tytuł strony w sekcji nagłówkowej na „Login page”
   b) Wewnątrz elementu `<h:body>` umieść element `<h:form>` (edytując źródło strony)
   c) Wewnątrz elementu `<h:form>` wprowadź znak `<` i poczekaj na pojawienie się okienka z podpowiedziami. Przejrzyj listę dostępnych znaczników i zwróć uwagę na zestawy (biblioteki) znaczników, które są dostępne dla stron JSF opartych o Facelets. Z listy dostępnych znaczników wybierz `<h:inputText>`

d) Wprowadź spację po umieszczonym przed chwilą znaczniku i poczekaj na pojawienie się okienka z dostępnymi atrybutami dla tego znacznika. Wybierz atrybut `id` i ustaw jego wartość na `usernameField`. Zamknij znacznik domykając element.

   Uwagi: Twórcy środowiska NetBeans w wersji 7 zrezygnowali z wizualnego edytora stron JSF twierdząc, że dzięki wspomaganiu w edycji kodu tworzenie stron w edytorze tekstowym jest równie wydajne, a daje programiście większą kontrolę nad kodem źródłowym i wynikowym HTML-em. Kontekstowe podpowiedzi dla istniejących już znaczników można wywołać kombinacją klawiszy Ctrl-Space.

e) Umieść wewnątrz elementu `<h:form>` element `<h:panelGrid>` z dwoma kolumnami (wykorzystaj jego atrybut `columns`). Przenieś pole tekstowe z nazwą użytkownika do wnętrza panelu.

f) Wewnątrz elementu `<h:panelGrid>`, a powyżej pola tekstowego umieść element `<h:outputLabel>`. Ustaw atrybut `for` tego elementu na identyfikator pola tekstowego z nazwą użytkownika. Następnie dodaj w tym elemencie atrybut `value` z wartością `Username`.

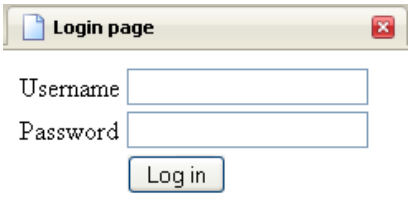
g) Uruchom aplikację. Podejrzyj źródło strony. Zwróć uwagę na elementy HTML odpowiadające komponentom JSF.

3. Wróć do edycji kodu źródłowego strony `index.xhtml`.

   a) Poniżej pola tekstowego z nazwą użytkownika, ale wewnątrz panelu, umieść etykietę (znacznik `<h:outputLabel>`) i pole do wprowadzania hasła (znacznik `<h:inputSecret>`). Ustaw identyfikator tego pola na `passwordField`. Jako tekst etykiety podaj `Password`. Następnie powiąż etykietę z polem.

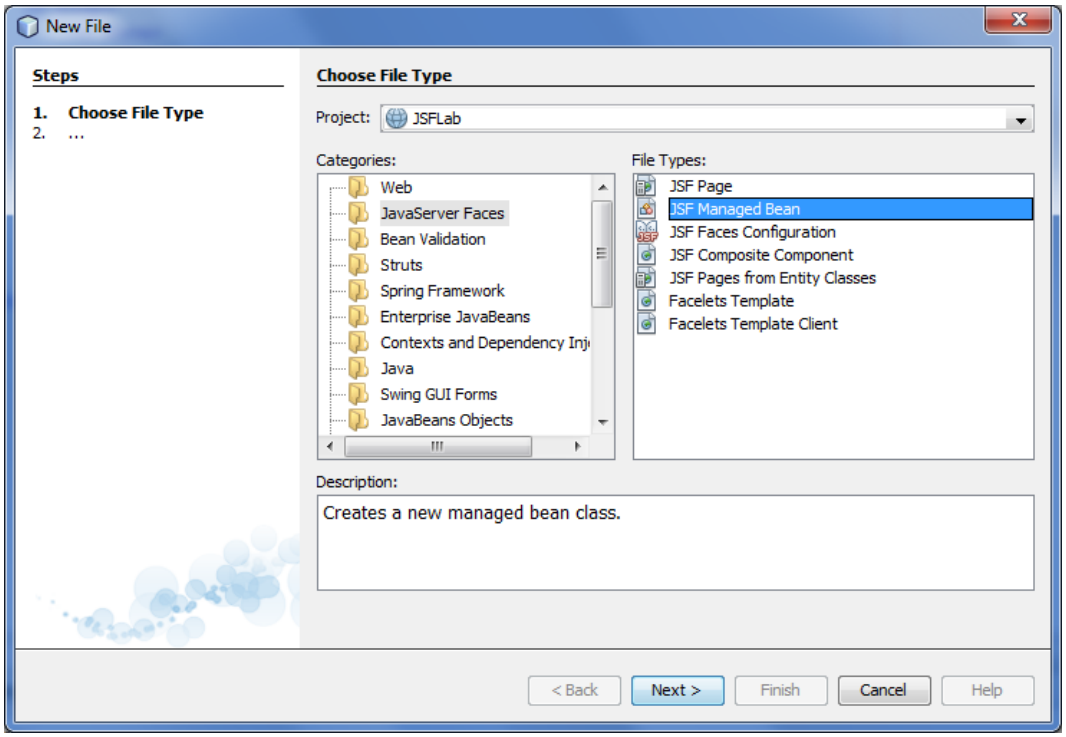
   b) Poniżej pola z hasłem umieść przycisk (znacznik `<h:commandButton>`). Ustaw identyfikator (`id`) przycisku na `loginButton` a atrybut `value` na `Log in`. Aby przycisk wyświetlany był pod polami do wprowadzania danych, a nie pod etykietami, poprzedź go w kodzie pustym elementem `<h:panelGroup>` (element ten jest wykorzystywany w miejscach gdzie powinien znajdować się dokładnie jeden komponent JSF i zazwyczaj służy do grupowania kilku komponentów).

   c) Uruchom stronę i upewnij się, że wygląda w przeglądarce jak na poniższym obrazku. Sprawdź co się dzieje po kliknięciu przycisku.

