



جامعة محمد الخامس بالرباط  
Université Mohammed V de Rabat



# **RAPPORT DU TP 3 : JAVA EE 7**

**Septembre 2020**

Réalisé par :  
**AMANAR Abderrahim**

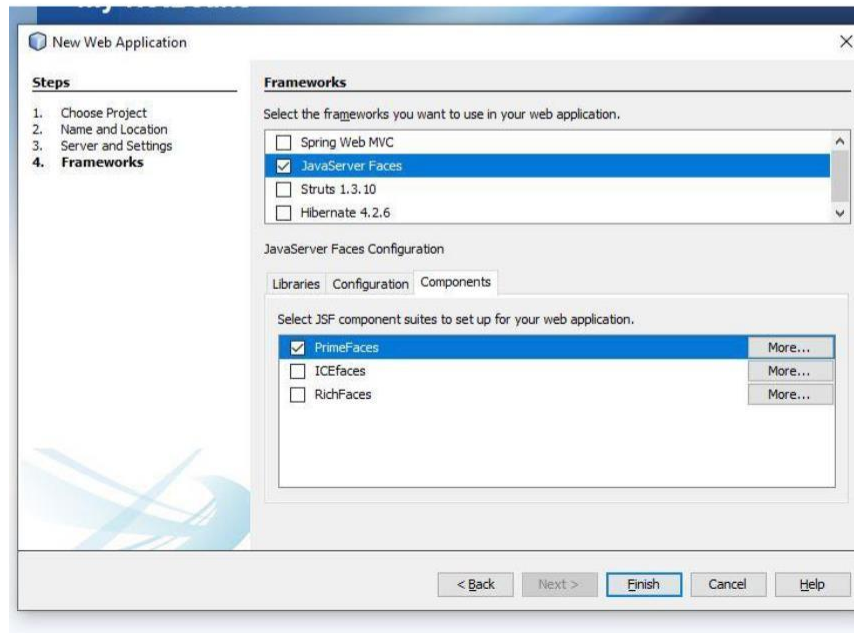
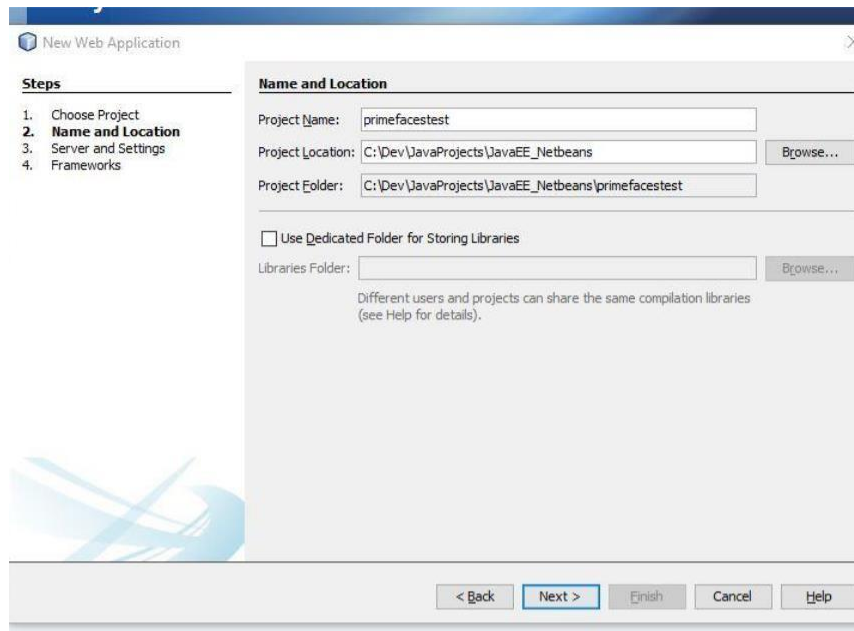
**Master Spécialisé Ingénierie de Données et Développement Logiciel  
ANNEE ACADEMIQUE : 2019/2020**

