



Java Server Faces

INTRODUCTION

Java server faces est un framework de développement web qui permet de créer des applications web dynamiques.

il permet :

- ☕ de créer des pages web dynamiques
- ☕ la mise en place d'une architecture mvc
- ☕ de faciliter l'utilisation des ressources du serveur
- ☕ la création de composants réutilisables
- ☕ le support de l'internationalisation
- ☕ la possibilité de créer ses propres composants



A QUOI SERT JAVA FACES ?



Le framework a été créé pour répondre à un besoin simple :

☕ faciliter la création d'application web dynamique

Il a été créé par Sun Microsystems en 2003 et est devenu un standard de l'industrie.

Il est maintenant géré par la fondation Eclipse.

Sa dernière version est la 2.3, sortie en 2017.

JAVA FACES ET JAVA PAGES



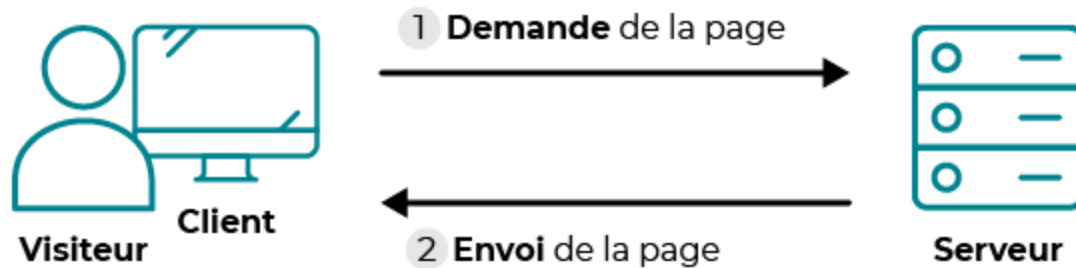
Java Server Pages est un framework de développement web qui permet de créer des pages web statiques.

Il a été utilisé pendant de nombreuses années pour créer des pages web, mais il a été progressivement remplacé par Java Faces.

De plus, Java Server Pages est déprécié depuis 2017.

PAGE WEB STATIQUE

Une page web statique est une page web qui ne change pas.
Elle est écrite en html et ne contient pas de code java.
Peux importe les actions de l'utilisateur, la page restera la même.
exemple : Un cours de mathématiques en ligne.



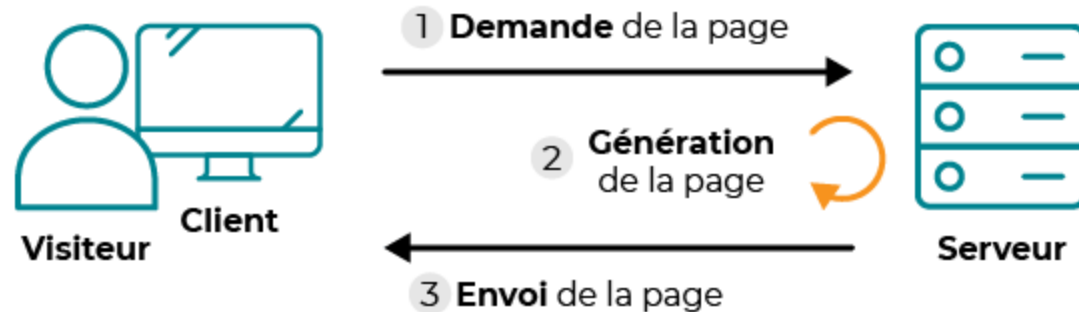
PAGE WEB DYNAMIQUE

Une page web dynamique est une page web qui change en fonction des actions de l'utilisateur.

Elle est écrite en html et contient du code java.

Elle peut envoyer des requêtes au serveur et recevoir des réponses.

exemple : Un site de vente en ligne.



ARCHITECTURE DE JAVA FACES



☕ Le client : ?

☕ Le serveur : ?

☕ Controller : ?

☕ Model : ?

☕ View : ?

ARCHITECTURE DE JAVA FACES



☕ **Le client** : c'est le navigateur de l'utilisateur.

☕ **Le serveur** : ?

☕ **Controller** : ?

☕ **Model** : ?

☕ **View** : ?

ARCHITECTURE DE JAVA FACES



☕ **Le client** : c'est le navigateur de l'utilisateur.

☕ **Le serveur** : c'est le serveur web qui héberge l'application.

☕ **Controller** : ?

☕ **Model** : ?

☕ **View** : ?

ARCHITECTURE DE JAVA FACES



- ☕ **Le client** : c'est le navigateur de l'utilisateur.
- ☕ **Le serveur** : c'est le serveur web qui héberge l'application.
- ☕ **Controller** : permet de gérer les requêtes et les réponses.
- ☕ **Model** : ?
- ☕ **View** : ?

ARCHITECTURE DE JAVA FACES



- ☕ **Le client** : c'est le navigateur de l'utilisateur.
- ☕ **Le serveur** : c'est le serveur web qui héberge l'application.
- ☕ **Controller** : permet de gérer les requêtes et les réponses.
- ☕ **Model** : c'est le modèle qui permet de gérer les données.
- ☕ **View** : ?

ARCHITECTURE DE JAVA FACES

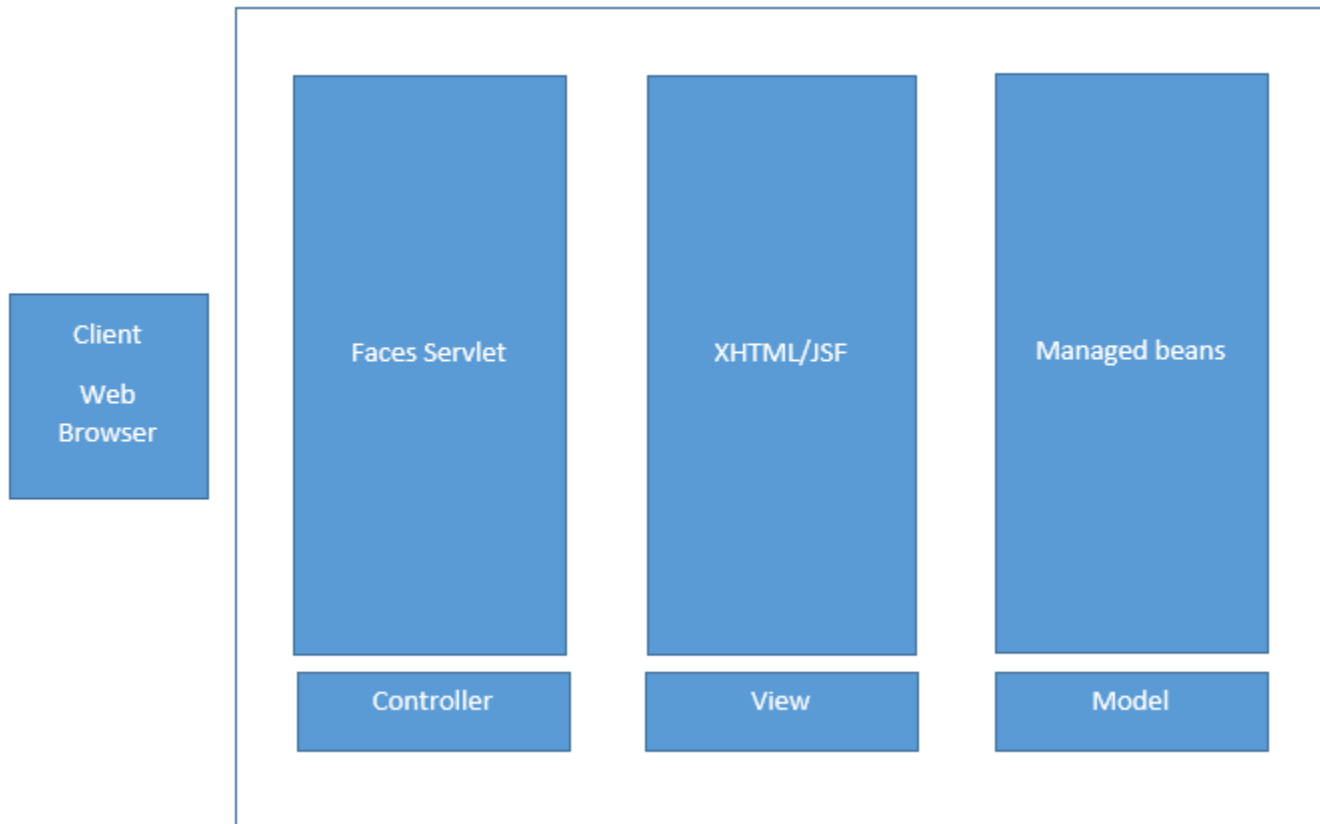


- ☕ **Le client** : c'est le navigateur de l'utilisateur.
- ☕ **Le serveur** : c'est le serveur web qui héberge l'application.
- ☕ **Controller** : permet de gérer les requêtes et les réponses.
- ☕ **Model** : c'est le modèle qui permet de gérer les données.
- ☕ **View** : c'est la vue qui permet de gérer l'affichage.