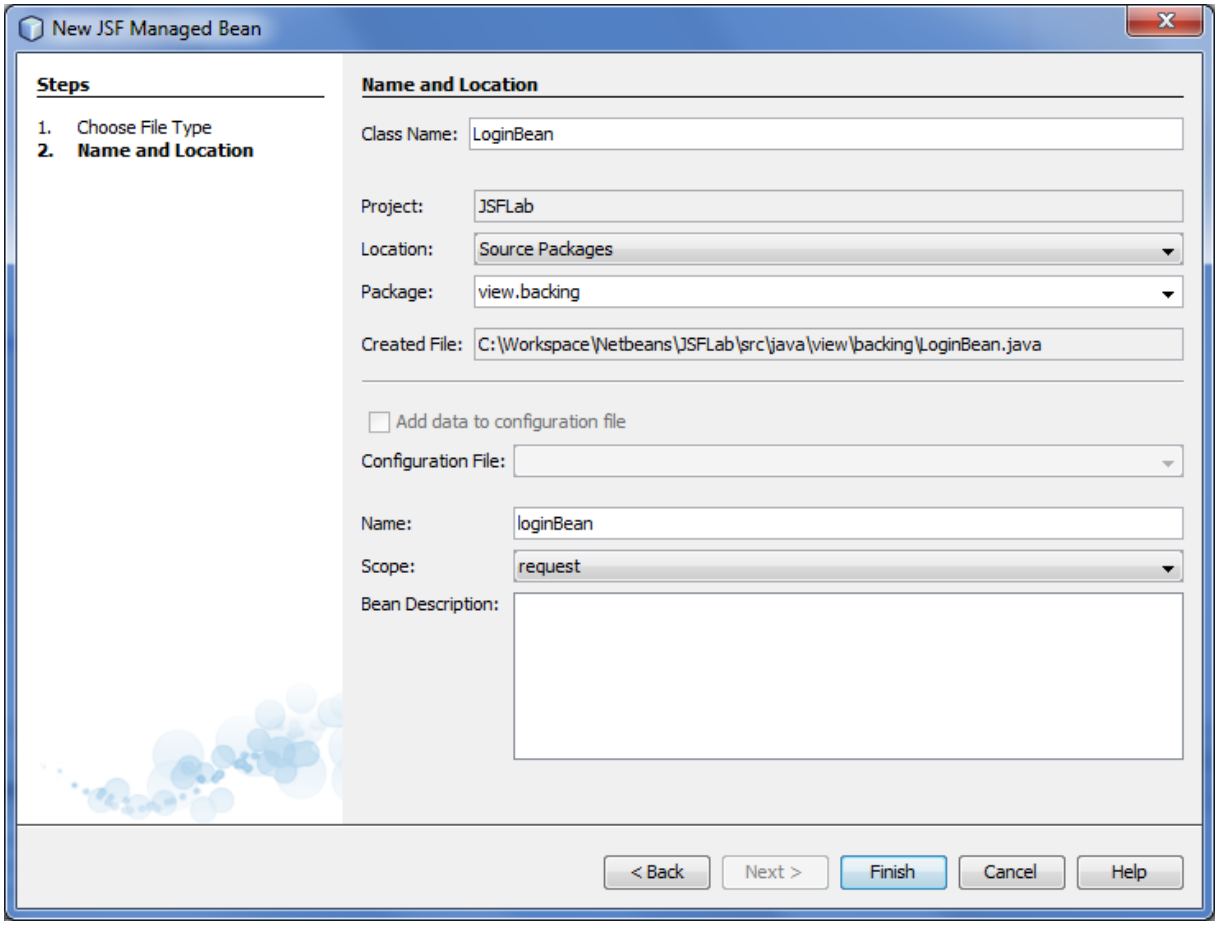


   d) Utwórz w projekcie nowy komponent zarządzany JSF (JSF Managed Bean), który będzie pełnił rolę backing bean dla utworzonej przed chwilą strony JSF.

   e) Z poziomu węzła projektu w panelu projektów uruchom poprzez menu kontekstowe kreator JSF Managed Bean z kategorii JavaServer Faces. W tym celu wybierz z menu kontekstowego pozycje New → Other..., następnie w panelu Categories zaznacz pozycję Java Server Faces a w panelu File Types pozycję JSF Managed Bean. kliknij przycisk Next >.



f) W drugim kroku kreatora podaj nazwę klasy (`LoginBean`), pakiet (`view.backing`), nazwę komponentu (`LoginBean` – powinna ustawić się automatycznie) i jego zasięg (`request`), a następnie kliknij przycisk Finish.