## Plan du rapport

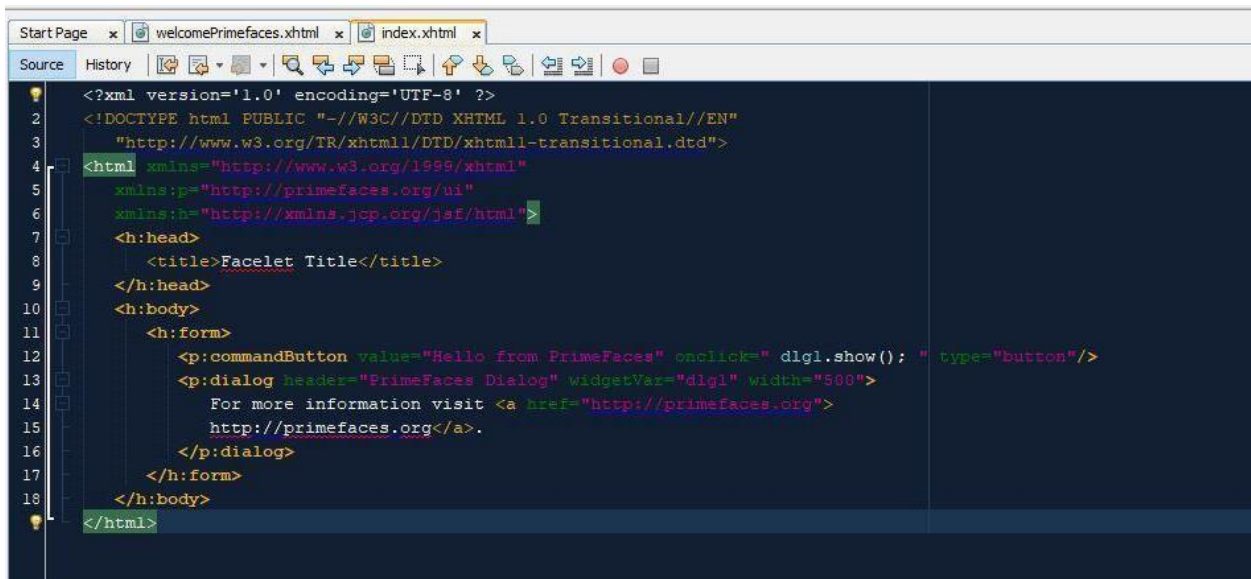
- ✓ Premier projet **PrimeFaces**
- ✓ des Utilisation **composants PrimeFaces** dans des applications **JSF**
- ✓ Les vues par onglets « **Tabbed views** »
- ✓ Assistant des interfaces
- ✓ Les composants **ICEfaces** en JSF
- ✓ La bibliothèque **RichFaces** pour JSF

# I. Premier projet primefaces

**PrimeFaces** nous permet de développer des applications web élégantes avec moins d'effort. PrimeFaces est fourni avec NetBeans 8.0.1. Pour utiliser PrimeFaces dans notre projet, on a besoin de créer un projet Java Web application. Lorsqu'on choisit le **framework JavaServer Faces**, on va cliquer sur l'onglet Composants et sélectionner PrimeFaces comme suite de composants.



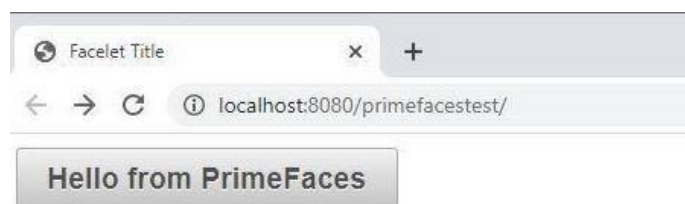
Lorsque le projet est créé, NetBeans va ajouter les bibliothèques de PrimeFaces nécessaires au projet. PrimeFaces est sélectionné comme suite de composants JSF. Le fichier ressemble à ceci :



```
<?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:p="http://primefaces.org/ui"
xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>Facelet Title</title>
  </h:head>
  <h:body>
    <h:form>
      <p:commandButton value="Hello from PrimeFaces" onclick="dlg1.show();" type="button"/>
      <p:dialog header="PrimeFaces Dialog" widgetVar="dlg1" width="500">
        For more information visit <a href="http://primefaces.org">
          http://primefaces.org</a>.
      </p:dialog>
    </h:form>
  </h:body>
</html>
```

