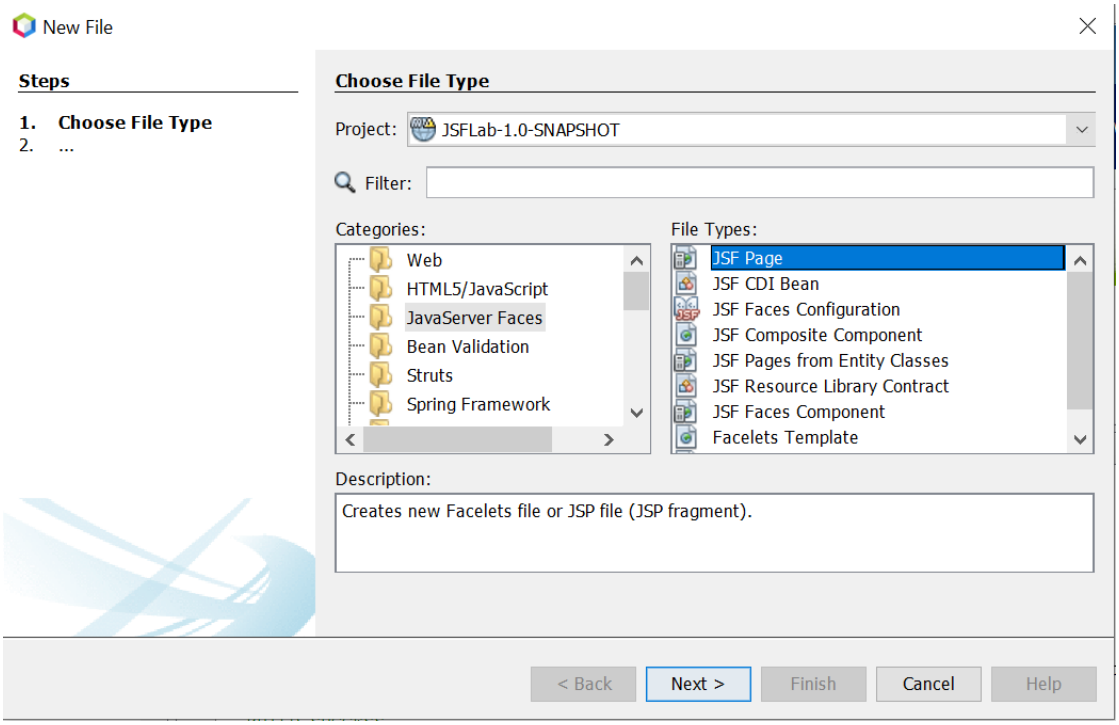
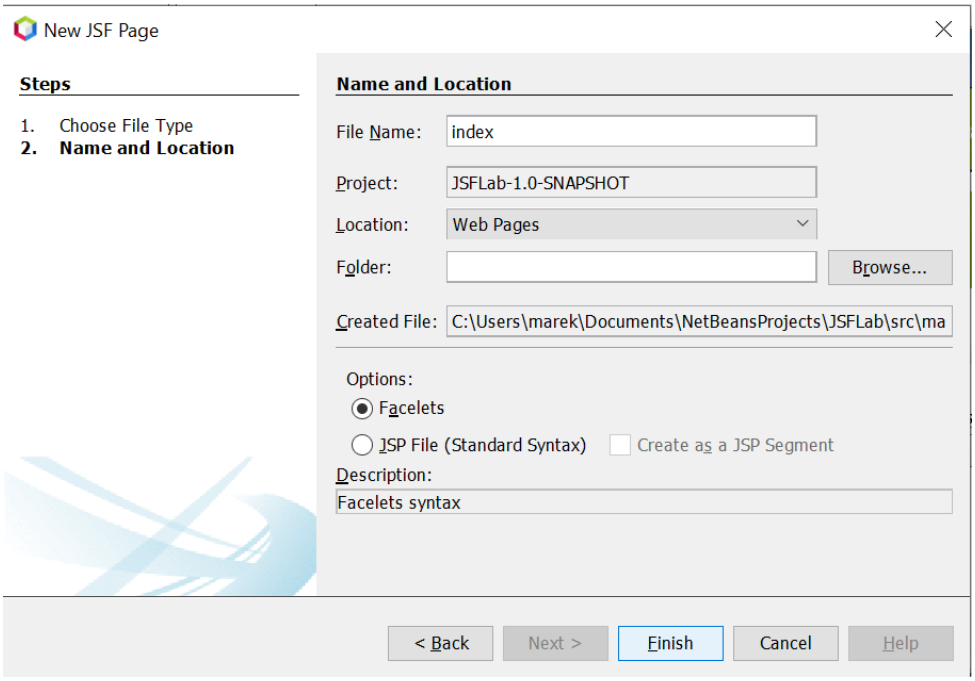
JavaServer Faces (JSF)