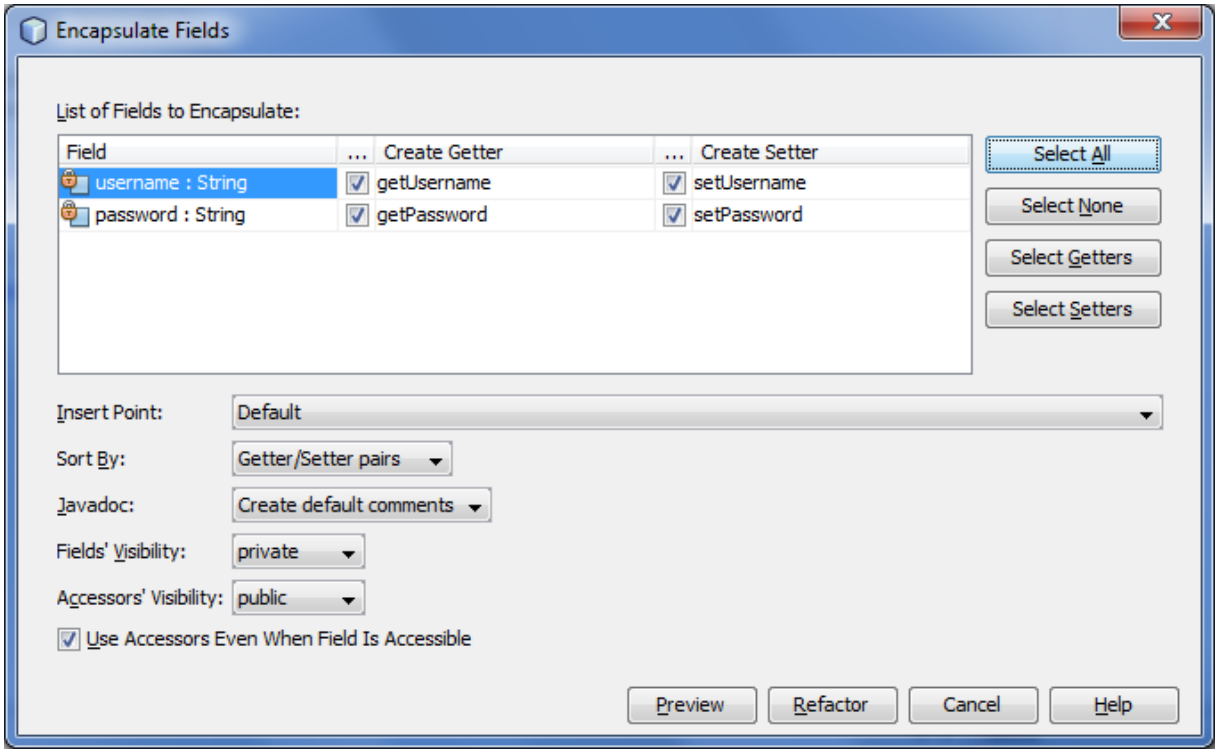


g) Obejrzyj kod wygenerowanej klasy komponentu. Zwróć uwagę na adnotacje, które zastępują ustawienia z dawniej obowiązkowego pliku `faces-config.xml`.

h) Dodaj ręcznie w klasie komponentu prywatne pola `username` i `password` typu `String`, w których będą przechowywane wartości z formularza.

i) Z poziomu edytora kodu wywołaj prawym klawiszem myszy menu kontekstowe i wybierz z niego opcję Refactor → Encapsulate Fields.

j) Zaznacz utworzenie publicznych setterów i getterów dla obu prywatnych pól i kliknij przycisk Refactor.



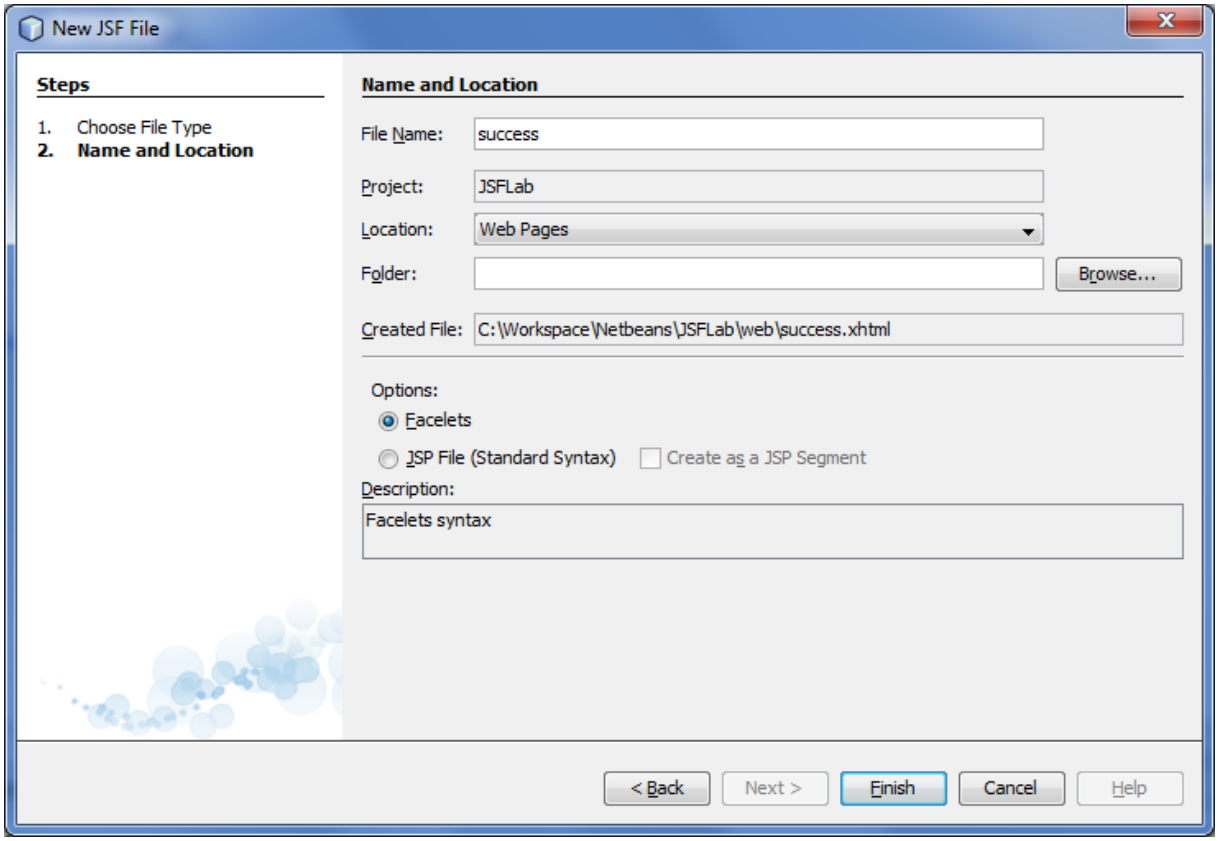
4. Wróć do edycji pliku `index.xhtml` aby powiązać pola formularza z właściwościami komponentu backing bean (będą to powiązania właściwości backing bean z wartościami komponentów).

   a) Dodaj w elemencie `<h:inputText>` atrybut `value`. Jako wartość tego atrybutu wprowadź znaki `#{` rozpoczynające wyrażenie EL i poczekaj na wyświetlenie listy dostępnych obiektów. Wybierz z listy zdefiniowany wcześniej komponent backing bean (`loginBean`). Następnie za jego nazwą wprowadź kropkę i podaj właściwość `username`.

b) Analogicznie powiąż pole do wprowadzania hasła z odpowiednią właściwością komponentu backing bean (`password`).

c) W elemencie `<h:commandButton>` dodaj atrybut `action` z wartością `success.xhtml`. Jest to nazwa pliku strony, która zostanie utworzona w kolejnym punkcie ćwiczenia.

