

## SOA – laboratorium nr 3

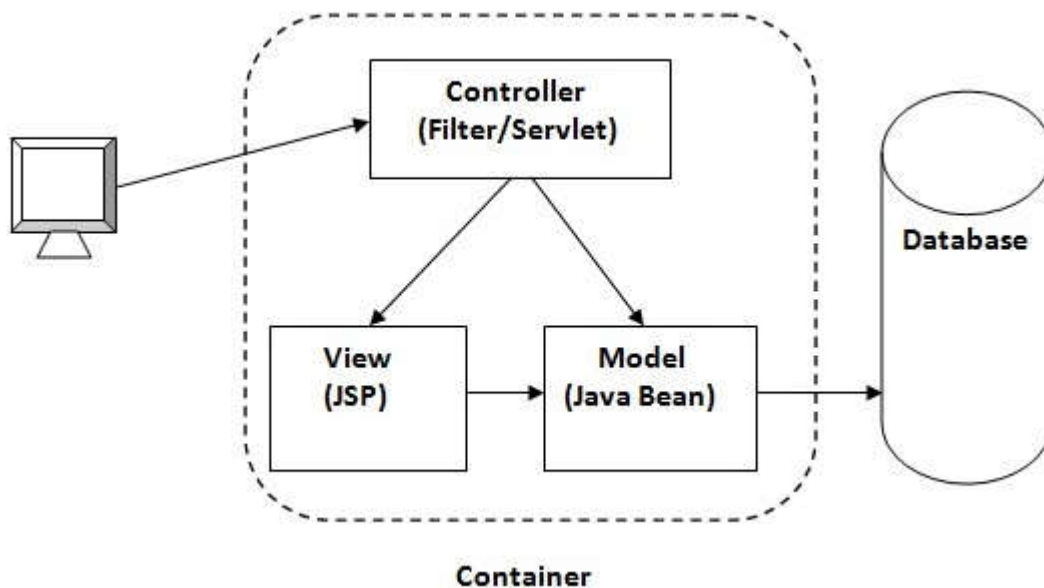
### JavaServer Faces

Celem laboratorium jest nauczenie się projektować warstwę prezentacyjną aplikacji webowych wykorzystującej technologię Java Server Faces (JSF)

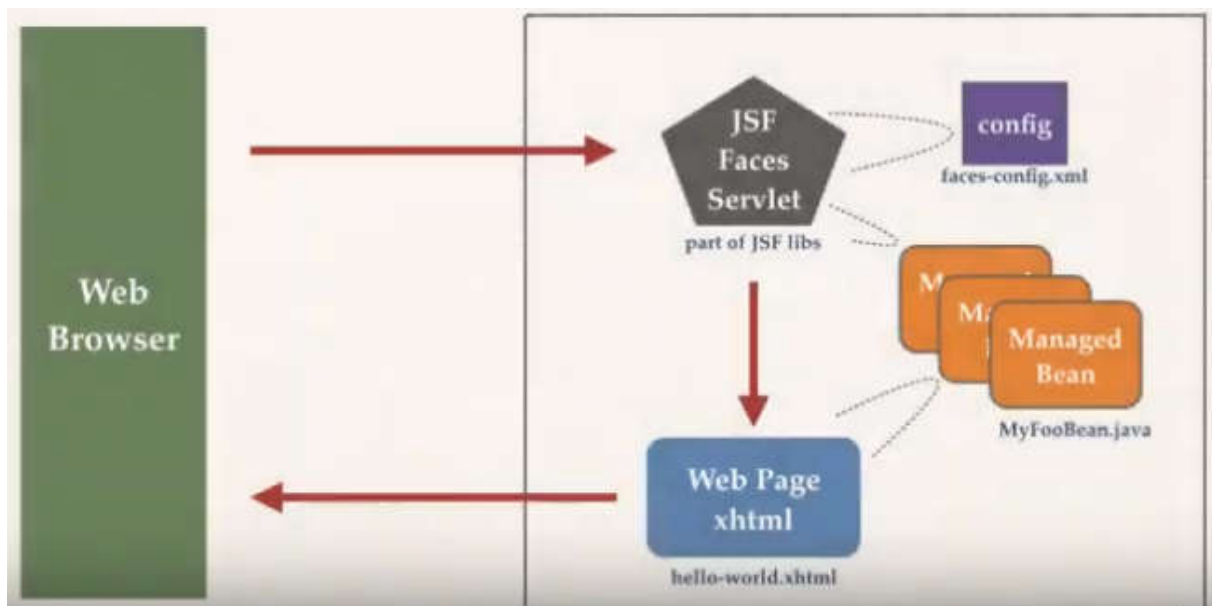
#### 1. Wstęp teoretyczny – przypomnienie najważniejszych informacji