Celem ćwiczenia jest przygotowanie prostej aplikacji wykorzystującej technologię JavaServer Faces. Aplikacja umożliwi sprawdzenie poprawności zalogowania się i w zależności od wyniku walidacji wyświetli stosowny dokument. Ćwiczenie obrazuje wykorzystanie podstawowych elementów JSF: komponentów backing bean, deklaratywnego zarządzania nawigacją w serwisie, obsługi wielojęzyczności i walidacji poprawności danych wprowadzanych przez formularz. Do wykonania ćwiczenia potrzebne jest środowisko NetBeans 12.0.

1. Uruchom NetBeans i utwórz nowy projekt. Wybierz File→New Project... i z listy kategorii wybierz Java with Maven, a z listy projektów wybierz Web Application. Kliknij przycisk Next >. Jako nazwę projektu podaj **JSFLab**, wyczyść pole z nazwą pakietu i kliknij przycisk Next >. Wybierz wersję Java EE Java EE 8 Web i upewnij się, że jako serwer aplikacji wybrany jest GlassFish lub Payara (jeśli nie jest dostępny do wyboru, to kliknij Add... i pobierz oraz zainstaluj najnowszą dostępną wersję). Kliknij przycisk Finish.
2. Utwórz w projekcie nową stronę JSF (New File→JavaServer Faces/JSF Page). Jako jej nazwę podaj `index`. Zwróć uwagę aby powstała strona ze składnią Facelets (a nie JSP). Rozszerzenie pliku strony `xhtml` zostanie dodane automatycznie.





3. Otwórz plik `web.xml` i obejrzyj dodaną przez kreator konfigurację serwletu kontrolera zwracając szczególną uwagę na odwzorowanie URL dla tego serwletu.
4. Przejdź do edycji utworzonej 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 zrezygnowali z wizualnego edytora stron JSF (dostępnego w wersji 6.x) 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`.

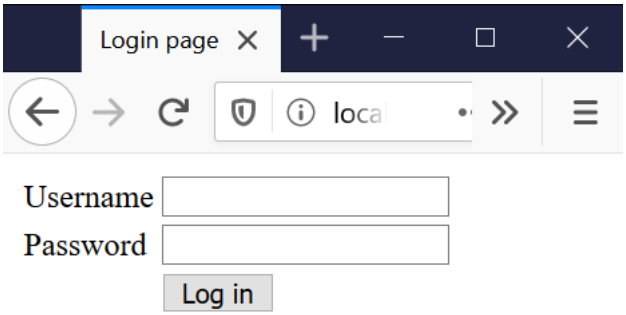
g) Uruchom aplikację. Podejrzyj źródło strony. Zwróć uwagę na elementy HTML odpowiadające komponentom JSF.

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

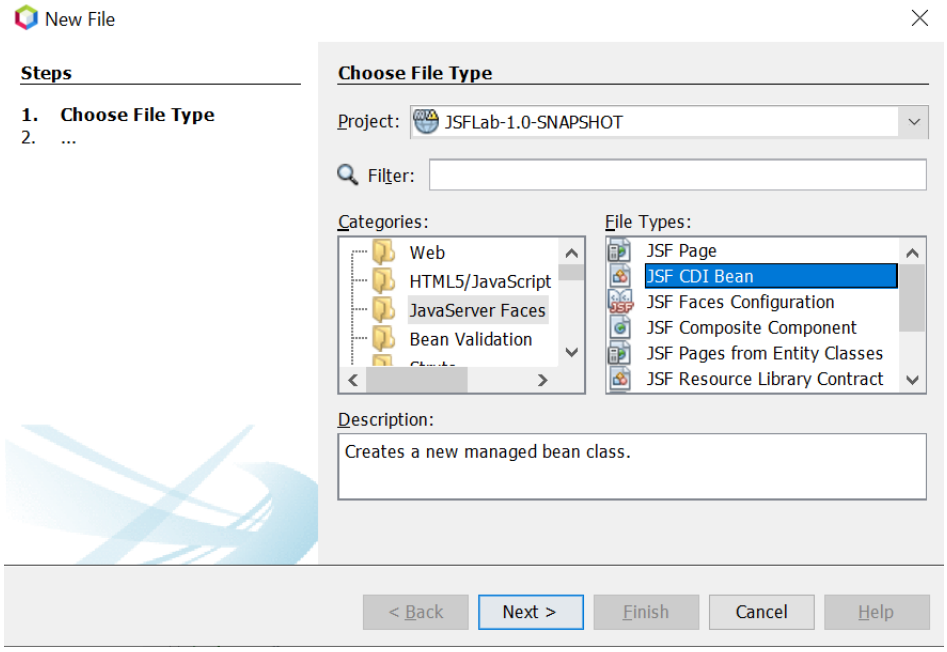
   a) Poniżej pola tekstowego z nazwą użytkownika, ale wewnątrz panelu, umieść etykietę (znacznik `<h:outputLabel>`) i pole do wprowadzania hasła (znacznik `<h:inputSecret>`). Ustaw identyfikator tego pola na `passwordField`. Jako tekst etykiety podaj `Password`. Następnie powiąż etykietę z polem.