Dans le cas de jsf, le controller est géré par le framework et le model est géré par l'utilisateur.

ARCHITECTURE DE JAVA FACES

L'architecture de jsf est construite de la manière suivante :



L'ARCHITECTURE MVC SOUS JSF



Une fois l'architecture MVC représentée, il est facile de comprendre comment jsf fonctionne.

jsf implémente la partie controller et la partie view de l'architecture MVC.

Il gère les requêtes et les réponses et affiche les pages web.

Il me nous reste plus qu'à créer le model.

INSTALLATION DE JSF



Pour utiliser jsf, il faut :

- ☕ avoir un serveur web (tomcat, glassfish, ...)
- ☕ avoir un ide java ee (netbeans, eclipse, ...)
- ☕ avoir un driver jdbc pour la base de données

INSTALLATION



Dans le cadre de ce cours, nous allons utiliser :

- ☕ un serveur web : tomcat
- ☕ un ide java ee : eclipse version 2019-06 java ee
- ☕ jdk 17 (veillez à bien installer le jdk et non le jre)

INSTALLATION DE TOMCAT



Pour installer tomcat, il faut :

- ☕ télécharger tomcat : <https://tomcat.apache.org/download-90.cgi>
- ☕ décompresser l'archive
- ☕ lancer le fichier startup.bat

INSTALLATION DE ECLIPSE



Pour installer eclipse, il faut :

- ☕ télécharger eclipse : <https://www.eclipse.org/downloads/>
- ☕ décompresser l'archive
- ☕ lancer eclipse.exe

INSTALLATION DE JDK



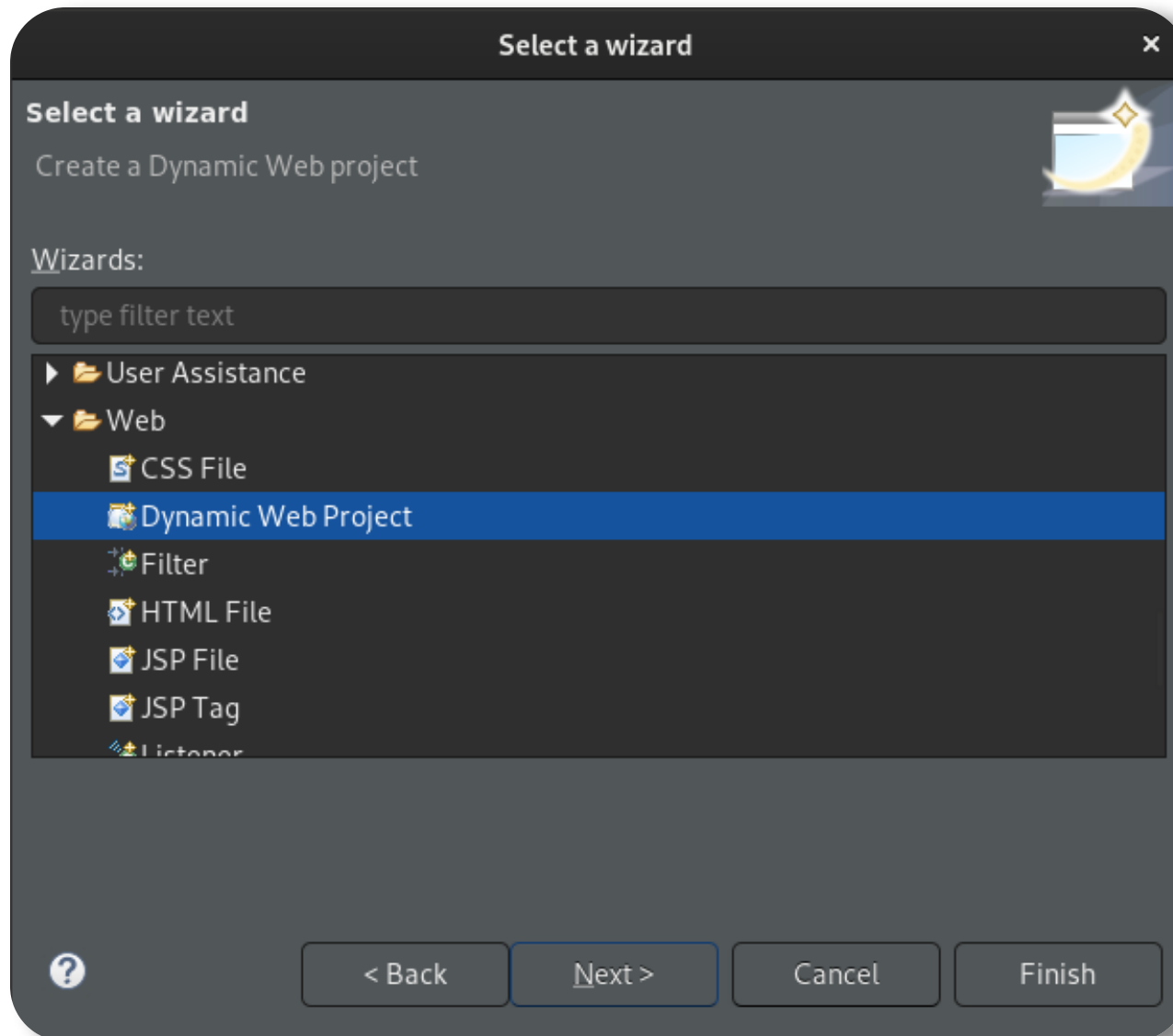
Pour installer jdk, il faut :



aller chercher le jdk :

<https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html>

CRÉATION D'UN PROJET



New Dynamic Web Project

Dynamic Web Project
Create a standalone Java-based Web Application or add it to a new or existing Enterprise Application.

Project name:

Project location
☒ Use default location
Location:

Target runtime

Dynamic web module version

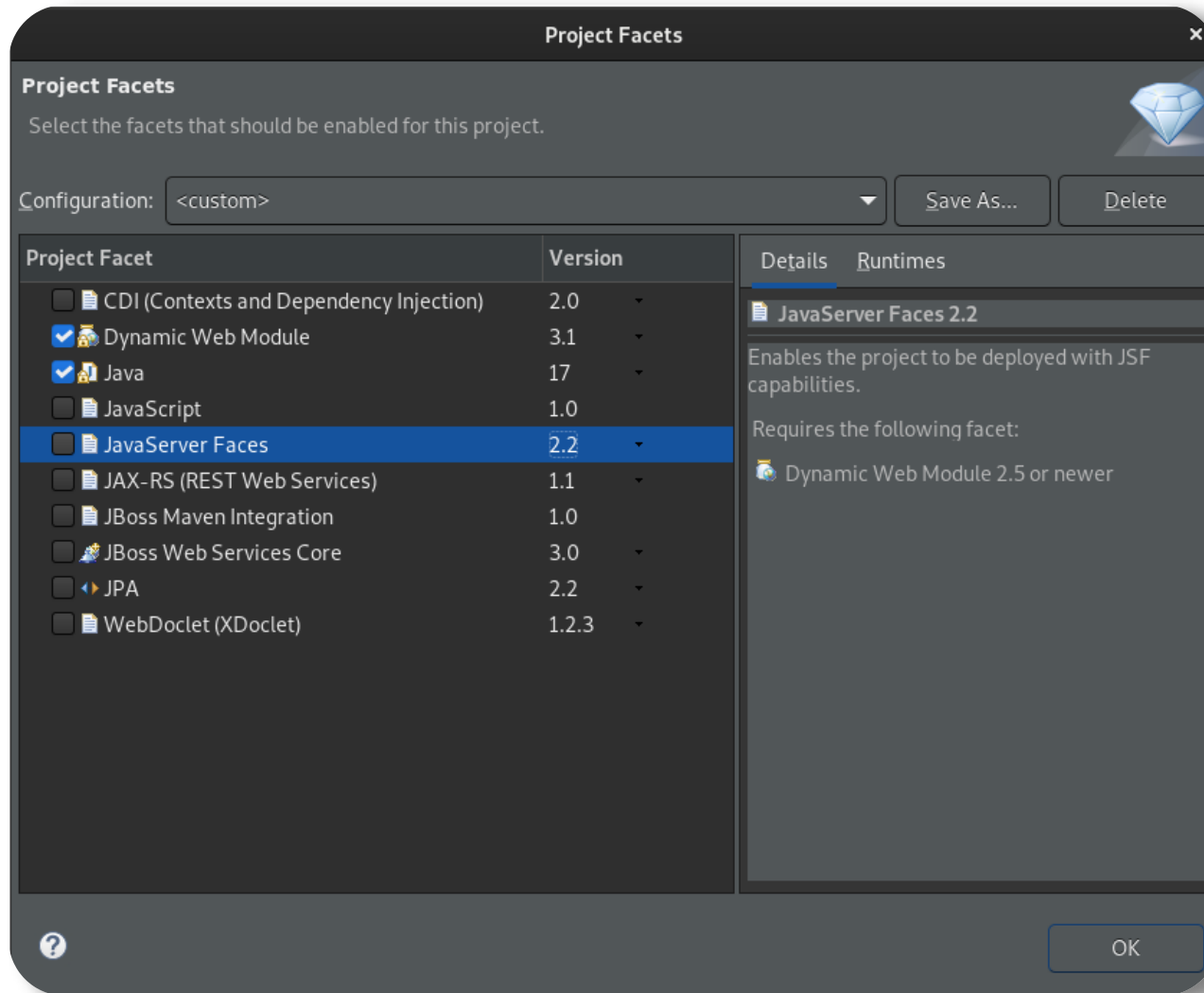
Configuration

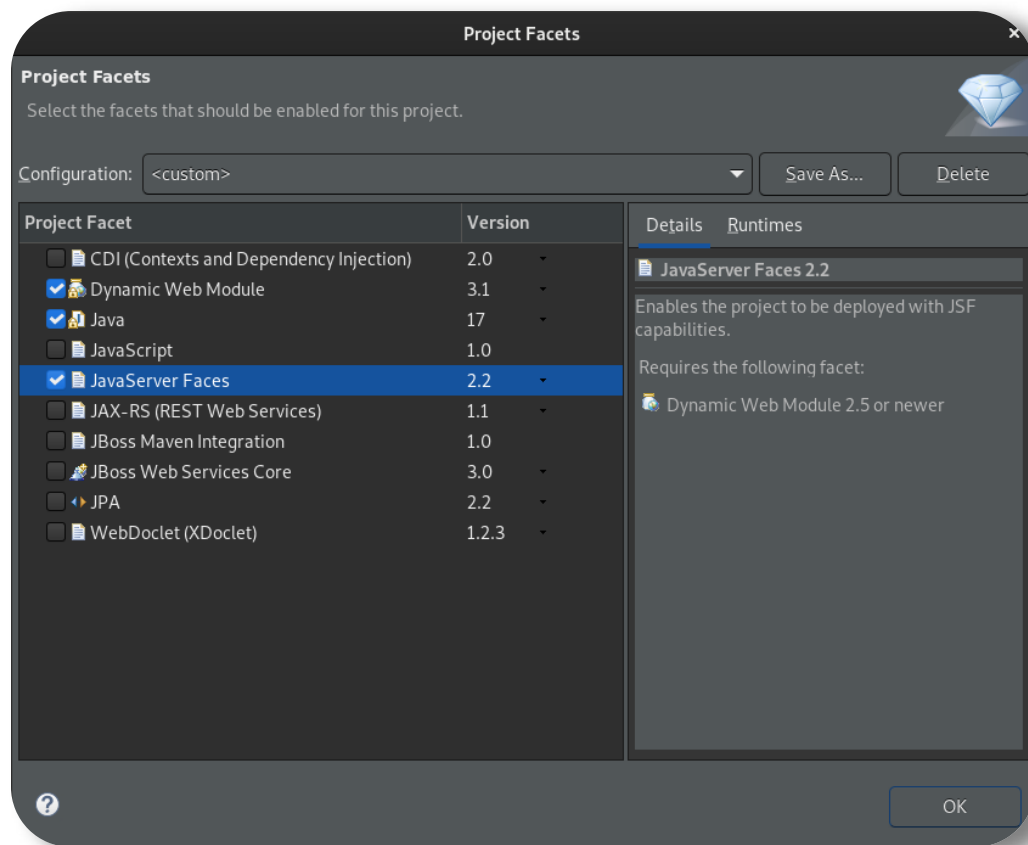
Hint: Get started quickly by selecting one of the pre-defined project configurations.

EAR membership
☐ Add project to an EAR
EAR project name:

Working sets
☐ Add project to working sets
Working sets:







New Dynamic Web Project

Web Module
Configure web module settings.

Context root: FirstJsf

Content directory: Webcontent

☒ Generate web.xml deployment descriptor

? < Back Next > Cancel Finish



New Dynamic Web Project

JSF Capabilities

⚠ Library configuration is disabled. Further classpath changes may be required later.

JSF Implementation Library

Type: Disable Library Configuration

This facet requires JSF implementation library to be present on project classpath. By disabling library configuration, user takes on responsibility of configuring classpath appropriately via alternate means.

☒ Configure JSF servlet in deployment descriptor

JSF Configuration File: /WEB-INF/faces-config.xml

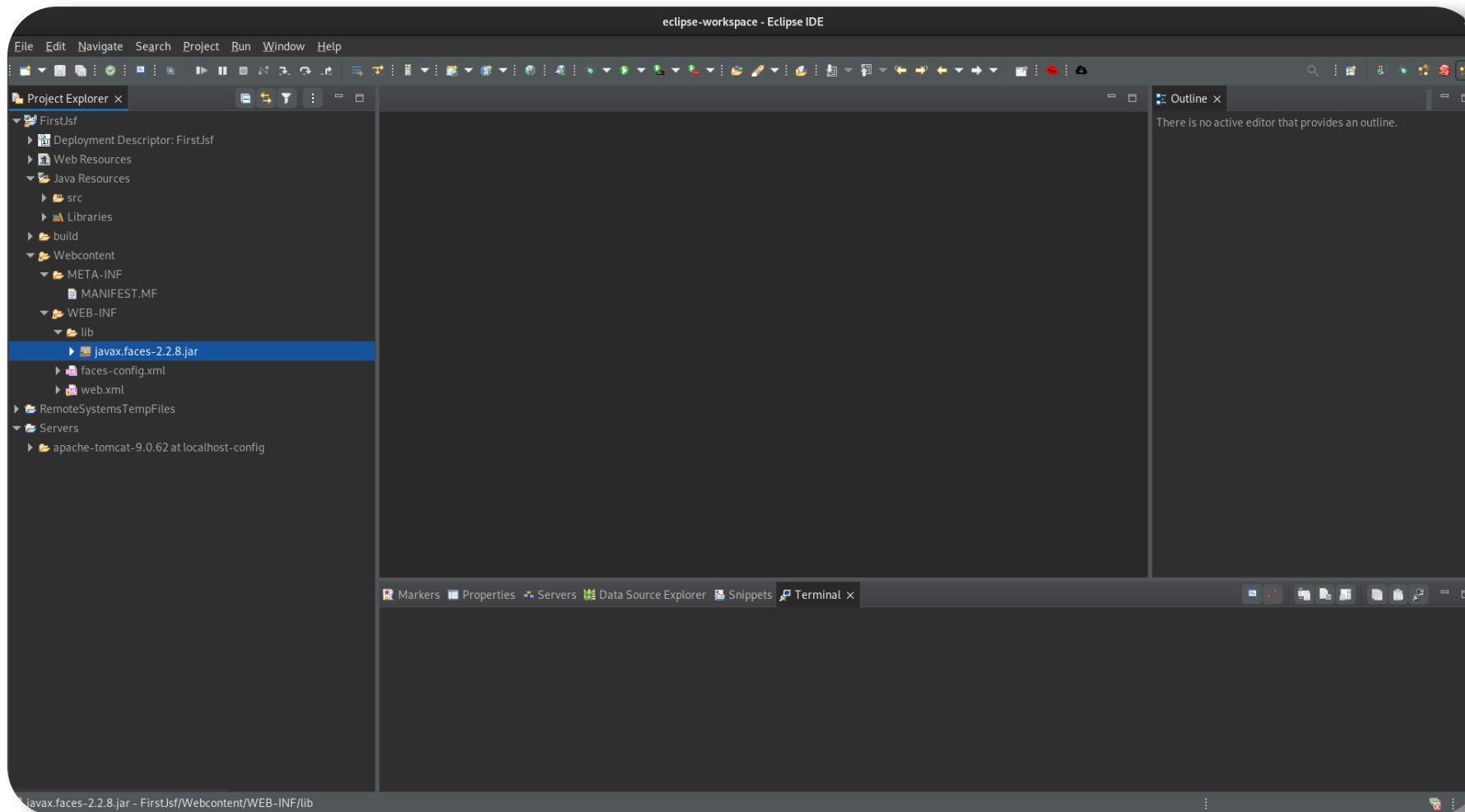
JSF Servlet Name: Faces Servlet

JSF Servlet Class Name: javax.faces.webapp.FacesServlet

URL Mapping Patterns: /faces/* Add... Remove

? < Back Next > Cancel Finish





LE CONTENU DE WEB-INF



WEB-INF est un dossier qui contient des fichiers qui ne doivent pas être accessibles par l'utilisateur.