JSF, czyli *Java Server Faces* jest powszechnie używanym frameworkiem o szerokiej funkcjonalności. Jego potencjał jest wykorzystywany do budowania aplikacji, których integralną częścią są złożone interfejsy użytkownika.

Ogólna architektura aplikacji działającej w oparciu o JSF wygląda następująco:



Bardziej szczegółowo wygląda to tak:



## ELEMENTY JSF

- **Komponenty interfejsu użytkownika:** Główny cel, do którego najczęściej jest wykorzystywany JSF to interfejsy użytkownika i złożone formularze, które dzisiaj są chlebem powszednim w aplikacjach webowych.
- **Zarządzanie ziarnami:** Kompatybilność z komponentami JavaBeans w znacznym stopniu automatyzuje i ułatwia współpracę pomiędzy poszczególnymi warstwami aplikacji.
- **Walidacja danych:** Często pożądana funkcjonalność w aplikacjach gdzie wprowadzane dane muszą spełniać pewne restrykcyjne kryteria. Tutaj odpowiednie komponenty mogą nam to umożliwić.
- **Konwersja danych:** Możliwość konwertowania danych do odpowiednich formatów, szczególnie przydatne przy projektach, gdzie musimy przetwarzać dużą ilość danych liczbowych.
- **Dynamiczny nasłuch komponentów:** Wygodne podpinanie zależności pod komponenty, które mają za zadanie reagować w odpowiedni sposób na żądanie klienta. Do tego celu również wykorzystywany jest wcześniej wspomniany AJAX, który jest wspierany przez JSF.
- **Nawigacja:** Elastyczne konfigurowanie relacji pomiędzy warstwami widoku. W łatwy sposób możemy zdefiniować jakie dodatkowe akcje mają się wydarzyć podczas wchodzenia na poszczególne strony.
- **Facelets:** Bardzo istotny element z punktu widzenia wizualnej architektury aplikacji. Często spotykany opis wyglądu, gdzie jeden plik odpowiedzialny za jedną stronę stanowi tak na prawdę kilka plików, w których są zdefiniowane konkretne części danej strony. Takie rozbieżności widoku na parę mniejszych elementów sprawia, że sam kod staje się bardziej przejrzysty i przyjazny modyfikacjom.

## ADNOTACJE

Istotnym elementem podczas pisania aplikacji z użyciem JSF są adnotacje, dzięki którym framework wie jak interpretować daną część kodu. Poniżej kilka podstawowych adnotacji:

**@ManagedBean** - mapuje daną klasę, dzięki temu możemy później odnieść się do pól tej klasy w warstwie widoku aplikacji

**@Named** - tak jak wyżej, jednak ta adnotacja nie zawsze jest kompatybilna z komponentami JSF i czasem się z nimi gryzie, dlatego też bezpieczniejszą opcją jest używanie **@ManagedBean**

**@ManagedProperty** - jak sama nazwa wskazuje odnosi się ona do jakiejś właściwości klasy, najczęściej pola, dzięki tej adnotacji dane pole może zostać wstrzyknięte do innego beana (zmapowanej klasy)

Często używanym mechanizmem podczas pisania aplikacji webowych jest definiowanie żywotności niektórych beanów, głównie stosowane jest to w klasach kontrolerów, odpowiedzialnych za zmianę widoku w aplikacji. Powszechnie ten zabieg jest używany np. w sklepach internetowych bądź innych systemach, gdzie pożądanym jest zapis wprowadzonych danych, które mogą się nam przydać w przyszłości (np. zapamiętywanie loginu i hasła). Poniższe adnotacje określają żywotność beanów:

**@ViewScoped** - podtrzymuje działanie beana, dopóki klient (użytkownik) nie zmieni strony

**@RequestScoped** - żywotność beana ogranicza się do wysłania przez klienta żądania na serwer i 'obumiera' w momencie otrzymania odpowiedzi

**@SessionScoped** - dany bean będzie funkcjonować tak długo, jak sesja HTTP, czyli do opuszczenia strony przez klienta

**@ApplicationScoped** - jak można wywnioskować po nazwie, bean poprzedzony tą adnotacją przestaje działać dopiero w momencie usunięcia aplikacji.