5. Utwórz w projekcie nową stronę JSF (opartą o Facelets) o nazwie `success.xhtml`.



a) W kodzie strony zastąp nagłówek i ciało poniższymi, wykorzystując wsparcie edytora NetBeans przy dodawaniu elementu wyświetlającego podaną nazwę użytkownika. Dzięki temu, że przekierowanie do nowej strony jest domyślnie realizowane w JSF w ramach jednego żądania, możliwy jest odczyt informacji z komponentu backing bean o zasięgu `request` obsługującego poprzednią stronę.

```
<h:head>
      <title>Welcome page</title>
  </h:head>
  <h:body>
      Welcome <h:outputText value="#{loginBean.username}"/>!
  </h:body>
```

b) Uruchom aplikację i przetestuj logowanie (oczywiście w aktualnej formie zawsze kończy się ono sukcesem).

6. Zmodyfikuj aplikację, tak aby nawigacja ze strony logowania była warunkowa, zależna od wprowadzonych danych.
   a) Przejdź do edycji klasy komponentu backing bean strony logowania.
   b) Dodaj w klasie publiczną bezargumentową metodę `login()` zwracającą `String`.
   c) Jako ciało metody wprowadź na razie instrukcję `return` zwracającą nazwę strony `success.xhtml`.
   d) Przejdź do edycji strony `index.html` i w kodzie przycisku zastąp nazwę strony `success.xhtml` odwołaniem do metody `login` komponentu backing bean (użyj odpowiedniego wyrażenia EL korzystając z wsparcia edytora kodu w NetBeans)

```
<h:commandButton id="loginButton" value="Log in"
                 action="#{loginBean.login()}"></h:commandButton>
```

   e) Zapisz wszystkie zmiany i uruchom aplikację, aby sprawdzić czy działa jak poprzednio.
   f) Wróć do edycji klasy komponentu backing bean strony logowania i zmień treść metody `login` tak aby kierowała do strony `success.xhtml` w przypadku gdy podane hasło jest identyczne z nazwą użytkownika, a w przeciwnym wypadku – do nieistniejącej jeszcze strony `failure.xhtml`. Kod metody może wyglądać jak poniższy:

```
public String login() {
    if (username.equals(password)) {
        return "success.xhtml";
    } else {
        return "failure.xhtml";
    }
}
```

7. Samodzielnie utwórz w projekcie stronę `failure.xhtml`. Strona powinna wyświetlać tekst „Sorry <username>...” (w miejscu <username> powinna pojawić się nazwa użytkownika, podana na stronie logowania) i umożliwiać powrót do strony logowania poprzez kliknięcie linku „Back”. Link powinien być utworzony jako komponent `<h:commandLink>`. Należy mu ustawić atrybuty analogiczne do tych co w przypadku przycisku. Ponieważ powrót do strony logowania będzie bezwarunkowy, nie ma w tym wypadku konieczności skorzystania z metody komponentu backing bean do obsługi nawigacji. Przykład rozwiązania podano poniżej.

```
<h:head>
    <title>Failure</title>
</h:head>
<h:body>
    <h:form>
        <h:panelGrid columns="1">
            <h:outputText value="Sorry #{loginBean.username}..."/>
            <h:commandLink id="backLink" value="Back"
                           action="index.xhtml"></h:commandLink>
        </h:panelGrid>
    </h:form>
</h:body>
```

Uruchom i przetestuj aplikację dla poprawnych i niepoprawnych danych logowania.

8. Powróć do edycji pliku `index.xhtml` aby rozbudować aplikację o mechanizmy walidacji.
   a) Ustaw w komponencie `usernameField` właściwość `required` na `true`.
   b) Zmień liczbę kolumn komponentu `<h:panelGrid>` na `3`.
   c) Umieść w formularzu komponent `<h:message>`, tak aby był wyświetlany po prawej stronie komponentu `usernameField`. Powiąż komponent `<h:message>` z polem tekstowym ustawiając wartość atrybutu `for`.
   d) Zmodyfikuj element `<h:inputSecret>`, tak aby miał odrębny znacznik zamykający (o ile jeszcze go nie ma), a następnie w treści elementu umieść komponent walidujący `<f:validateLength>`. Ustaw atrybuty walidatora (`minimum` i `maximum`), tak aby wprowadzone hasło musiało mieć od 4 do 6 znaków.
   e) Dodaj do formularza komponent `<h:message>`, który będzie wyświetlał komunikat o niepoprawnej długości wprowadzonego hasła.
   f) Uruchom aplikację i przetestuj dodane mechanizmy walidacyjne (pozostaw puste pole z nazwą użytkownika a w pole z hasłem wprowadź ciąg znaków o długości np. 7).
9. Wróć do edycji strony `index.xhtml` i pod panelem `<h:panelGrid>` umieść komponent `<h:selectBooleanCheckbox>` ustawiając jego atrybut `id` na `acceptCheckBox`. Obok niego umieść komponent `<h:outputText>` i ustaw jego atrybut `value` na `I accept the terms of service`.
10. Przejdź do edycji kodu klasy komponentu backing bean strony (`LoginBean`) i dokonaj poniższych modyfikacji:
    a) Dodaj prywatne pole `acceptCheckbox` typu `HtmlSelectBooleanCheckbox`.
    b) Dodaj prywatne pole `loginButton` typu `HtmlCommandButton`.
    c) Zaimportuj obie klasy wykorzystane jako typy powyższych pól.
    d) Utwórz dla obu dodanych pól publiczne metody getter i setter (posłuż się odpowiednim kreatorem).