Il contient :

- ☕ le fichier web.xml : fichier de configuration du serveur web
- ☕ le dossier lib : dossier qui contient les librairies
- ☕ le dossier classes : dossier qui contient les classes compilées

LE FICHER WEB.XML

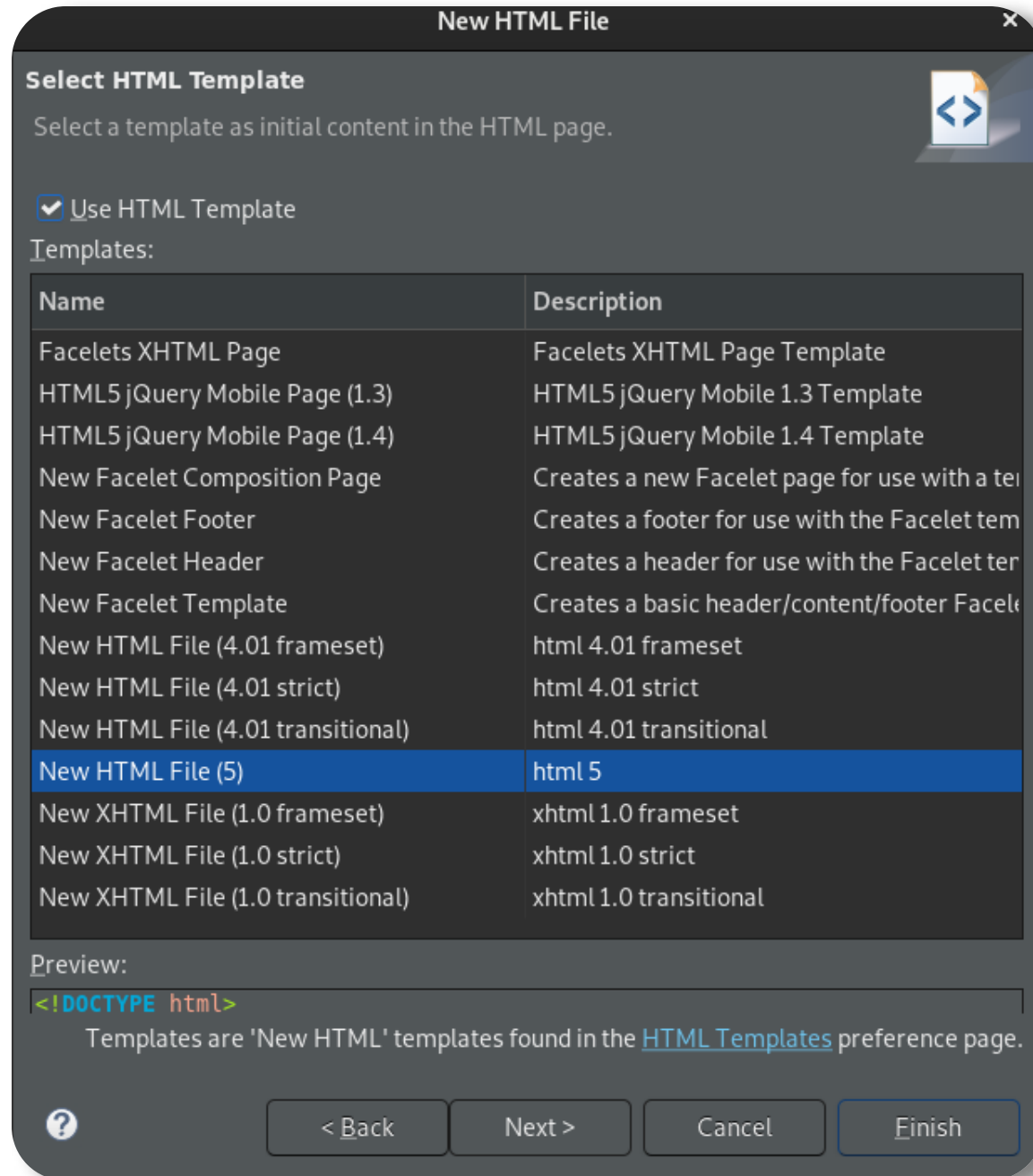
Le fichier web.xml est un fichier de configuration qui permet de définir les paramètres de l'application.

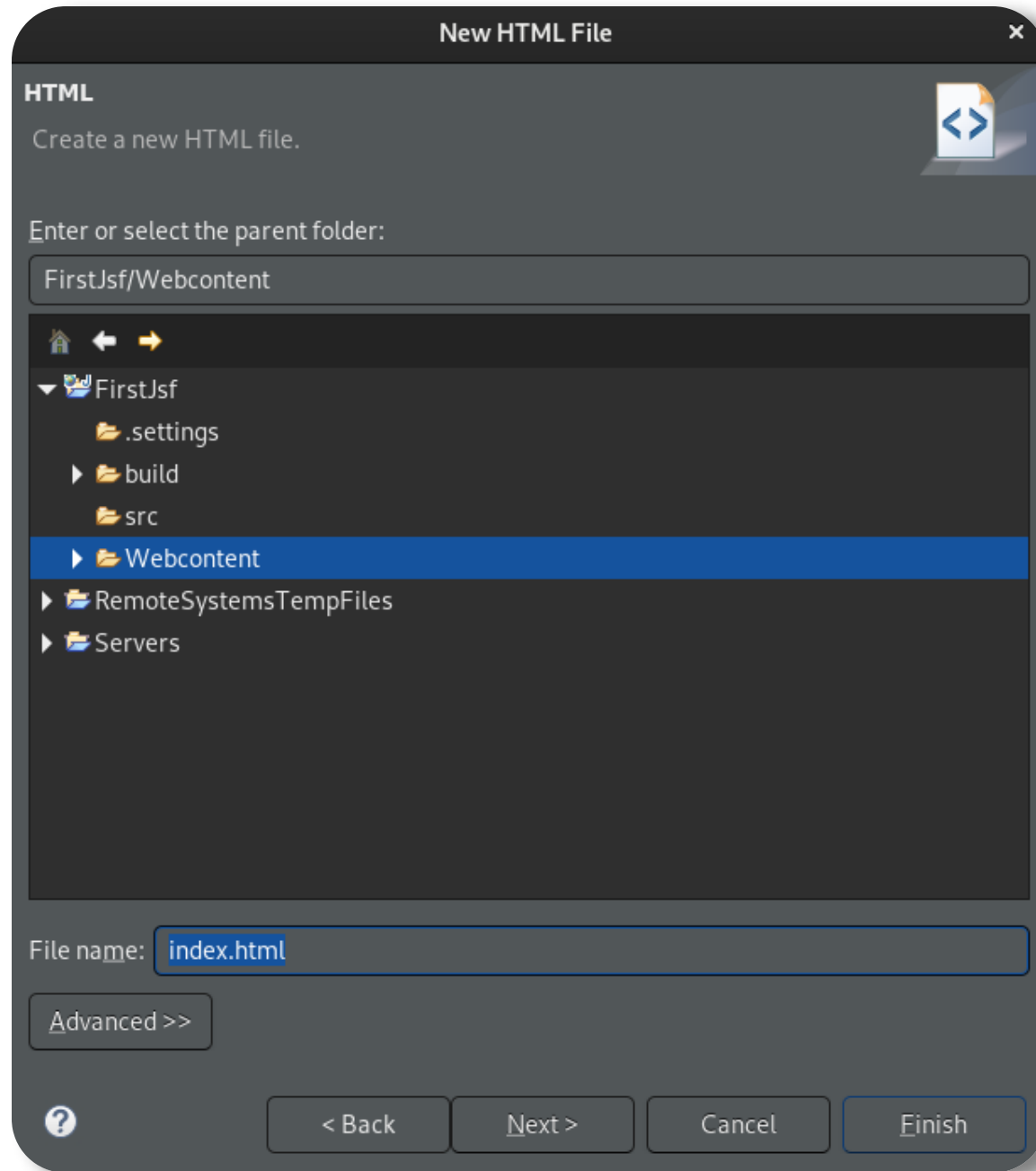
Il est possible de définir :