Przydatne linki dotyczące JSF

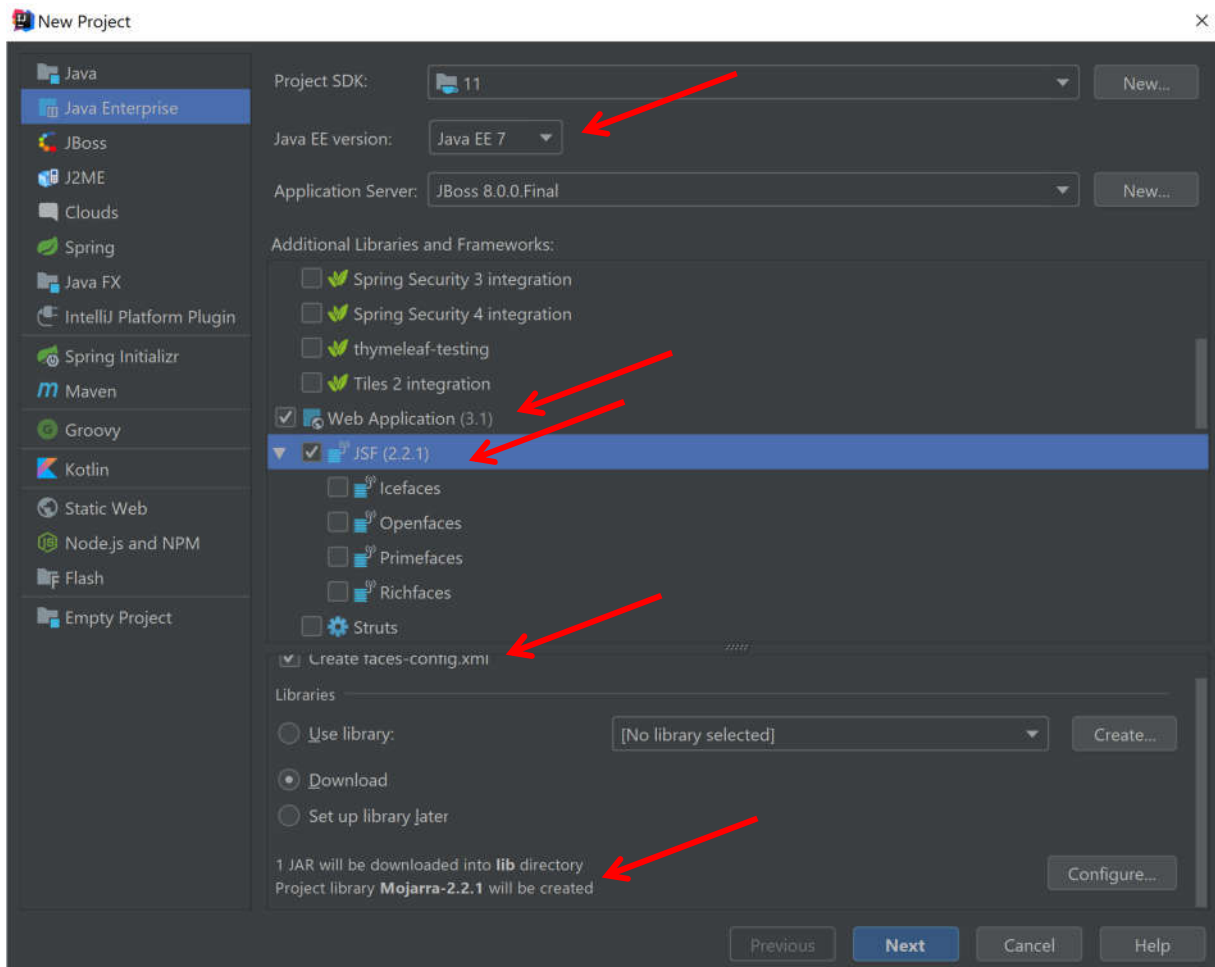
- Strona projektu JSF - [javaserverfaces.java.net](http://javaserverfaces.java.net)
- Dokumentacja techniczna - [javaserverfaces.java.net/docs](http://javaserverfaces.java.net/docs)