   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).

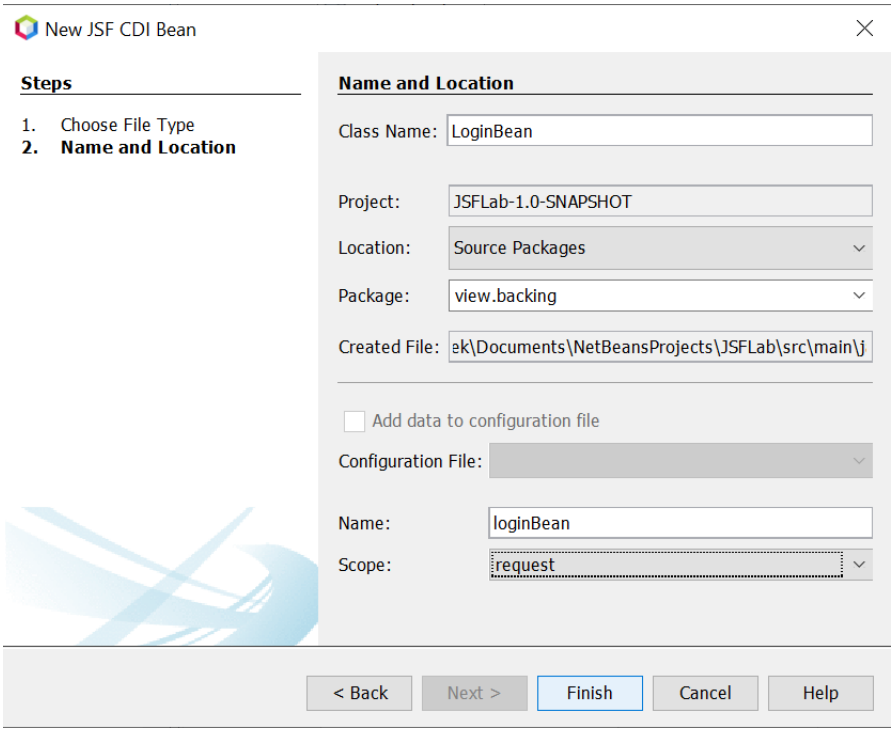
   c) Uruchom stronę i upewnij się, że wygląda w przeglądarce jak na poniższym obrazku. Sprawdź co się dzieje po kliknięciu przycisku.



   d) Utwórz w projekcie nowy komponent CDI (New File→JavaServer Faces/JSF CDI Bean), który będzie pełnił rolę backing bean dla utworzonej przed chwilą strony JSF. W pierwszym kroku kreatora kliknij przycisk Next >.



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



f) Obejrzyj kod wygenerowanej klasy komponentu. Zwróć uwagę na adnotacje CDI nadające komponentowi: zasięg i nazwę pod którą będzie dostępny na stronie.

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

h) Z poziomu edytora kodu wywołaj prawym klawiszem myszy menu kontekstowe i wybierz z niego opcję Refactor → Encapsulate Fields. Zaznacz utworzenie publicznych setterów i getterów dla obu prywatnych pól i kliknij przycisk Refactor.

6. 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`. Jest to nazwa pliku strony, która zostanie utworzona w kolejnym punkcie ćwiczenia.

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

a) W kodzie strony `success.xhtml` zastąp nagłówek i ciało poniższą treścią. 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).