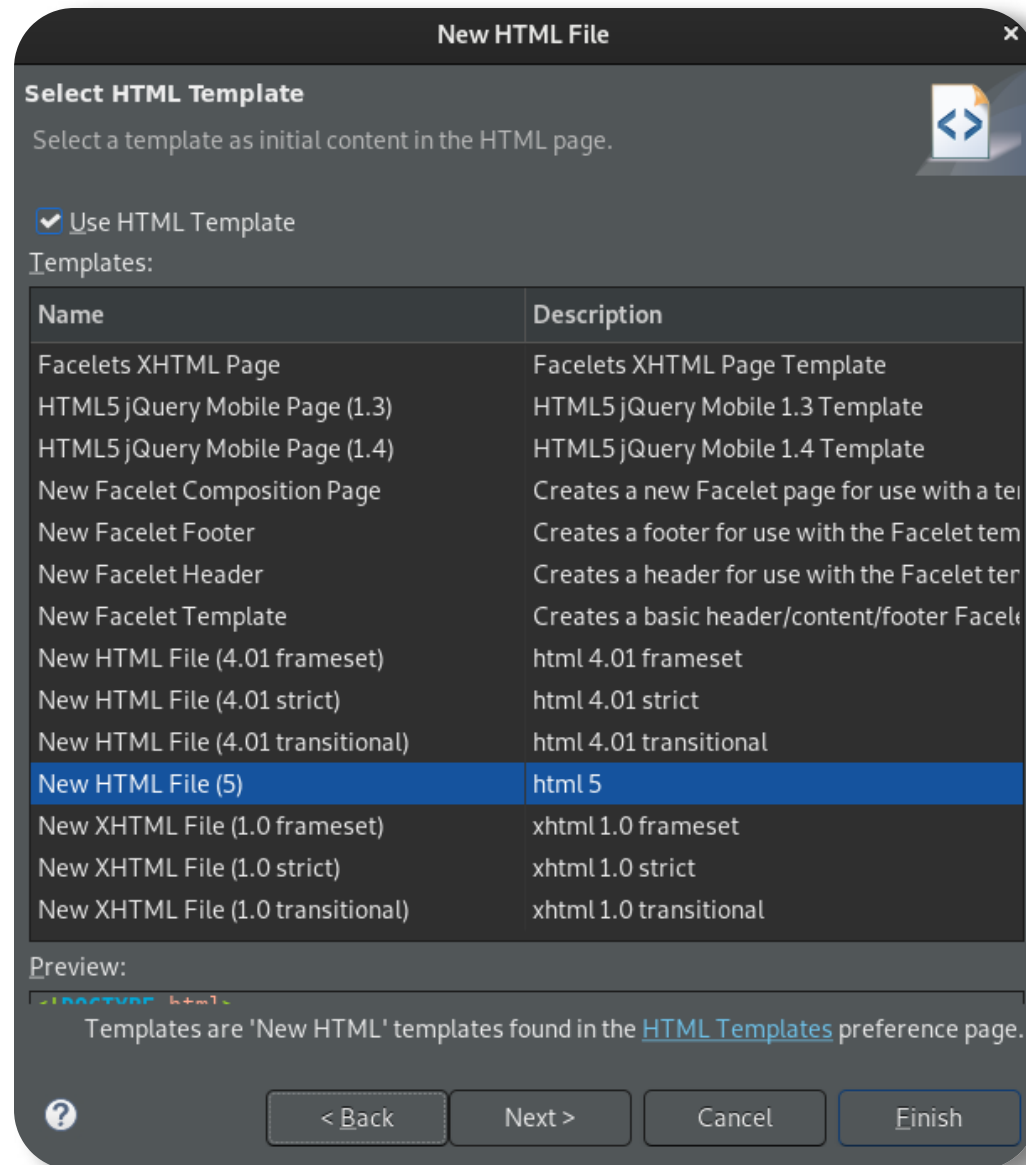
- ☕ le nom de l'application
- ☕ les servlets de l'application
- ☕ les servlet-mapping de l'application

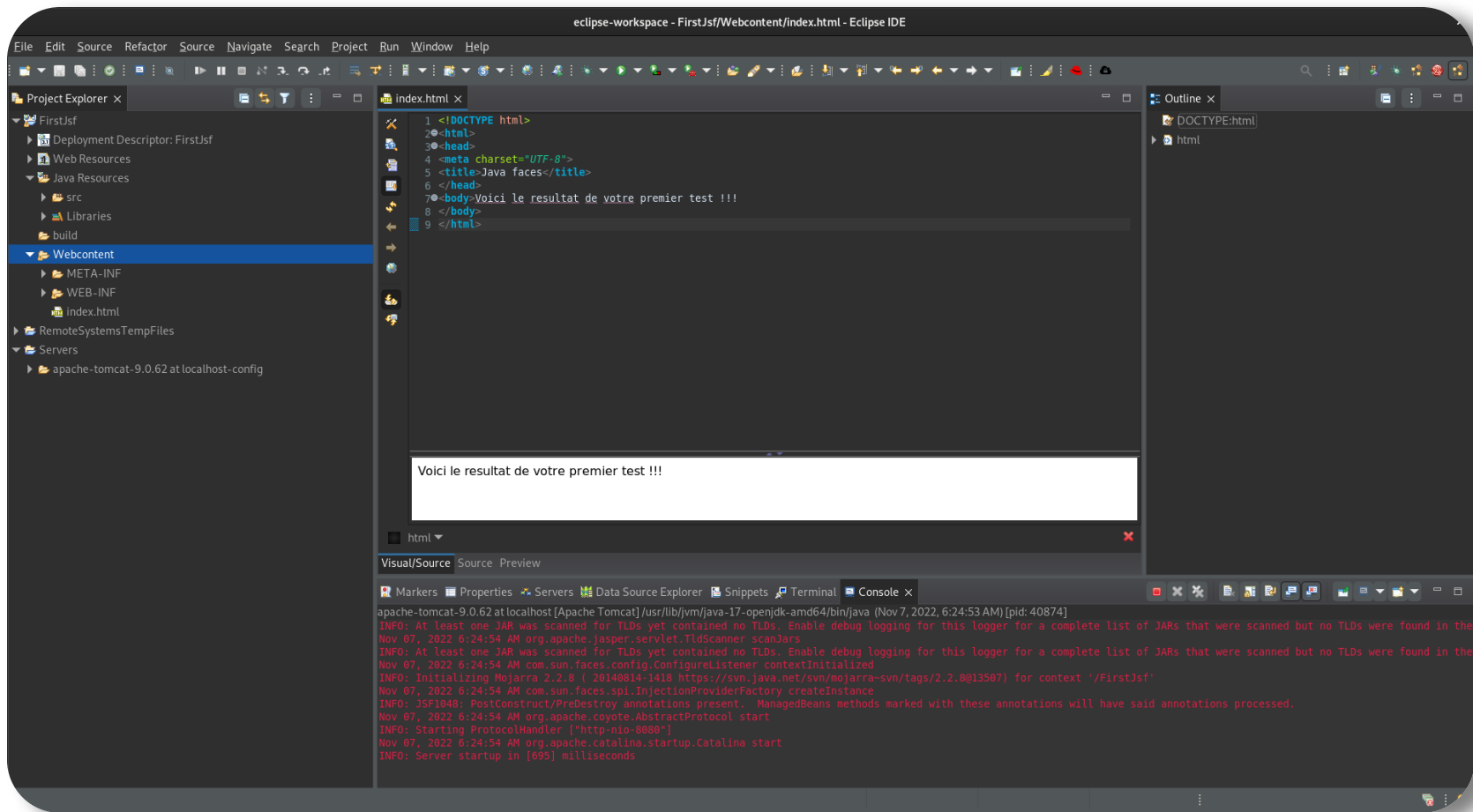
```
<servlet-mapping>  
<servlet-name>jsp</servlet-name>  
<url-pattern>*.jsf</url-pattern>  
</servlet-mapping>
```











Run On Server

Run On Server

Select which server to use

How do you want to select the server?

☒ Choose an existing server

☐ Manually define a new server

Select the server that you want to use:

type filter text

Server	State
▼ localhost	
📁 apache-tomcat-9.0.62 at localhost	🟢 Started

Apache Tomcat v9.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5, 6, 7, and 8 Web modules.

☒ Always use this server when running this project

?