Par convention, les balises **PrimeFaces** utilisent le **préfixe p**. Le premier composant PrimeFaces qu'on voit dans notre page est `<p:commandButton>`, ce composant est similaire au composant standard JSF bouton de commande, mais avec certains avantages, tels que le rendu est bien fait sans avoir à appliquer manuellement les feuilles de style CSS.

Lorsqu'on exécute l'application, on peut voir la page générée automatiquement :



Lorsqu'on clique sur le bouton, une boîte de dialogue s'affiche :



**PrimeFaces** nous permet de créer des applications web élégantes avec un moindre effort. Nous allons voir que les composants de PrimeFaces nous facilitent et simplifient beaucoup le développement des applications web.

## II. Utilisation des composants primefaces dans des applications jsf

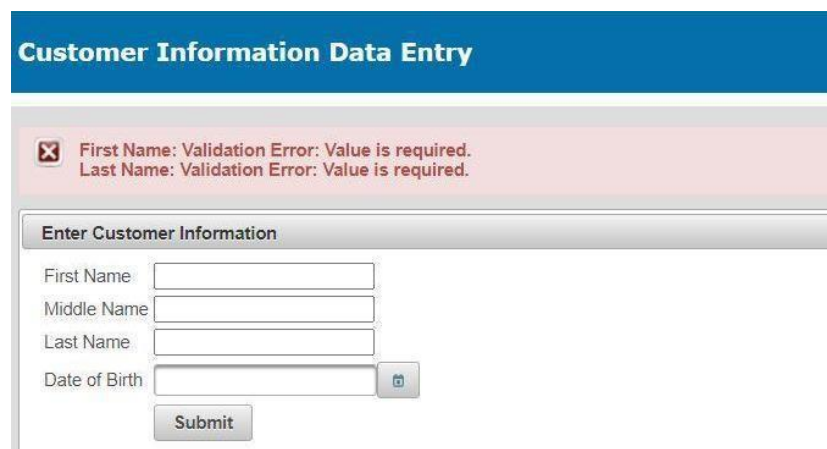
Les composants PrimeFaces nous permet de gagner beaucoup de temps lors du développement de nos applications JSF. La page JSF « **primefacesintro.xhtml** » est une simple page d'entrée de données du client et dans notre exemple, on a utilisé le template facelet « **template.xhtml** », ce qui nous a permet d'obtenir de très bons styles CSS.

Lorsqu'on exécute le projet, le navigateur affiche le résultat suivant :



The screenshot shows a web application titled "Customer Information Data Entry". Below the title is a form with a header "Enter Customer Information". The form contains four input fields: "First Name", "Middle Name", "Last Name", and "Date of Birth". The "Date of Birth" field has a calendar icon. A "Submit" button is located at the bottom of the form.

L'avantage de `<p:messages>` par rapport à `<h:messages>` est que les messages d'erreur sont, par défaut, bien formatés.



The screenshot shows the same "Customer Information Data Entry" form, but now with validation error messages. A red banner at the top of the form area displays two messages: "First Name: Validation Error: Value is required." and "Last Name: Validation Error: Value is required.". The form fields and the "Submit" button are still visible below the messages.

**<p:calendar>** est un nouveau composant de PrimeFaces utilisé pour la saisie d'une date. L'utilisateur peut sélectionner une date en cliquant sur l'icône générée par ce composant. Par défaut, le composant calendrier rend un champ de texte, et quand l'utilisateur clique sur ce champ de texte, l'icône calendrier apparaît. Dans notre exemple, on va mettre l'attribut `showOn` au bouton, ce qui a pour effet de rendre une icône calendrier à côté du champ de texte.