## 2. Przykładowa aplikacja z wykorzystaniem JSF

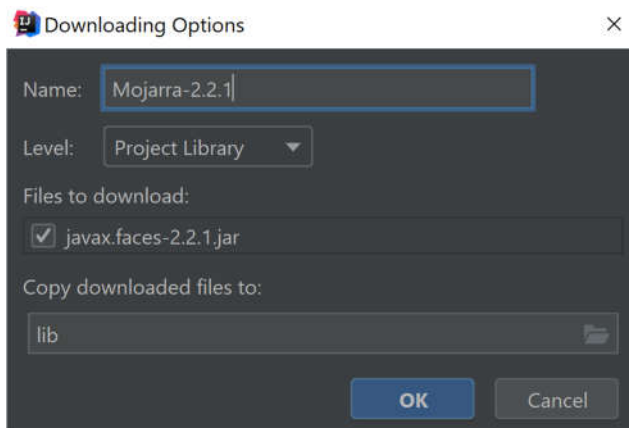
Zaczynamy tworzyć nowy projekt.

Proszę zwrócić uwagę na zaznaczone przez mnie komponenty oraz wersje JavaEE.

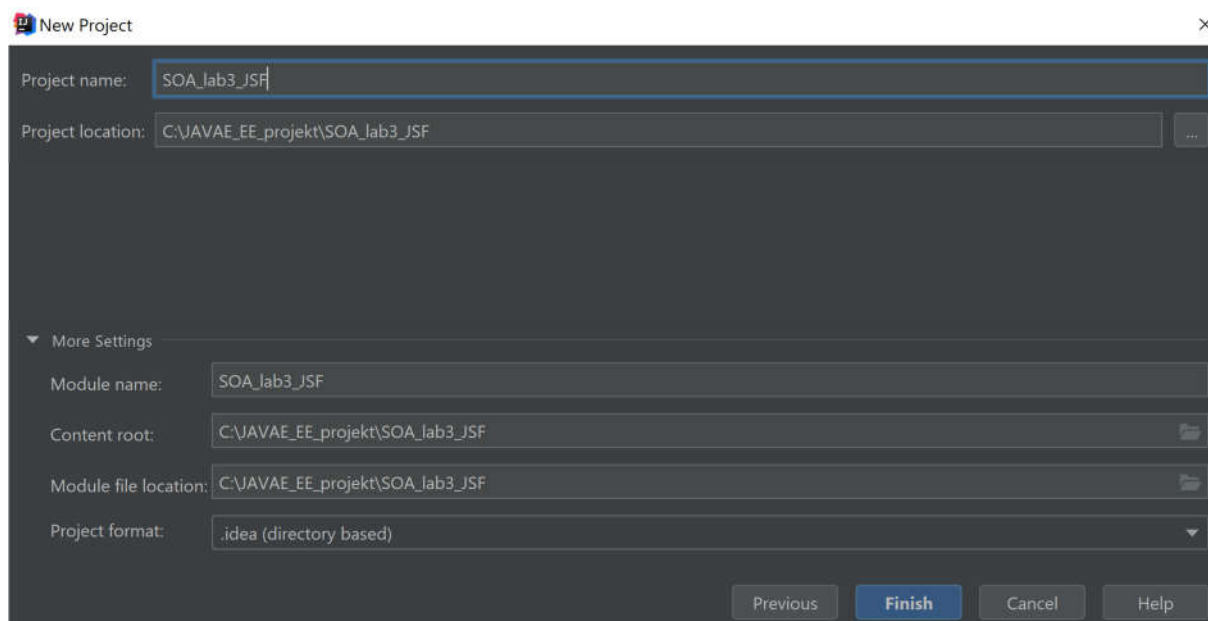
Na dzień tworzenia niniejszego przewodnika (JavaEE 8 i najnowszej wersji JSF 2.4.0 „gryzą się” ze sobą i nie udało się poprawnie deploy na serwer.



Przy okazji można sprawdzić czy są inne bibliotki implementujące specyfikację JSF – naciskając przycisk Configure.