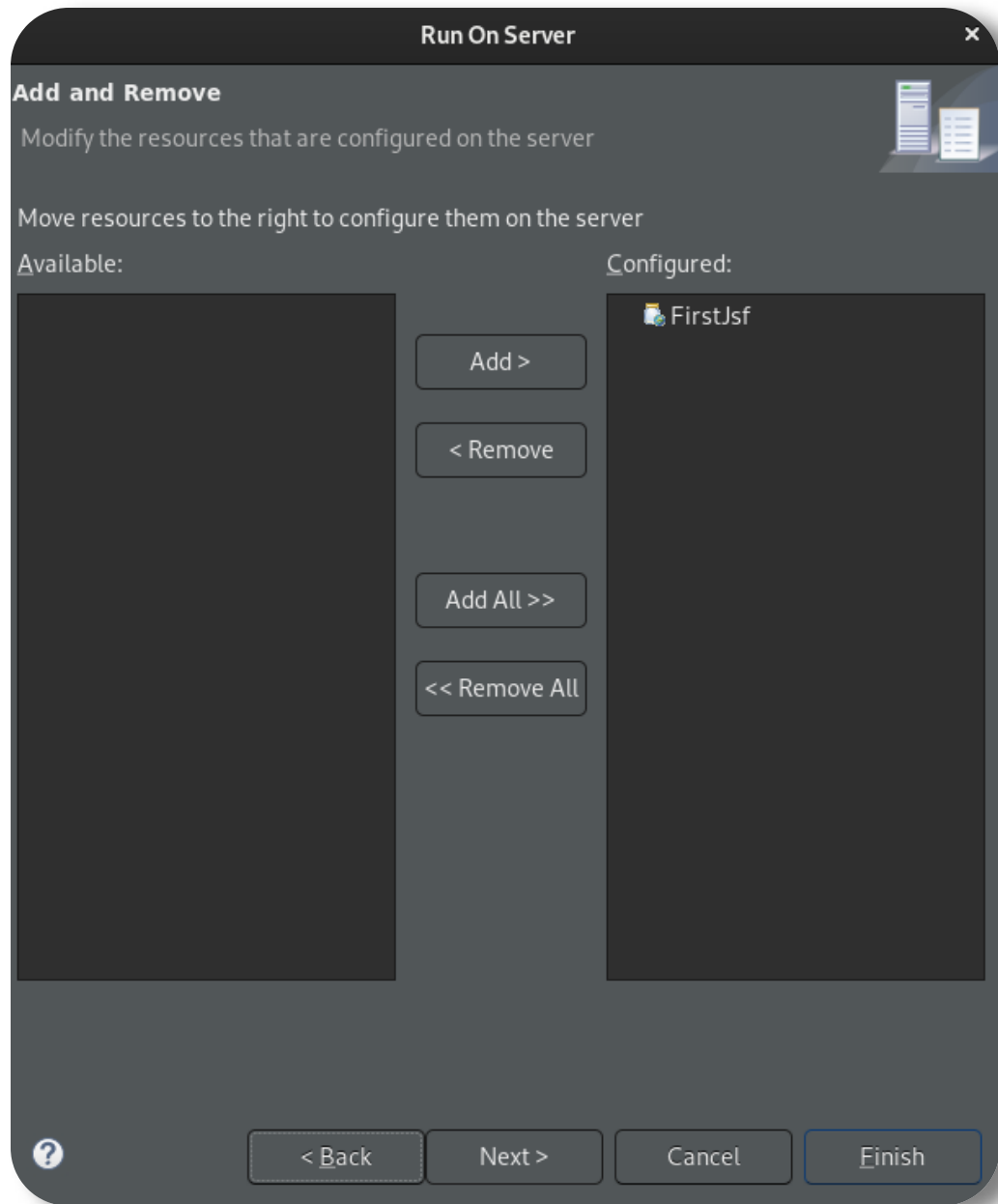
< Back

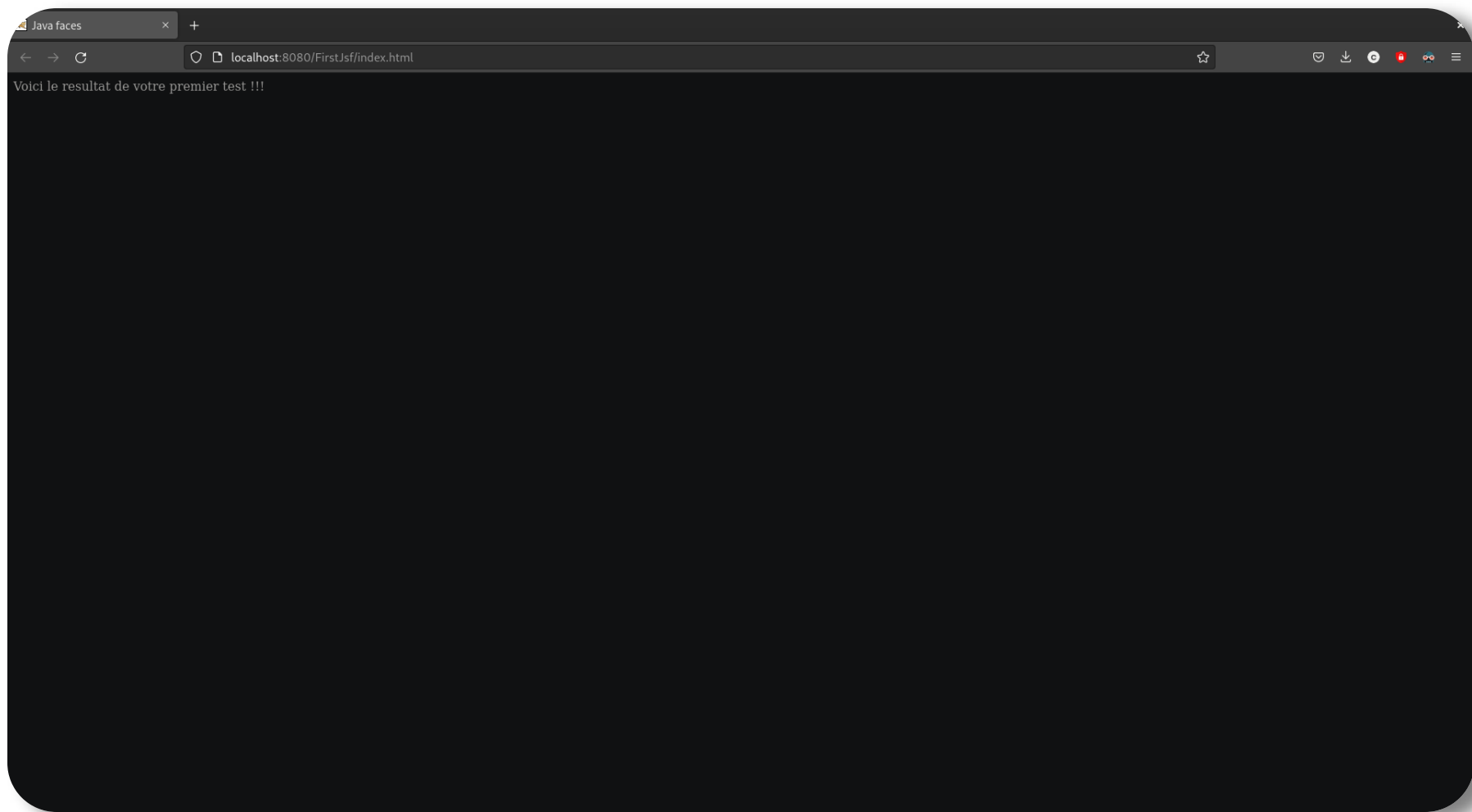
Next >

Cancel

Finish







NOTRE PREMIER HTML AVEC JSF

Pour créer notre premier html avec jsf, nous créons un fichier nommé index.html dans le dossier webapp.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Java faces</title>
  </head>
  <body>
    Voici le resultat de votre premier test !!!
  </body>
</html>
```



QU'EST CE QUE LE FICHER XHTML ?

Le fichier xhtml est un fichier html qui permet de créer des pages web dynamiques.

Il est composé de balises html et de balises jsf.

☕ Jsf : gère la partie dynamique

☕ Html : gère la partie statique

Le fichier xhtml est composé de 3 parties :

☕ le header

☕ le body

☕ le footer





```
<!DOCTYPE html>
<html
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://xmlns.jcp.org/jsf/html"
  xmlns:f="http://xmlns.jcp.org/jsf/core"
>
  <h:head>
    <title>Java faces</title>
  </h:head>
  <h:body>
    <!-- votre code s'effectuera ici -->
  </h:body>
</html>
```

LES BALISES JSF



Les balises jsf permettent de créer des pages web dynamiques.

Elles sont composées de 3 parties :

```
<h:balise attribut="valeur">contenu</h:balise>
```

LA BALISE INPUTTEXT



La balise `inputText` permet de créer un champ de saisie.

```
<h:inputText />
```


LES BALISES OUTPUTTEXT



La balise `outputText` permet d'afficher du texte.