11. Wróć do edycji strony `index.xhtml` i dokonaj poniższych modyfikacji:

    a) Powiąż komponent `<h:selectBooleanCheckbox>` z dodaną właśnie właściwością w klasie backing bean. (Wykorzystaj atrybut `binding` i odpowiednie wyrażenie EL.)

    b) Analogicznie powiąż komponent `<h:commandButton>` z przygotowaną do tego celu właściwością komponentu backing bean.

```
...
  <h:commandButton id="loginButton" value="Log in"
                       binding="#{loginBean.loginButton}"
                       action="#{loginBean.login()}">
  </h:commandButton>
</h:panelGrid>
<h:selectBooleanCheckbox id="acceptCheckBox"
                         binding="#{loginBean.acceptCheckbox}">
</h:selectBooleanCheckbox>
...
```

12. Ustaw komponentowi przycisku atrybut `disabled` na `true`.

13. W klasie komponentu backing bean dodaj poniższą metodę:

```
    public void activateButton(ValueChangeEvent e)
    {
        if (acceptCheckbox.isSelected())
            loginButton.setDisabled(false);
        else
            loginButton.setDisabled(true);
    }
```

    Zaimportuj odpowiednią bibliotekę.

14. Wróć do edycji kodu strony `index.xhtml` i dodaj w elemencie pola wyboru atrybut `valueChangeListener` z wartością będącą wyrażeniem EL wskazującym metodę dodaną w poprzednim kroku ćwiczenia w klasie backing bean.

```
<h:selectBooleanCheckbox
   id="acceptCheckBox"
   binding="#{loginBean.acceptCheckbox}"
   valueChangeListener="#{loginBean.activateButton}">
</h:selectBooleanCheckbox>
```

15. Uruchom aplikację i zaobserwuj sposób jej działania przy zaznaczaniu i odznaczaniu pola wyboru. Na czym polega problem?

16. Wróć do edycji kodu strony `index.xhtml`. Dodaj w elemencie pola wyboru atrybut `onchange` z wartością `submit()` powodującą zatwierdzenie formularza funkcją JavaScript.

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

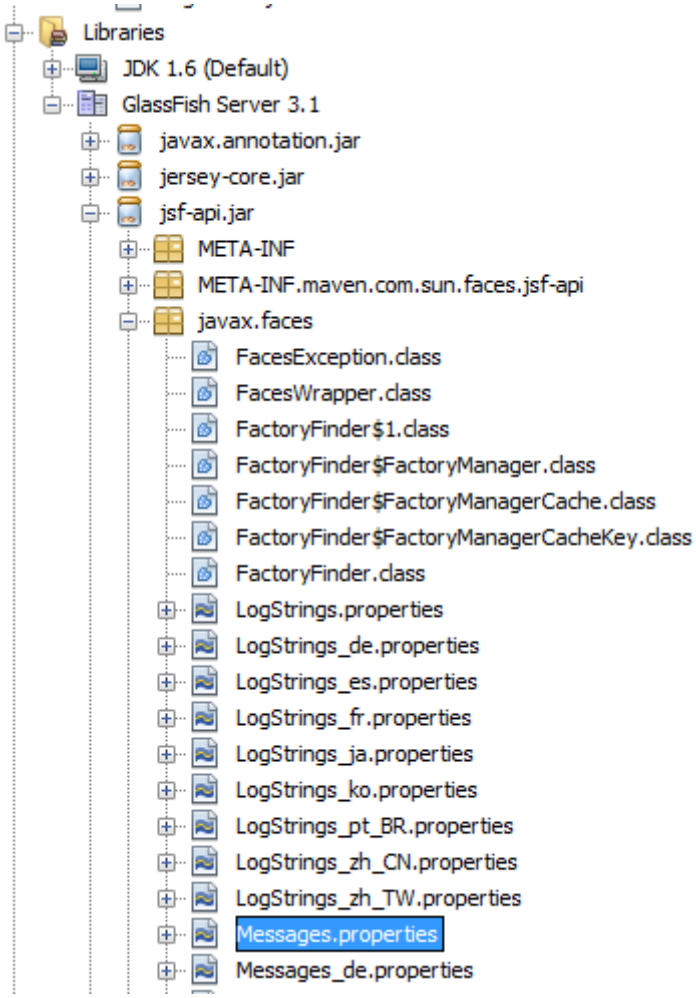
17. Ponownie uruchom aplikację i zaobserwuj sposób jej działania przy zaznaczaniu i odznaczaniu pola wyboru przy niewypełnionych polach z nazwą użytkownika i hasłem. Na czym teraz polega problem?

18. Ustaw atrybut `immediate` na `true` dla pola wyboru. Następnie dodaj w metodzie komponentu backing bean wywoływanej w reakcji na zmianę stanu pola wyboru na końcu poniższy kod i ponownie uruchom i przetestuj aplikację (nie zapomnij zaimportować klasy `FacesContext`).

```
FacesContext context = FacesContext.getCurrentInstance();
context.renderResponse();
```