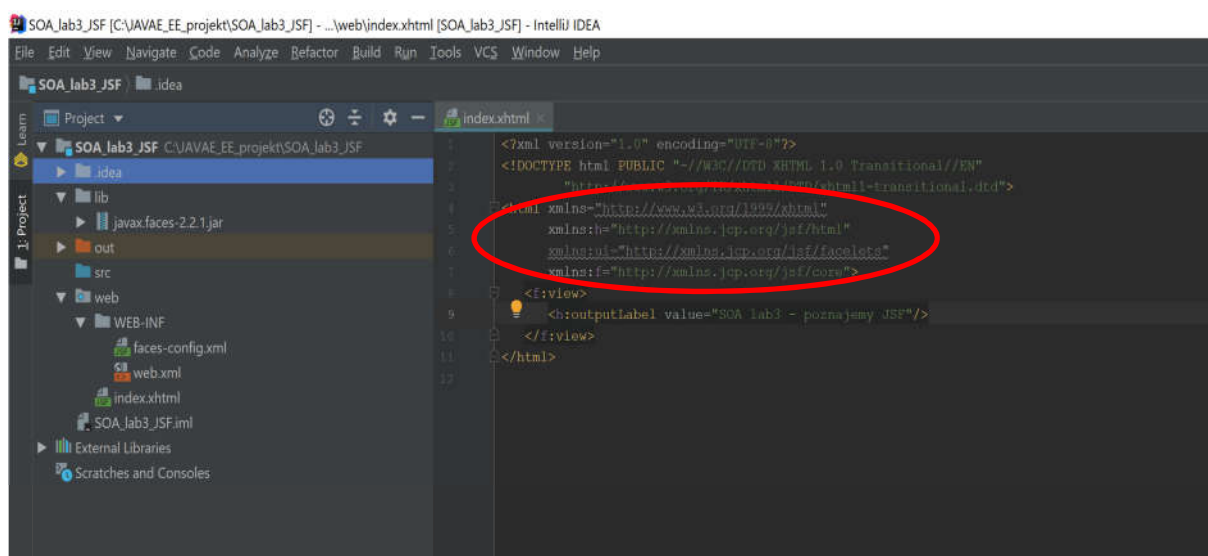
Uzupełniamy metryczkę projektu:



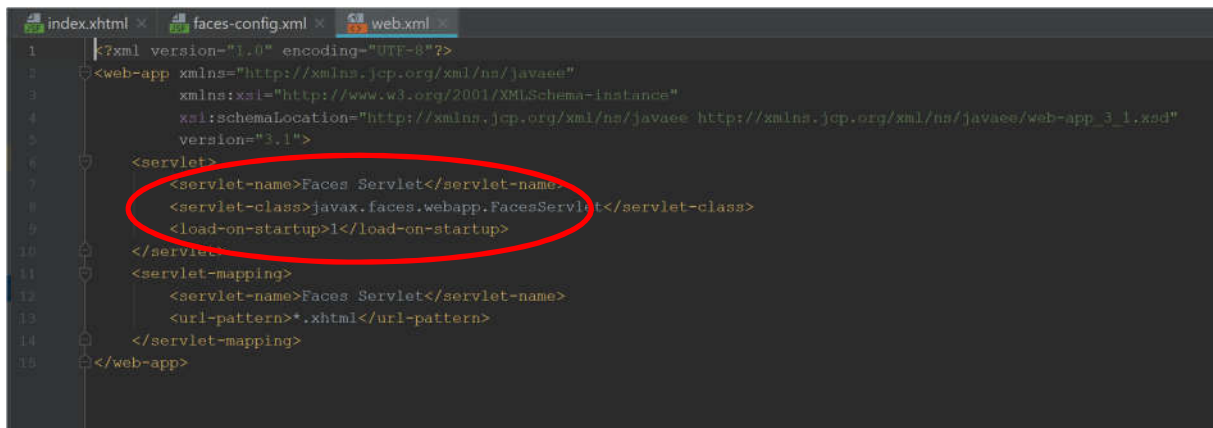
Po stworzeniu struktura projektu powinna wyglądać tak jak poniżej.

Proszę zwrócić uwagę na zaznaczone wpisy – są wymagane do tego aby można było korzystać z JSTL oraz komponentów UI JSF.

Modyfikujemy wpis na stronie personalizując ją troszkę.