```
<h:outputText value="Le texte affiché" />
```

LES EXPRESSIONS DE LIAISON

Les expressions de liaison permettent de lier une variable à une balise.

```
<h:outputText value="#{variable}" />
```

Cette expression fonctionnera comme une variable.

Elle possède plusieurs syntaxes, mais pour l'instant nous allons nous concentrer sur celle-ci. :

```
#{variable}
```



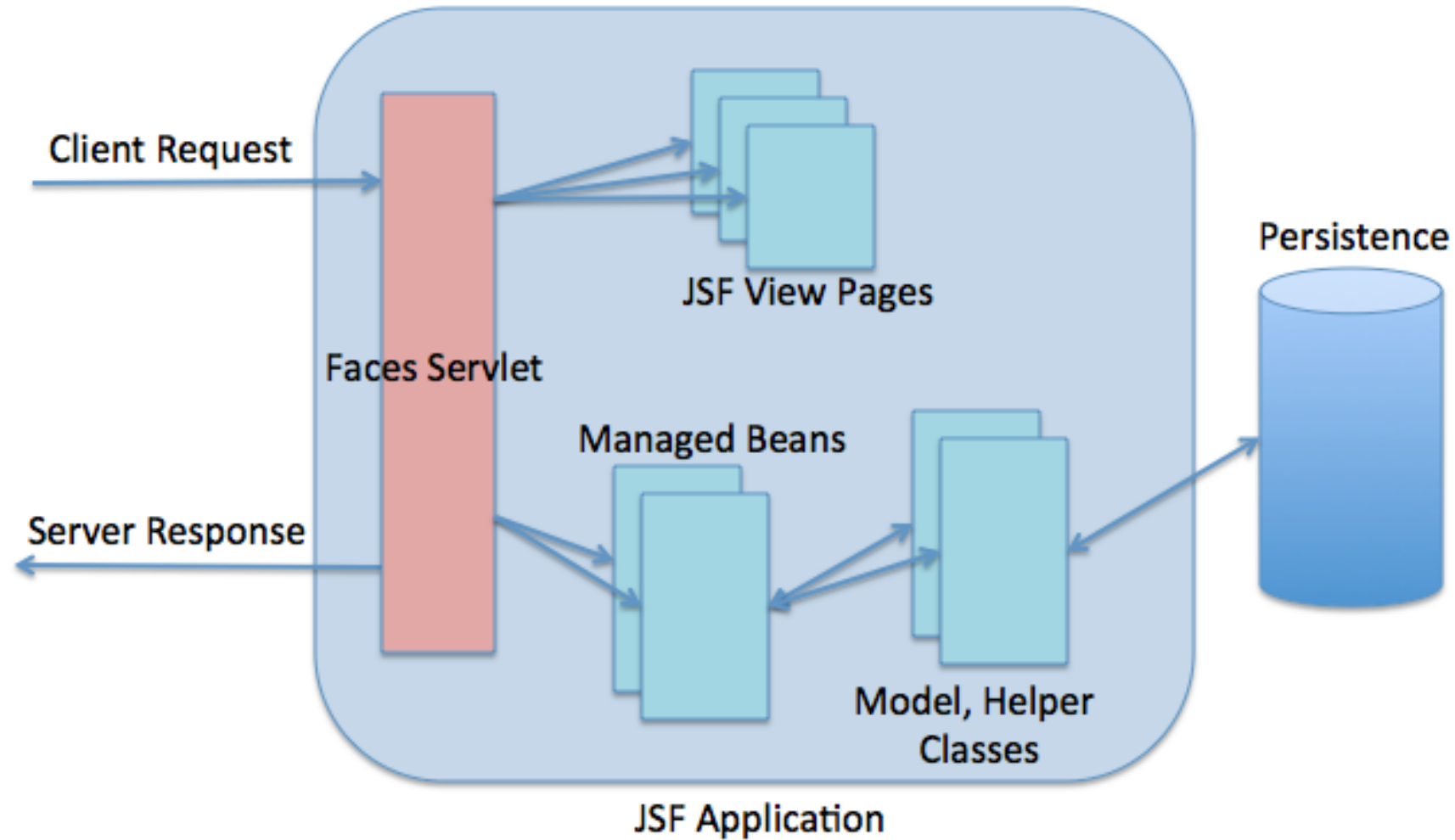
LES BALISES COMMANDBUTTON



La balise `commandButton` permet de créer un bouton.

```
<h:commandButton value="Post/Get/Submit" action="page2" />
```

REPRESENTATION DE JSF



TP : MA PREMIÈRE PAGE WEB AVEC JSF



Vous devez créer deux pages web :

- ☕ la première page vous demandera votre nom.
- ☕ la deuxième page vous affichera : "Bonjour " votre nom.

SUITE DE TP



```
<h:form>
  <h:outputText value="Votre nom : " />
  <h:inputText value="#{firstName}" />
  <h:outputText value="Votre prenom : " />
  <h:inputText value="#{lastName}" />
  <h:commandButton value="Valider" action="page2" />
</h:form>
```

SUITE DE TP



```
<h:form>  
    <h:outputText value="Bonjour #{firstName} #{lastName}" />  
</h:form>
```


LES BALISES INPUT HTML 5

L'HtML 5 a introduit de nouvelles balises pour créer des formulaires.

 input type

 text

 password

 email

 number

 date

 time

 color

 checkbox



LES BALISES INPUT HTML 5



```
<input type="text" value="#{variable}" />
<input type="password" value="#{variable}" />
<input type="email" value="#{variable}" />
<input type="number" value="#{variable}" />
<input type="date" value="#{variable}" />
d<input type="time" value="#{variable}" />
<input type="color" value="#{variable}" />
<input type="checkbox" value="#{variable}" />
```

LES BEANS



Les beans sont :

☕ des pojos

Ils doivent avoir :

☕ de getters et de setters qui respectent une convention de nommage particulière pour les attributs

☕ un constructeur par défaut sans arguments

☕ être serializable

☕ être annoté avec `@ManagedBean`

LES MANAGED BEANS

Les managed beans sont des beans gérés par le conteneur d'application(jsf).

Ils sont créés et détruits par le conteneur d'application.

Dans le fichier web.xml, nous devons déclarer le bean.

```
<managed-bean>  
<managed-bean-name>bean</managed-bean-name> // nom du bean  
<managed-bean-class>com.bean.Bean</managed-bean-class> // le chemin de la classe  
<managed-bean-scope>session</managed-bean-scope> // session, request, application  
</managed-bean>
```



TP



Créer trois beans :

- ☕ un bean qui contient un attribut nom
- ☕ un bean qui contient un attribut prenom
- ☕ un bean qui contient un attribut age

Créer une page web qui saisira les informations de l'utilisateur et qui les affichera.

LES SCOPES



Les scopes permettent de définir la portée d'un bean.

- ☕ session : le bean est créé à la création de la session et détruit à la fin de la session.
- ☕ request : le bean est créé à la création de la requête et détruit à la fin de la requête.
- ☕ application : le bean est créé à la création de l'application et détruit à la fin de l'application.

LES CYCLES DE VIE DES BEANS

Le cycle de vie d'un bean est composé de 6 étapes :




- ☕ restore view / reconstruct component tree : cette étape permet de restaurer la vue et de reconstruire l'arbre des composants.
- ☕ apply request values : cette étape permet de récupérer les valeurs des composants.
- ☕ process validations : cette étape permet de valider les composants.
- ☕ update model values : cette étape permet de mettre à jour les valeurs des composants.
- ☕ invoke application : cette étape permet d'appeler l'application.
- ☕ render response : cette étape permet de rendre la réponse.



LES EXPRESSIONS DE LIAISON DE BEANS



Les expressions de liaison peuvent être utilisées dans les balises jsf pour :

-  afficher des valeurs
-  récupérer des valeurs
-  appeler des méthodes

LES COMPOSANTS DE BASE

Les composants de base sont des composants qui permettent de créer des formulaires.



- ☕ **inputText** : permet de créer un champ de texte.
- ☕ **inputSecret** : permet de créer un champ de mot de passe.
- ☕ **inputHidden** : permet de créer un champ caché.
- ☕ **inputTextarea** : permet de créer un champ de texte multiligne.
- ☕ **inputFile** : permet de créer un champ de fichier.
- ☕ **inputCheckbox** : permet de créer une case à cocher.
- ☕ **inputRadio** : permet de créer un bouton radio.

LES CHECKBOX



Les checkbox permettent de créer des cases à cocher.

```
<h:selectBooleanCheckbox value="#{bean.bac}" />
```

```
private boolean bac;
```

LES BOUTONS RADIO



Les boutons radio permettent de créer des boutons radio.