The screenshot shows a web form titled "Enter Customer Information". It contains four input fields: "First Name" with the value "Abderrahim", "Middle Name" (empty), "Last Name" with the value "AMANAR", and "Date of Birth" (empty). A small calendar icon is visible next to the "Date of Birth" field. A date selection calendar is open, showing the month of September 2020. The calendar has a header with "Sep" and "2020". The days of the week are listed as Su, Mo, Tu, We, Th, Fr, Sa. The dates are arranged in a grid, with the 17th of September highlighted in bold.

En cliquant sur le bouton dans notre exemple, l'utilisateur est dirigé vers une page de confirmation.

The screenshot shows a web form titled "Customer Information Confirmation". It contains a section titled "Customer Information" with four fields: "First Name" with the value "Abderrahim", "Middle Name" (empty), "Last Name" with the value "AMANAR", and "Date of Birth" with the value "22/08/1995". The "First Name" field has a red wavy underline underneath the text.

### III. Les vues par onglets «tabbed views»

En général, les formulaires HTML ont plusieurs champs ce qui complique la tâche de l'utilisateur. C'est pourquoi, on les divise en deux ou plusieurs onglets, pour alléger la page. Pour cet effet, PrimeFaces utilise un composant **<p:tabView>** qui permet de générer facilement des onglets. L'exemple suivant illustre comment utiliser ce composant :

```
<?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:p="http://primefaces.org/ui"
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
xmlns:f="http://xmlns.jcp.org/jsf/core">
<ui:composition template="./template.xhtml">
<ui:define name="top">
<h2>Customer Information Data Entry</h2>
</ui:define>
<ui:define name="content">
<h:form>
<p:messages/>
<h:panelGrid columns="1" style="width: 100%">
<p:tabView>
<p:tab title="Personal Information">
<h:panelGrid columns="2">
<h:outputLabel for="firstName" value="First Name" styleClass="requiredLbl"/>
<h:inputText id="firstName" label="First Name" value="#{customer.firstName}" required="true"/>
<h:outputLabel for="middleName" value="Middle Name" styleClass="optionalLbl"/>
<h:inputText id="middleName" label="Middle Name" value="#{customer.middleName}"/>
<h:outputLabel for="lastName" value="Last Name" styleClass="requiredLbl"/>
<h:inputText id="lastName" label="Last Name" value="#{customer.lastName}" required="true"/>
<h:outputLabel for="birthDate" value="Date of Birth" styleClass="optionalLbl"/>
<p:calendar id="birthDate" value="#{customer.birthDate}" showOn="button" navigator="true"/>
</h:panelGrid>
</p:tab>
<p:tab title="Address">
<h:panelGrid columns="2">
<h:outputLabel for="line1" value="Line 1" styleClass="requiredLbl"/>
<h:inputText id="line1" label="Line 1" value="#{customer.line1}" required="true"/>
<h:outputLabel for="line2" value="Line 2" styleClass="optionalLbl"/>
<h:inputText id="line2" label="Line 2" value="#{customer.line2}" required="false"/>
</h:panelGrid>
</p:tab>
</p:tabView>
</h:panelGrid>
</h:form>
</ui:define>
</ui:composition>
</html>
```

Comme on peut le voir dans le code précédent, l'élément racine pour une interface à onglets est **<p:tabView>** à l'intérieur duquel, il doit y avoir un ou plusieurs composants **<p:tab>**. Chaque **<p:tab>** contient les champs de saisie qui correspondent à l'onglet. **<p:tab>** a un attribut **title** rendu comme titre de l'onglet. L'exécution de notre projet affiche la page suivante, et en cliquant sur chaque onglet, on peut voir les composants correspondants.

Un nouveau composant PrimeFaces a été utilisé dans le troisième onglet. **<p:inputMask>** empêche les utilisateurs de mal entrer les données. On l'a utilisé pour les champs de saisie correspondant au numéro de téléphone.