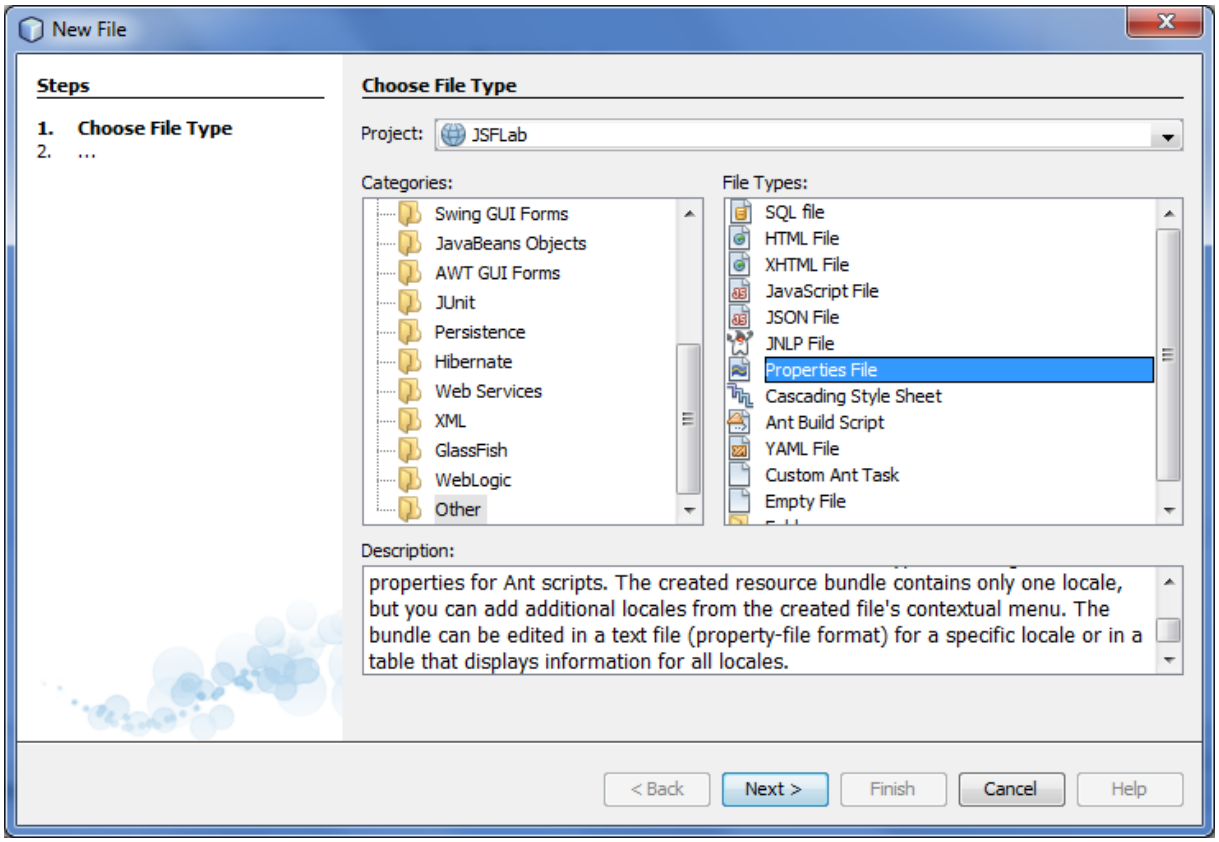
19. Ustaw własne komunikaty dla użytych walidatorów. W tym celu:

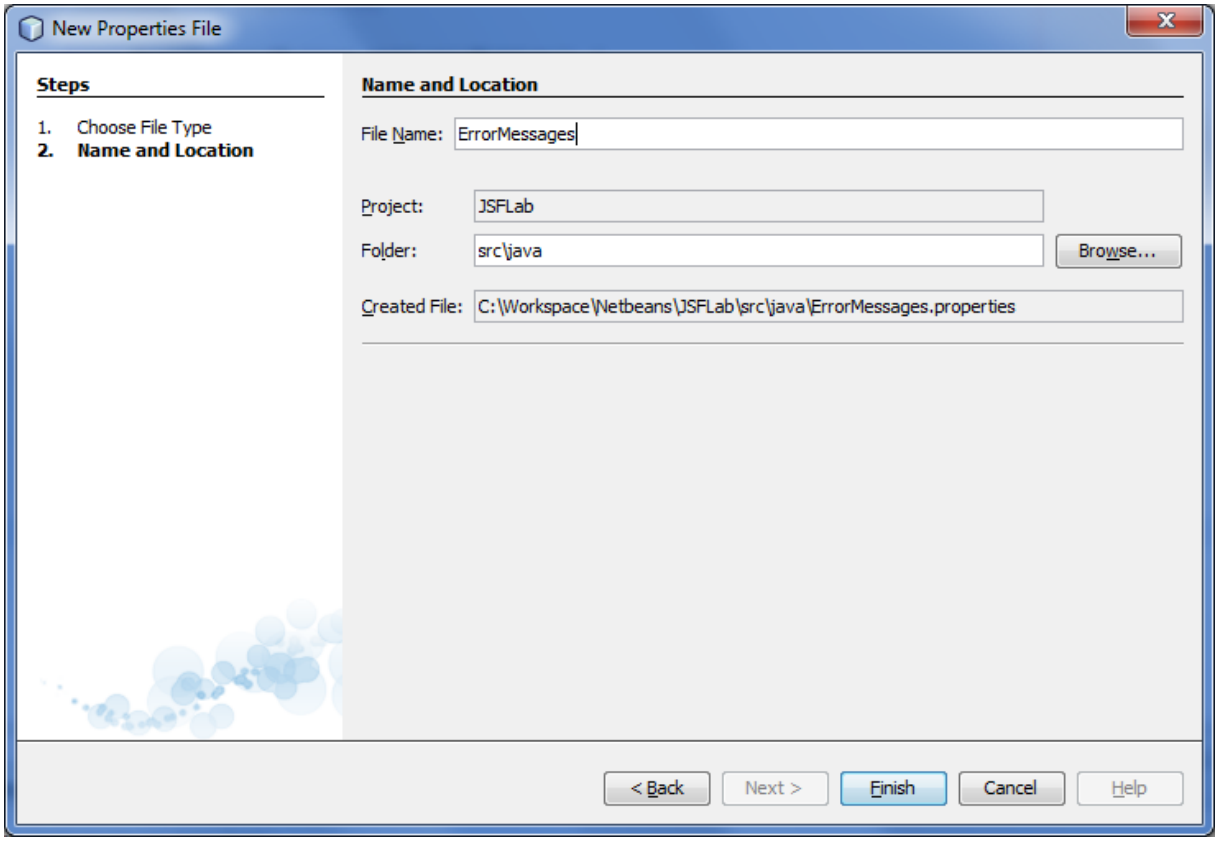
a) Rozwiń zawartość archiwum `jsf-api.jar` dołączonego do projektu. Odszukaj plik `Messages.properties` w folderze `javax.faces`.