Warto odwiedzić też plik face-config.html w celu zaznajomienia z jego strukturą. Widać w nim wyraźnie nazwę servletu pełniącego rolę kontrolera – jest on tworzony automatycznie przez JSF, nie musimy się nim zajmować.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
5         version="3.1">
6     <servlet>
7         <servlet-name>Faces Servlet</servlet-name>
8         <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
9         <load-on-startup>1</load-on-startup>
10    </servlet>
11    <servlet-mapping>
12        <servlet-name>Faces Servlet</servlet-name>
13        <url-pattern>*.xhtml</url-pattern>
14    </servlet-mapping>
15 </web-app>
```

Zweryfikujemy czy wszystko jest poprawnie skonfigurowane – uruchamiając naszą aplikację na serwerze WildFly.

Jak widać poniżej wszystko działa, komponenty UI są poprawnie interpretowane przez bibliotekę i serwer.



SOA lab3 - poznajemy JSF

Rozszerzymy funkcjonalność naszej aplikacji. Naszym celem jest stworzenie aplikacji do sprawdzania czy masz szczęście w grach losowych.

Utwórz dwa nowe pliki xhtml ( wygrana.xhtmll, przegrana.xhtmll) oraz zmodyfikuj plik index.xhtmll zgodnie z poniższymi wytycznymi.

Index.xhtm

```
Wygrana.xhtml x Losowanie.java x Przegrana.xhtml x index.xhtml x
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5 xmlns:h="http://xmlns.jcp.org/jsf/html"
6 xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
7 xmlns:f="http://xmlns.jcp.org/jsf/core">
8
9 <h:head>
10
11 </h:head>
12
13 <h:body>
14 <h1> Sprawdź czy masz szczęście w losowaniu </h1>
15 <h:form>
16 <h:outputLabel> Podaj swoje dane: </h:outputLabel> <br/>
17 PESEL <h:inputText /> <br/>
18 IMIE <h:inputText /> <br/>
19 WIEK <h:inputText /> <br/>
20
21 <h:commandButton value="Wyslij" action="#{Losowanie.wyslij}" ></h:commandButton>
22 </h:form>
23
24 </h:body>
25
26 </html>
27
```

Strona po uruchomieniu powinna wyglądać tak jak poniżej:



## Sprawdz czy masz szczęście w losowaniu

Podaj swoje dane:

PESEL

IMIE

WIEK

Po naciśnięciu przycisku Wyslij powinna się wyświetlic jedna z dwóch stron : Wygrana.xhtml lub Porazka.xhtml w zależności od wyniku losowania.

Strona index.html będzie zarządzana przez odpowiedniego Beana, którego musimy teraz stworzyć:

```
Wygrana.xhtml x Losowanie.java x Przegrana.xhtml x index.xhtml x
1 package pl.agh.kis.soa;
2
3
4 import javax.faces.bean.ManagedBean;
5 import javax.faces.bean.RequestScoped;
6
7 @ManagedBean(name = "Losowanie")
8 @RequestScoped
9 public class Losowanie {
10
11     public String wyslij() {
12         if (Math.random() < 0.2)
13             return "Wygrana";
14         else
15             return "Przegrana";
16     }
17 }
18
19
```

Jest to zwykła klasa Java z odpowiednimi adnotacjami.

Wynik zwracany przez metodę wyslij jest jednocześnie nazwa pliku który będzie wyświetlony w odpowiedzi na żądanie.

Przygotuj zawartość obu stron wynikowych. Strona Porazka.xhtml może wyglądać tak jak poniżej:

```
Wygrana.xhtml x Losowanie.java x Przegrana.xhtml x index.xhtml x
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE html
3     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
4     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
5
6 <html xmlns="http://www.w3.org/1999/xhtml"
7       xmlns:h="http://java.sun.com/jsf/html">
8
9 <h:head>
10     <title>Przegrana</title>
11 </h:head>
12
13 <h:body>
14     <h:form>
15         <h1> Przegrales!</h1>
16         <p> Spróbuj jeszcze raz <h:link value="Kliknij" outcome="index" /> </p>
17     </h:form>
18 </h:body>
19
20 </html>
21
```

Strona wygrana.xhtml jest dużo prostsza:



```
Wygrana.xhtml x Losowanie.java x Przegrana.xhtml x index.xhtml x
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE html
3 PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
4 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
5
6 <html xmlns="http://www.w3.org/1999/xhtml"
7 xmlns:h="http://java.sun.com/jsf/html">
8
9 <h:head>
10 <title>Wygrana</title>
11 </h:head>
12
13 <h:body>
14 <h1> Wygrałeś </h1>
15 </h:body>
16
17 </html>
```

Uruchom aplikację i przetestuj jej działanie. Ja po wykonaniu testu otrzymałem taki o to wynik. Ciekawe dlaczego to zawsze musi być porażka?



## Przegrałeś!

Spróbuj jeszcze raz [Kliknij](#)

Zadanie można rozszerzyć o następujące elementy:

1. Zmień nazwy zwracanych wartości przez metodę wyślij na OK i NOT\_OK. Wykorzystując plik faces-config.xml zdefiniuj nowe zasady nawigacji pomiędzy stronami (w zasadzie takie same jak poprzednio – wygrana OK to wyświetlenie pliku Wygrana.xhtml, przegrana NOT\_OK to Porazka.xhtml).
2. Niech na stronach z wynikami znajdzie się informacja o imieniu i wieku gracza.
3. Weryfikacja wprowadzonych danych – czy zostały podane i czy np. spełniają pewne reguły np. dotyczy to PESEL lub wieku (liczby ujemne lub > 120 lat nie powinny być akceptowane) - zastosuj komponenty walidacyjne.

---

## Praca własna.

### Zadanie 1

Utwórz aplikację, która po uruchomieniu losuje liczbę z przedziału 1-5. Zapisz ją jako zmienną statyczną w klasie obsługującej ManagedBean, np. tak:

```
static int losuj = (int)(Math.random() * 5 + 1);
```

Na stronie głównej (index.xhtml) utwórz 5 przycisków, które powinny przenosić do pięciu stron o nazwach 1.xhtml, 2.xhtml, 3.xhtml, 4.xhtml i 5.xhtml. Na stronach tych pojawia się jej numer.

index.xhtml

1.xhtml

Wciśnięcie przycisku z odpowiednią liczbą przenosi do strony o danym numerze. Jednak, gdy numer strony jest taki sam jak wybranej liczby, wówczas przeniesie na stronę o nazwie trafiony.xhtml, która wyświetla informację o trafieniu liczby wylosowanej przez komputer. Jednocześnie losowana jest nowa liczba. Niech każda ze stron ma licznik odwiedzin wyświetlający aktualną ilość wszystkich odwiedzin na stronie.

## Zadanie 2

Napisz aplikację, która przelicza wartość złotówki na wybraną przez użytkownika walutę i odwrotnie.

Założenia:

1. Aplikacja składa się z 2 części: obliczeniowej oraz administracyjnej.
2. W części obliczeniowej na stronie znajduje się pole tekstowe z możliwością wpisania kwoty, rozwijana lista z nazwą waluty, w jakiej podana została kwota, oraz lista walut, na którą ma zostać dokonane przeliczenie oraz przycisk dokonujący obliczeń i przenoszący na nową stronę w wynikami. (w wersji prostszej listy są generowane statycznie, a wartości pobierane z odpowiedniej klasy. W wersji rozbudowanej do generowania list potrzebna jest druga część programu).
3. Część administracyjna polega na zaprogramowaniu jeszcze jednej strony, na której użytkownik może dodawać waluty, które mają się pojawiać w listach rozwijanych. Domyślnie mają być dwie waluty- PLN i EUR.

Wartość kursu walut ma być średnią wartością ściągniętą z NBP. Skorzystaj z klasy NBPCconnector.

Wartość waluty otrzymuje się wywołując funkcję `exchangeRate("skrótowa_nazwa_valuty")`:  
`String euro = NBPCconnector.exchangeRate("EUR")`.

Uwaga! Wartości waluty ściągnij tylko raz w czasie działania całej aplikacji.

## Zadanie 3

Zbuduj formularz do wyświetlania informacji o wybranym samochodzie w komisie. Formularz zawiera następujące pola:

- Marka samochodu (lista rozwijalna)
- Model (lista rozwijalna)

Zawartość pola model zależy od wybranej marki. Zmiana marki skutkuje zmianą dostępnych w polu model listy modeli.

- pola określające zakres cenowy szukanego samochodu
  - pole wyboru typ silnika ( ON lub benzyna)
  - pola do wprowadzenia swojego imienia oraz numeru telefonu.

oraz przycisk zatwierdzający i czyszczący formularz. Przycisk staje się aktywny dopiero gdy wszystkie pola są poprawnie wypełnione.

Jeśli wszystkie pola zostały poprawnie wypełnione, to po naciśnięciu przycisku akceptacji na dole ekrany wyświetla się lista pojazdów ( marka model cena) ofert spełniających zdefiniowane warunki.

## Zadania do oddania

#### Zadanie 4

Stwórz aplikację ułatwiającą zarządzanie posiadanymi książkami. Książki są wysiedlane w postaci tabelarycznej z następującymi polami do wyświetlenia: tytuł, autor, typ książki (typu wojenna, romans, kryminał itp.), cena zakupu, waluta, ilość stron. Aplikacja powinna umożliwiać dynamiczne filtrowanie wyświetlanych danych po dowolnej ilości kryteriów: np. zakres cenowy i typ waluty. Dodatkowo powinna być możliwość ukrywania wybranych kategorii. Cena zakupu książki powinna umożliwiać wyświetlanie ceny w walucie polskiej lub oryginalnej, jeśli książka była kupiona np. w Euro lub USD Oczywiście przeliczanie odbywało by się globalnie dla całej listy książek. Użytkownik powinien mieć możliwość wyboru/zaznaczenia interesujących go pozycji i po akceptacji wyboru wyświetlić mu podsumowanie zamówienia – ilość pozycji + suma zamówionych książek.

Ze względu na fakt, że nie przerabialiśmy jeszcze współpracy z bazą danych dane niech będą zaszyte w dowolnej kolekcji zczytywanej przez beany.

#### Zadanie 5.

Jesteś już specjalistą w zakresie ankiet, więc otrzymujesz coraz więcej zleceń. Twój kolejny klient poprosił Cię o przygotowanie ankiety dla internetowego sklepu z odzieżą. Ma ona być skierowana zarówno do nowych klientów użytkowników, badając ich preferencje zakupowe, jak również do obecnych klientów, sprawdzając ich poziom zadowolenia z towarów i obsługi. Twoim zadaniem jest zaproponowanie i wykonanie ankiety z uwzględnieniem części wspólnej, która zawiera:

- imię (wymagane)
- adres e-mail (wymagane, poprawny adres)
- wiek (wartość między 10 a 100)
- płeć
- wykształcenie
- wzrost ( dla kobiet w zakresie 150 -185 dla mężczyzn 165-200)

Na podstawie płci należy wyświetlić dodatkowe informacje:  
w przypadku kobiet:

- obwód biustu
- wielkość miseczki
- talia
- biodra

w przypadku mężczyzn:

- klatka
- pas

Dodatkowo po zaakceptowaniu powyższej części formularza wyświetla się dodatkowa sekcja pytań:  
Pytania wspólne dla nowych klientów:

- Ile jesteś w stanie przeznaczyć miesięcznie na zakup ubrania? (Lista wyboru, odpowiedzi: „do 100 zł”, „100-500 zł”, „500-1000 zł”, „powyżej 1000 zł”.)
- Jak często dokonujesz zakupu ubrania? (Lista wyboru, odpowiedzi: „Codziennie”, „Raz w tygodniu”, „Raz w miesiącu”, „Kilka razy w roku”.)
- W jakich kolorach preferujesz ubrania? (Lista wielokrotnego wyboru, odpowiedzi: „Kolorowo-jaskrawych”, „Stonowanych w szarościach”, „W czerni i bieli”, „W samej czerni”.)
- Jakiego rodzaju ubrania najchętniej kupujesz?

W przypadku, gdy klient jest kobietą, pobierz odpowiedź z listy (Lista wielokrotnego wyboru, odpowiedzi: „garsonki”, „bluzki”, „spódniczki”, „spodnie”.)

Gdy mężczyzną: (Lista wielokrotnego wyboru, odpowiedzi: „spodnie”, „spodenki”, „garnitury”, „koszule”, „krawaty”).

Koniecznie sprawdź poprawność wpisywanych danych, a w podsumowaniu ankiety wyświetl informacje wypełnione przez użytkownika. Klient zażyczył sobie również, by na stronie była możliwość losowego wyświetlania reklam wraz z rejestracją ilości kliknięć banera.