L'utilisation de **<p:inputMask>** permet d'appliquer correctement le formatage des données sans utiliser la validation JSF.



The screenshot shows a web application interface for entering customer information. At the top is a blue header bar with the text "Customer Information Data Entry". Below this is a tabbed interface with three tabs: "Personal Information", "Address", and "Phone Numbers". The "Phone Numbers" tab is currently selected and highlighted. Under this tab, there are three text input fields, each preceded by a label: "Home", "Mobile", and "Work". At the bottom of the form is a "Submit" button.



## IV. Assistant des interfaces

Les assistants sont utiles chaque fois qu'on a besoin des utilisateurs pour remplir des champs de saisie dans un ordre spécifique. On peut utiliser un assistant d'interfaces, lequel peut être réalisé facilement avec l'élément `<p:wizard>` de PrimesFaces.

```
6  <xmlns:ui="http://java.sun.com/jsf/facets">
7
8  <xmlns:f="http://java.sun.com/jsf/core">
9  </xmlns:f>
10 <ui:composition template="./template.xhtml">
11 </ui:composition>
12 <ui:define name="top">
13 <h2>Customer Information Data Entry</h2>
14 </ui:define>
15 <ui:define name="content">
16 <h:form id="form">
17 <h:panelGrid columns="1" style="width: 100%;">
18
19 | <p:wizard id="wizard">
20 |   <p:tab title="Personal Information" id="personalInfo">
21 |     <p:panel header="Personal Information">
22 |       <p:messages/>
23 |       <h:panelGrid columns="2">
24 |         <h:outputLabel for="firstName" value="First Name"
25 |           styleClass="requiredLbl"/>
26 |         <h:inputText id="firstName" label="First Name"
27 |           value="#{customer.firstName}" required="true"/>
28 |         <h:outputLabel for="middleName" value="Middle Name"
29 |           styleClass="optionalLbl"/>
30 |         <h:inputText id="middleName" label="Middle Name"
31 |           value="#{customer.middleName}" />
32 |         <h:outputLabel for="lastName" value="Last Name"
33 |           styleClass="requiredLbl"/>
34 |         <h:inputText id="lastName" label="Last Name" value="#{customer.lastName}" required="true"/>
35 |         <h:outputLabel for="birthDate" value="Date of Birth"
36 |           styleClass="optionalLbl"/>
37 |         <p:calendar id="birthDate"
38 |           value="#{customer.birthDate}" showOn="button" navigator="true"/>
39 |       </h:panelGrid>
40 |     </p:panel>
41 |   </p:tab>
42 </p:wizard>
43 </h:form>
44 </ui:define>
45 </ui:composition>
46 </pre>
```

Pour générer un assistant d'interface avec PrimeFaces, on doit utiliser le composant `<p:wizard>`, puis lui imbriquer un composant `<p:tab>` pour chaque étape de l'assistant.

Dans le dernier tab, on a ajouté une balise `<p:commandButton>` pour soumettre les données au serveur. La valeur de l'attribut `ActionListener` de `CommandButton` est une méthode du bean géré `CustomerController`.

```
1 package fsr;
2 import java.io.Serializable;
3 import javax.enterprise.context.SessionScoped;
4 import javax.inject.Named;
5 import javax.faces.event.ActionEvent;
6 import javax.faces.application.FacesMessage;
7 import javax.faces.context.FacesContext;
8 @SessionScoped
9 @Named("customerController")
10 public class CustomerController implements Serializable {
11     /** Creates a new instance of CustomerController */
12     public CustomerController() {}
13     public void saveCustomer(ActionEvent actionEvent) {
14         //In a real application, we would save the data,
15         //In this example we simply show a message.
16         FacesMessage facesMessage = new FacesMessage("Data Saved Successfully");
17
18         facesMessage.setSeverity(FacesMessage.SEVERITY_INFO);
19         FacesContext.getCurrentInstance().addMessage(null, facesMessage);
20     }
21 }
```

La méthode « saveCustomer() » ajoute un simple message qui sera affiché dans la page de confirmation. Dans une application réelle, il faut sauvegarder les données dans une base de données.