b) Otwórz plik `Messages.properties` w środowisku NetBeans. Spróbuj odnaleźć klucze związane z błędami walidacji występującymi w naszej aplikacji i obejrzyj komunikaty o błędach.

c) Dodaj w projekcie z poziomu węzła Source Packages plik `properties` korzystając z kreatora Properties File z kategorii Other. Nazwij plik `ErrorMessages.properties`.





d) Wklej do utworzonego pliku poniższe informacje (skopiowane z przeglądanego wcześniej pliku z domyślnymi komunikatami; każdy klucz w jednym wierszu):

```
javax.faces.component.UIInput.REQUIRED={0}: Validation Error: Value is
```

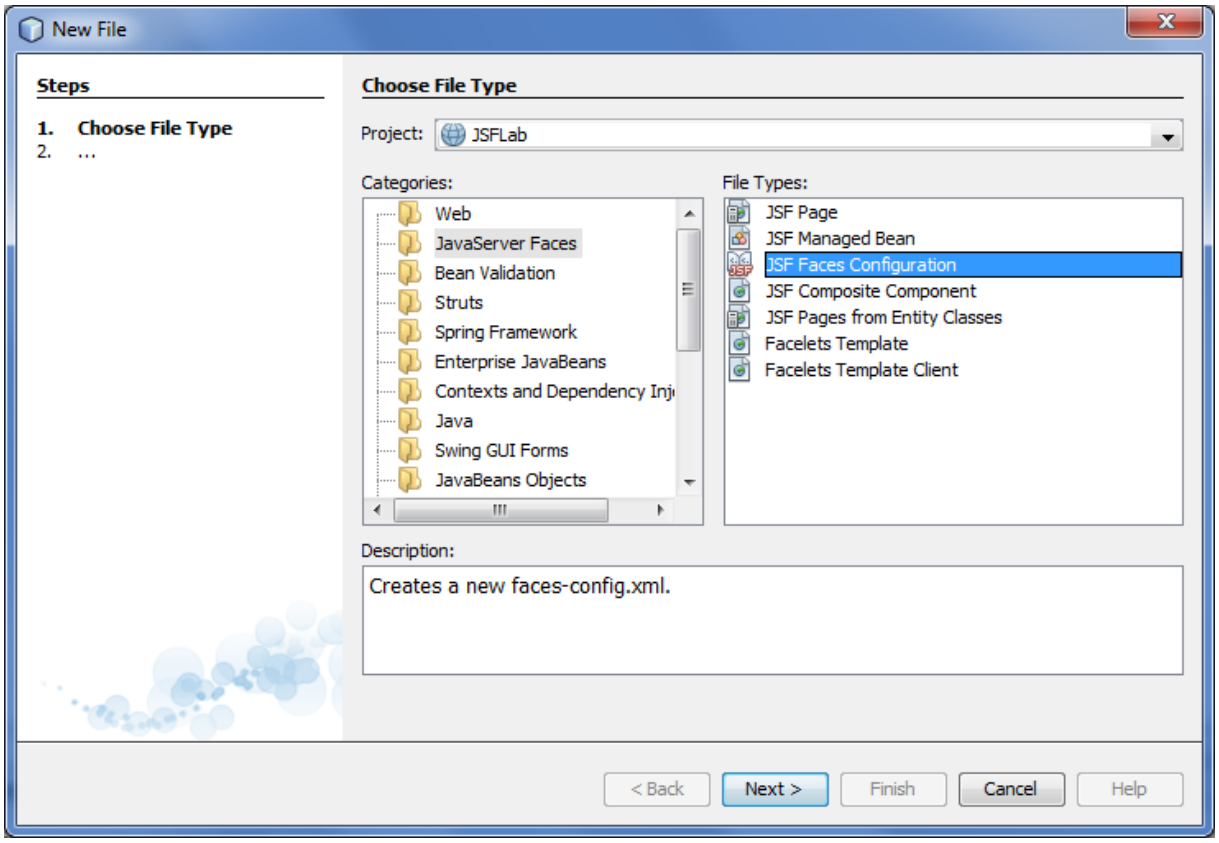
```
required.
javax.faces.validator.LengthValidator.MAXIMUM={1}: Validation Error:
Value is greater than allowable maximum of ''{0}''
javax.faces.validator.LengthValidator.MINIMUM={1}: Validation Error:
Value is less than allowable minimum of ''{0}''
```