8. 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`.

d) Przejdź do edycji strony `index.html` i w kodzie przycisku zastąp nazwę strony `success` 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` w przypadku gdy podane hasło jest identyczne z nazwą użytkownika, a w przeciwnym wypadku – do nieistniejącej jeszcze strony `failure`. Kod metody może wyglądać jak poniższy:

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

9. 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.

Uruchom i przetestuj aplikację dla poprawnych i niepoprawnych danych logowania.

10. 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. Ponieważ jest to pierwszy element na stronie z rodziny znaczników JSF Core, konieczne będzie zadeklarowanie użycia tej rodziny znaczników w sposób analogiczny do obecnej już deklaracji dla rodziny znaczników JSF HTML.
    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).
11. 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`.
12. 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).

13. 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>
...
```

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

15. 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 klasę zdarzenia.

16. 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>
```

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

18. 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>
```

19. 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?

20. 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();
```

21. Ustaw własne komunikaty dla użytych mechanizmów walidacji:
    a) W komponencie do wprowadzania nazwy użytkownika ustaw atrybut `requiredMessage` na „Username is required.”.
    b) W komponencie do wprowadzania hasła ustaw atrybut `validatorMessage` na „Password must be from 4 to 6 characters long.”.
    c) Uruchom aplikację i sprawdź czy zdefiniowane własne komunikaty faktycznie się wyświetlają.

22. Przygotuj aplikację do obsługi wielu języków poprzez przeniesienie komunikatów, etykiet, itd. Do pliku zasobów. W tym celu wykonaj poniższe kroki:
    a) 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.
    b) W pliku `faces-config.xml` jako element zagnieżdżony w elemencie dokumentu wstaw poniższą zawartość w celu wskazania języka angielskiego jako domyślnego w aplikacji.

```
<application>
    <locale-config>
        <default-locale>en</default-locale>
    </locale-config>
</application>
```

c) Dodaj w projekcie z poziomu węzła Other Sources / src/main/resources plik `properties` korzystając z kreatora Properties File z kategorii Other. Nazwij plik `ApplicationMessages.properties`. (Uwaga: Taką lokalizację pliku narzuca struktura projektu Mavena.)

d) W utworzonym pliku `properties` 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
validation.user= Username is required.
validation.password= Password must be from 4 to 6 characters long.
```

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

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

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

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

g) Uruchom i przetestuj aplikację.

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

a) Dodaj do aplikacji plik properties (w tej samej lokalizacji co poprzednio) o nazwie bazowej `ApplicationMessages_pl`.

b) Przekopiuj do nowo utworzonego pliku komunikaty z pliku utworzonego wcześniej, a następnie przetłumacz treści komunikatów w pliku 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

c) 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>
```

d) Uruchom i przetestuj aplikację.

24. W przypadku wykorzystywania tego samego pliku z komunikatami na wszystkich stronach aplikacji, lepszym rozwiązaniem niż użycie znacznika `<f:loadBundle>` jest rejestracja pliku komunikatów w pliku `faces-config.xml`. Zmodyfikuj aplikację realizując poniższe kroki:

a) W elemencie `<application>` pliku `faces-config.xml` umieść poniższy element `<resource-bundle>`:

```
<resource-bundle>
      <base-name>ApplicationMessages</base-name>
      <var>AppMessages</var>
</resource-bundle>
```

b) Z kodu wszystkich 3 stron JSF usuń niepotrzebny już element `<f:loadBundle>`.

c) Uruchom i przetestuj aplikację.

Zadanie do samodzielnego wykonania

Dodaj do aplikacji programową walidację danych formularza, polegającą na sprawdzeniu czy użytkownik nie podał danych logowania scott/tiger. Jeśli tak, to strona z formularzem logowania powinna zostać wyświetlona ponownie, a nad formularzem powinien być wyświetlony komunikat o błędzie (z wsparciem wielojęzyczności!) informujący, że ten system to nie Oracle.

Wskazówki:

- należy metodą addMessage() obiektu FacesContext programowo dodać komunikat (typu „błąd”) do kontekstu żądania JSF

- tekst komunikatu należy programowo (z poziomu metody komponentu managed bean) pobrać z resource bundle wskazanego w pliku `faces-config.xml`

- komunikat nie powinien być przypisany do żadnego komponentu formularza

- do wyświetlenia komunikatu należy wykorzystać komponent <h:messages>

- należy tak skonfigurować komponent <h:messages>, aby nie wyświetlał komunikatów z walidatorów przypisanych do poszczególnych pól formularza