Lorsqu'on exécute le projet, le navigateur affiche le résultat suivant :

The screenshot shows a web form titled 'Personal Information' within a wizard interface. At the top, there are three tabs: 'Personal Information' (active), 'Phone Numbers', and 'Confirmation'. Below the tabs, the form contains the following fields:

- First Name:
- Middle Name:
- Last Name:
- Date of Birth:  with a calendar icon to its right.

At the bottom right of the form, there is a button labeled '→ Next'.

Le **composant Assistant** ajoute automatiquement un bouton Suivant en bas à droite, le fait de cliquer sur ce bouton nous amène à la page suivante de l'assistant.

The screenshot shows the next step in the wizard, titled 'Phone Numbers'. The tabs at the top are 'Personal Information', 'Phone Numbers' (active), and 'Confirmation'. The form contains the following fields:

- Home:
- Mobile:
- Work:

At the bottom left, there is a button labeled '← Back', and at the bottom right, there is a button labeled '→ Next'.

L'assistant génère à la fois un bouton précédent et un bouton suivant. Une fois on atteint la dernière page, on clique sur le bouton submit on voit le message de confirmation généré

par la méthode « saveCustomer() » de notre bean géré "CustomerController". Ce message est rendu par le composant `<p:messages>`, qui fournit un bon style, sans utiliser les CSS.

**Customer Information Data Entry**

Personal Information

Phone Numbers

Confirmation

First Name

Abderrahi

Middle Name

Last Name

AMANAR

Date of Birth

20/07/1996

Home Phone

Mobile Phone

Work Phone

Submit

← Back


Si on clique sur le bouton « submit » on obtient :