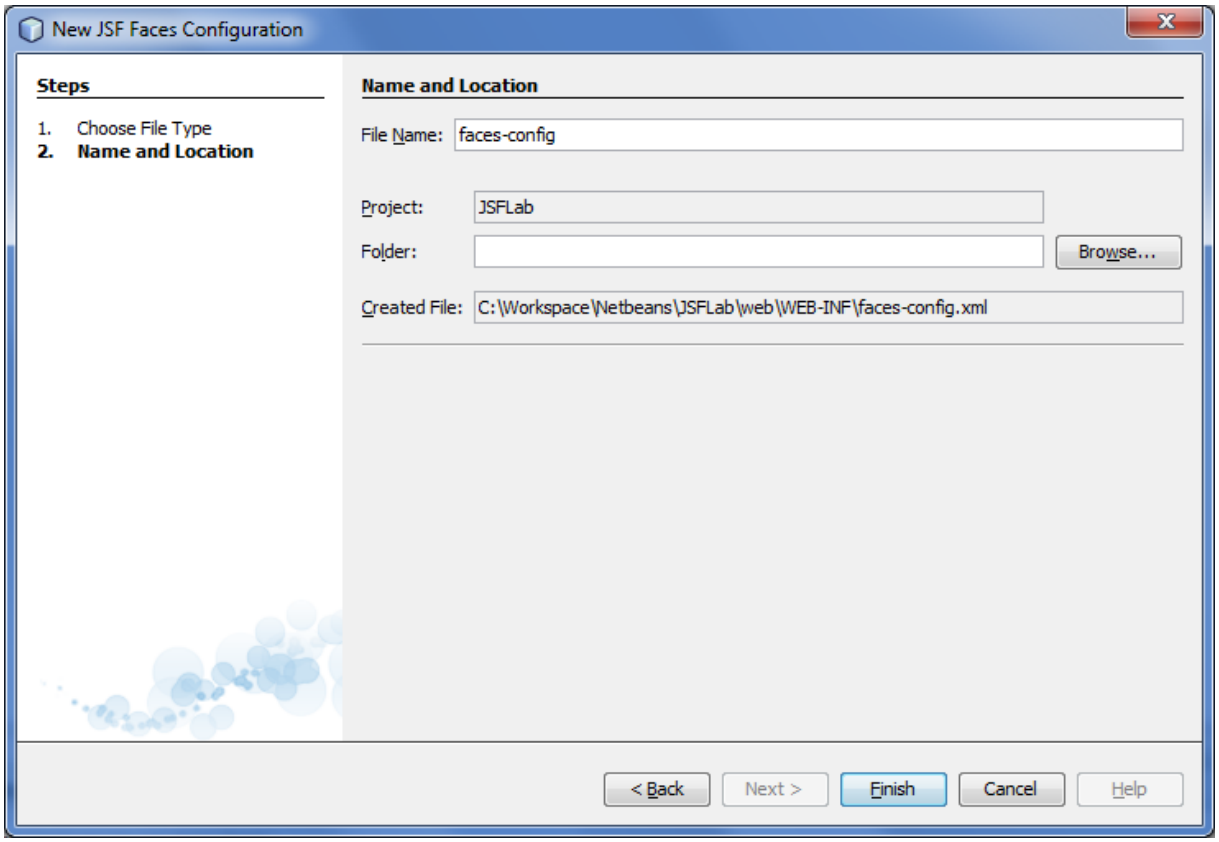
e) Popraw zawartość pliku na poniższą:

```
javax.faces.component.UIInput.REQUIRED=Value is required.
javax.faces.validator.LengthValidator.MAXIMUM=Value is greater than
allowable maximum of ''{0}''
javax.faces.validator.LengthValidator.MINIMUM=Value is less than
allowable minimum of ''{0}''
```

f) Zapisz zmiany.

g) Dodaj w projekcie XML-owy plik konfiguracyjny JSF, uruchamiając z poziomu węzła projektu kreator JSF Faces Configuration z kategorii JavaServer Faces. Pozostaw domyślną zaproponowaną nazwę i lokalizację pliku.





h) Wyświetl tekstową wersję pliku klikając przycisk XML. Następnie w treści pliku `faces-config.xml` umieść poniższy kod, wskazujący utworzony wcześniej plik `properties` jako plik komunikatów aplikacji i przy okazji wskazujący, że domyślnym wspieranym językiem jest angielski.

```
<application>
    <locale-config>
        <default-locale>en</default-locale>
    </locale-config>
    <message-bundle>ErrorMessages</message-bundle>
</application>
```

i) Uruchom i przetestuj aplikację (powinny ulec zmianie teksty komunikatów o błędach walidacji pól).

20. Dodaj w aplikacji obsługę języka polskiego. W tym celu wykonaj poniższe kroki:

a) Dodaj w aplikacji kolejny plik properties (w tej samej lokalizacji co dodany wcześniej plik z komunikatami o błędach). Nazwij go `ApplicationMessages.properties`.

b) W utworzonym pliku umieść poniższe wpisy:

```
label.user=Username
label.password=Password
label.login=Log in
label.back=Back
text.welcome=Welcome
text.sorry=Sorry
title.welcome=Welcome page
title.sorry=Access denied!
title.login=Login page
```

c) W kodzie wszystkich 3 stron JSF dodaj po znaczniku otwierającym `<html>` następujący wiersz z odwołaniem do pliku komunikatów:

```
<f:loadBundle var="AppMessages" basename="ApplicationMessages" />
```

d) W kodzie wszystkich 3 stron JSF zastąp statyczne teksty odwołaniami do kluczy z pliku komunikatów zgodnie z poniższym przykładem:

```
#{AppMessages['title.login']}
```

e) Uruchom i przetestuj aplikację.

f) Dodaj do aplikacji 2 pliki properties o nazwach bazowych `ErrorMessages_pl` i `ApplicationMessages_pl`.

g) Przekopiuj do nowo utworzonych plików komunikaty z dwóch plików utworzonych wcześniej, a następnie przetłumacz treści komunikatów w plikach z przyrostkiem nazwy `_pl` na język polski (klucze komunikatów pozostaw bez zmian!). Uwaga: polskie znaki zastąp ich kodami Unicode:

Ą - \u0104; ą - \u0105
Ć - \u0106; ć - \u0107
Ę - \u0118; ę - \u0119
Ł - \u0141; ł - \u0142
Ń - \u0143; ń - \u0144
Ó - \u00d3; ó - \u00f3
Ś - \u015a; ś - \u015b
Ź - \u0179; ź - \u017a
Ż - \u017b; ż - \u017c

h) W pliku `faces-config.xml` pod wierszem ustawiającym domyślną lokalizację dodaj poniższy wpis mówiący, ze aplikacja wspiera również język polski.

```
            <supported-locale>pl</supported-locale>
```

i) Uruchom i przetestuj aplikację.