**Customer Information Data Entry**

Personal Information

Phone Numbers

Confirmation

 Data Saved Successfully

First Name

Abderrahim

Middle Name

Last Name

AMANAR

Date of Birth

20/07/1996

Home Phone

Mobile Phone

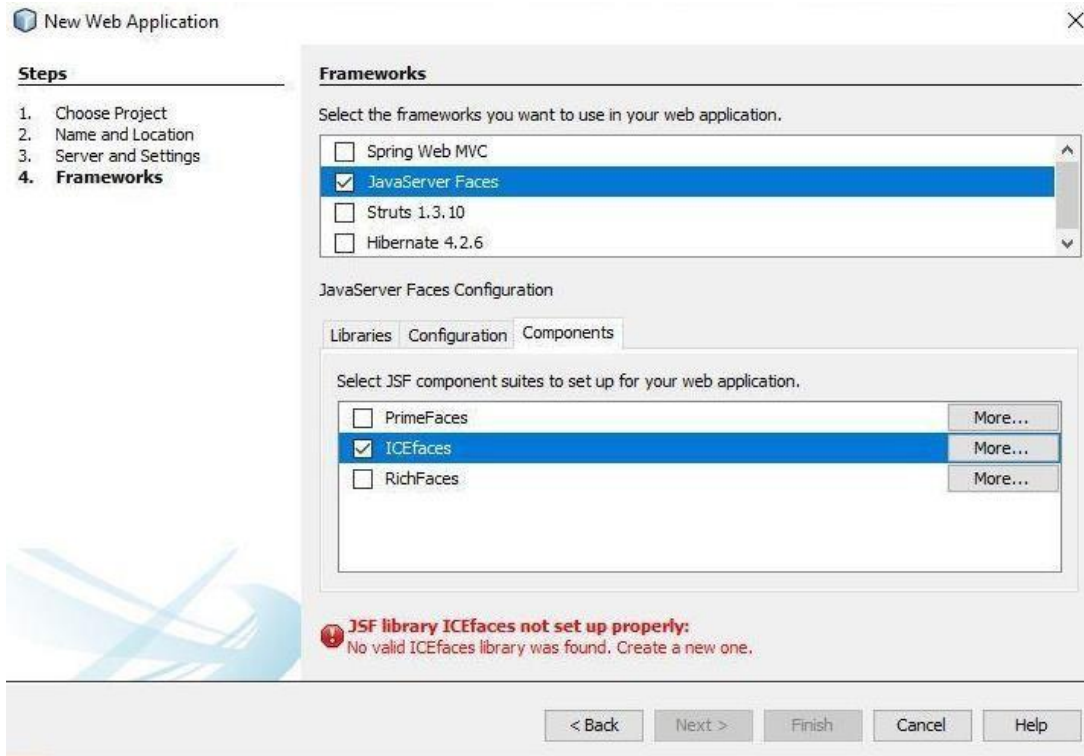
Work Phone

Submit

← Back

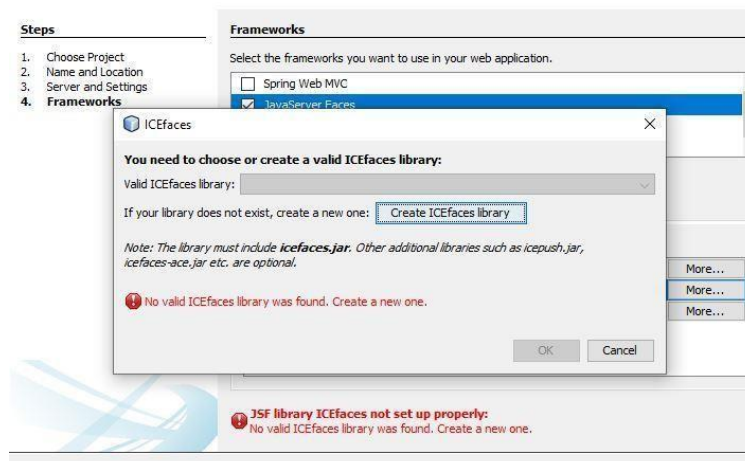
## V. LES COMPOSANTS ICEFACES EN JSF

**ICEfaces** est une autre **librairie JSF** qui simplifie le développement des applications JSF. Pour utiliser ICEfaces dans une application JSF, créer un nouveau projet web application. Lorsqu'on sélectionne JSF comme Framework, on clique sur l'onglet Components et on coche l'option ICEfaces.



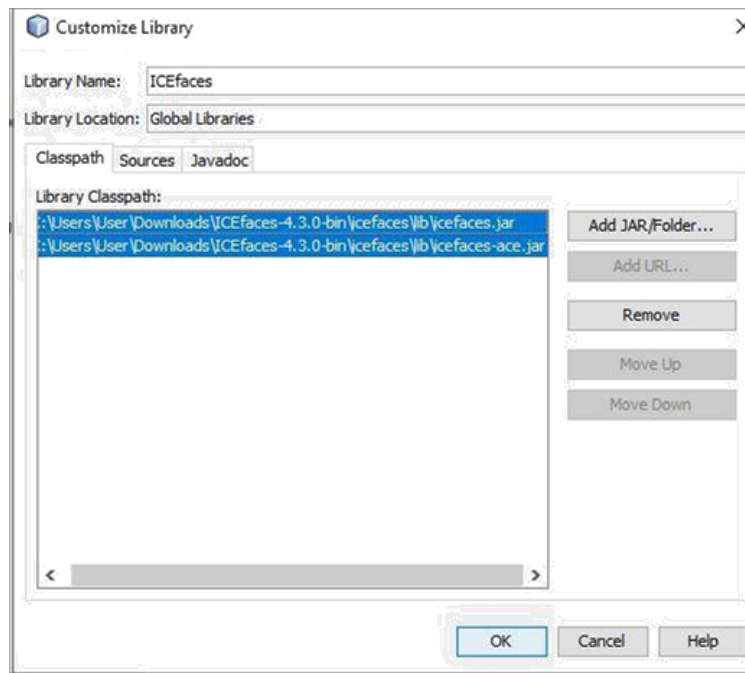
Contrairement à PrimeFaces, la librairie **ICEfaces** n'est intégrée dans NetBeans. Il faut la télécharger à partir de <http://www.icesoft.org/java/downloads/icefaces-downloads.jsf> et créer une nouvelle librairie.

Pour créer une nouvelle librairie ICEfaces dans NetBeans, on clique sur « **More...** » bouton à côté la case icefaces. La boîte suivante s'affiche :



Après avoir cliqué sur ok, on doit localiser les fichiers jar de **ICEfaces** puis on clique sur le bouton « Add JAR/Folder... » pour les ajouter à notre librairie.

Après avoir cliqué sur ok, NetBeans crée la librairie ICEfaces.



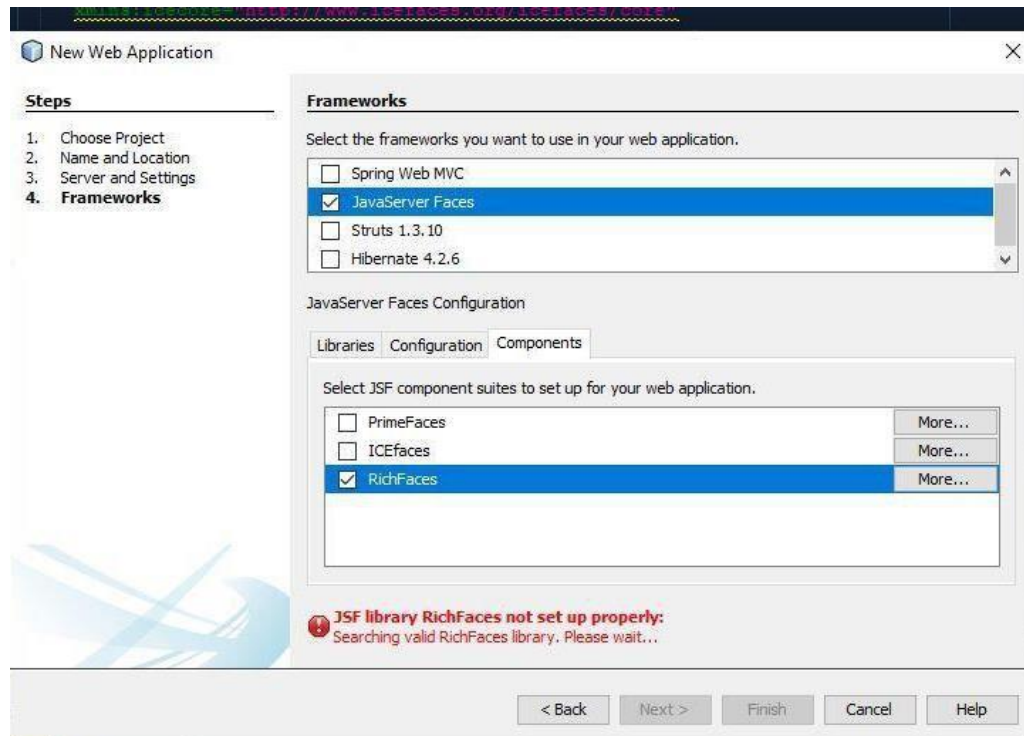
Comme pour PrimeFaces, NetBeans génère une application **ICEfaces** exemple qu'on peut utiliser comme point de démarrage.



## VI. La bibliotheque richfaces pour jsf

On peut démarrer une application java web et choisir **RichFaces** comme sources des composants. On doit télécharger et créer une librairie pour NetBeans.

Allez dans le site <http://www.jboss.org/richfaces/download/stable.html> et choisir la dernière version stable.



Les fichiers à ajouter à la librairie NetBeans sont comme suit:

- **richfaces-a4j-4.5.4.Final.jar**
- **richfaces-rich-4.5.4.Final.jar**
- **richfaces-core-4.5.4.Final.jar**

RichFaces a aussi certaines dépendances externes

- **guava-18.0.jar**
- **sac-1.3.jar**
- **cssparser-0.9.14.jar**

On crée alors la librairie **RichFaces** pour NetBeans comme suit :