```
<h:selectOneRadio value="#{bean.sexe}">
    <f:selectItem itemLabel="homme" itemValue="homme" />
    <f:selectItem itemLabel="femme" itemValue="femme" />
</h:selectOneRadio>
```

```
private String sexe;
```

LES CONDITIONS EN JSF



Les conditions permettent d'afficher ou non des composants.

☕ **h:outputText** : permet d'afficher du texte.

☕ **value** : permet de définir le texte à afficher.

☕ **rendered** : permet de définir si le composant doit être affiché ou non.

exemple :

```
<h:outputText value="Bonjour #{bean.nom}" rendered="#{bean.bac}" />
```

LES FORMULAIRES

Jsf permet la création de formulaires, nous pouvons valider les données saisies par l'utilisateur.

Le bouton submit permet de valider les données saisies par l'utilisateur.

Le bouton reset permet de réinitialiser les données saisies par l'utilisateur.

```
<h:form>
  <h:inputText value="#{bean.attribut}" />
  <h:commandButton value="valider" action="#{bean.methode()}" />
</h:form>
```



TP : FORMULAIRE

Vous devez créer un formulaire qui permet de saisir les informations suivantes :

☕ nom

☕ prenom

☕ sexe

Dans la deuxième page vous afficherez : "Bonjour ", "monsieur" ou "madame", le nom et le prenom de l'utilisateur.



LES FORMULAIRES



```
<h:form>
  <h:selectOneMenu value="#{bean.attribut}">
    <f:selectItem itemValue="valeur1" itemLabel="valeur1" />
    <f:selectItem itemValue="valeur2" itemLabel="valeur2" />
  </h:selectOneMenu>
  <h:selectBooleanCheckbox value="#{bean.attribut}" />
  <h:commandButton value="valider" action="#{bean.methode()}" />
</h:form>
```

TP : FORMULAIRE

Vous devez créer un site web qui permet de saisir les informations suivantes :

☕ nom

☕ age

☕ sexe

☕ poids

☕ taille

Vous allez devoir ajouter un bouton qui permet de calculer l'IMC.

la formule de l'IMC est : $\text{poids} / \text{taille}^2$.

exemple : <https://www.santepratique.fr/nutrition/calcul-imc>



LES CONVERTERS

Les converters permettent de convertir les valeurs des composants.

Les converters sont des classes qui implémentent l'interface Converter.

Elles doivent être annotées avec @FacesConverter.

```
@FacesConverter("converter")
public class Converter implements javax.faces.convert.Converter {
    @Override
    public Object getAsObject(FacesContext context, UIComponent component, String value) {
        return value;
    }

    @Override
    public String getAsString(FacesContext context, UIComponent component, Object value) {
        return value.toString();
    }
}
```



LES CONVERTERS



```
<h:inputText value="#{bean.attribut}">  
    <f:convertDateTime pattern="dd/MM/yyyy" />  
</h:inputText>
```

LES CONVERTERS DE BASE



- ☕ **convertDateTime** : permet de convertir une date.
- ☕ **convertNumber** : permet de convertir un nombre.
- ☕ **convertFloat** : permet de convertir un nombre à virgule flottante.
- ☕ **convertDouble** : permet de convertir un nombre à virgule flottante.
- ☕ **convertLong** : permet de convertir un nombre entier.
- ☕ **convertInteger** : permet de convertir un nombre entier.
- ☕ **convertShort** : permet de convertir un nombre entier.
- ☕ **convertCharacter** : permet de convertir un caractère.
- ☕ **convertBoolean** : permet de convertir un booléen.

DES EXEMPLES D'UTILISATION



```
<h:inputText value="#{bean.attribut}">
    <f:convertDateTime pattern="dd/MM/yyyy" />
</h:inputText>

<h:inputText value="#{bean.attribut}">
    <f:convertDouble />
</h:inputText>

<h:inputText value="#{bean.attribut}">
    <f:convertInteger />
</h:inputText>
```

LEURS UTILISATIONS



```
<h:inputText value="#{bean.attribut}">
    <f:convertDateTime pattern="dd/MM/yyyy" />
</h:inputText>
<h:inputText value="#{bean.attribut}">
    <f:convertNumber type="currency" currencySymbol="€" />
</h:inputText>
<h:inputText value="#{bean.attribut}">
    <f:convertNumber type="percent" />
</h:inputText>
```

DROP DOWN LIST



La drop down list permet de créer une liste déroulante.

```
<h:selectOneMenu value="#{bean.nomAttribut}">
  <f:selectItem itemLabel="choix1" itemValue="valeur1" />
  <f:selectItem itemLabel="choix2" itemValue="valeur2" />
  <f:selectItem itemLabel="choix3" itemValue="valeur3" />
</h:selectOneMenu>
```

```
private String nomAttribut;
```

TP : DROP DOWN LIST

Dans votre page web, vous devez créer une liste déroulante qui contient les choix suivants :

- ☕ france
- ☕ belgique
- ☕ allemagne
- ☕ italie

Cette liste déroulante doit être liée à une variable, pays de la classe user.



DROP DOWN LIST 2



Plutôt que de coder en dur chaque élément de la liste déroulante, nous pouvons faire des listes uniques :



```
class bean{
    private String attribut;
    private List<String> listeAttribut;

    bean(){
        listeAttribut = new ArrayList<String>();
        listeAttribut.add("choix1");
        listeAttribut.add("choix2");
        listeAttribut.add("choix3");
    }

    public String getAttribut(){
        return attribut;
    }

    public void setAttribut(String attribut){
        this.attribut = attribut;
    }

    public List<String> getListeAttribut(){
        return listeAttribut;
    }
}
```




```
<h:selectOneMenu value="#{bean.nomAttribut}">  
    <f:selectItems value="#{bean.getAttribut}" />  
</h:selectOneMenu>
```

LES VALIDATIONS

Les validations permettent de valider les composants.

JSF propose plusieurs validations :

- ☕ required : permet de vérifier si un composant est rempli.
- ☕ length : permet de vérifier la longueur d'un composant.
- ☕ regex : permet de vérifier si un composant correspond à une expression régulière.
- ☕ range : permet de vérifier si un composant est compris entre deux valeurs.
- ☕ compare : permet de comparer deux composants.



LES VALIDATIONS



Les validators fonctionnent avec le tag h:message.

```
<h:inputText  
    value="#{bean.nomAttribut}"  
    require="true"  
    requiredMessage="La saisie est obligatoire !"  
>  
<h:message for="nomAttribut" styleClass="errorBlock" />
```

LES VALIDATIONS



```
<h:inputText value="#{bean.nomAttribut}">  
  <f:validateLength minimum="5" maximum="10" />  
</h:inputText>
```

LES DIFFÉRENTES VALIDATIONS



```
<h:inputText value="#{bean.nomAttribut}">
  <f:validateLength minimum="5" maximum="10" /> // longueur entre 5 et 10
  <f:validateRegex pattern="^[a-zA-Z]+$" /> // expression reguliere
  <f:validateRange minimum="18" maximum="100" /> // entre 18 et 100
  <f:validateRequired /> // obligatoire
  <f:validateDoubleRange minimum="0" maximum="100" /> // entre 0 et 100
  <f:validateLongRange minimum="0" maximum="100" /> // entre 0 et 100
  <f:validateCompare value="valeur" /> // compare avec une valeur
  <f:validateCompare value="#{bean.nomAttribut}" /> // compare avec une autre
  valeur
</h:inputText>
```

UTILISATION DES VALIDATIONS

```
<h:inputText value="#{bean.nomAttribut}">
    <f:validateLength minimum="5" maximum="10" />
</h:inputText>
```

```
public class bean{
    private String nomAttribut;

    public String getNomAttribut(){
        return nomAttribut;
    }

    public void setNomAttribut(String nomAttribut){
        this.nomAttribut = nomAttribut;
    }
}
```



TP : VALIDATIONS

Dans un nouveau projet web JSF, vous devez créer une page web qui affichera deux boutons :

- ☕ un bouton pour créer un nouvel utilisateur
- ☕ un bouton pour vous connecter en tant qu'utilisateur

lorsque vous cliquez sur le bouton créer un nouvel utilisateur, vous devez afficher un formulaire qui contient les champs suivants :

- ☕ username
- ☕ password
- ☕ email
- ☕ age





Si l'utilisateur n'a pas rempli tous les champs, il doit afficher un message d'erreur.

Si l'utilisateur a rempli tous les champs, il devra arriver sur une page qui affiche les informations de l'utilisateur.

Si l'utilisateur arrive sur la page de connexion, il doit pouvoir se connecter avec son username et son password.

Une fois connecté, il doit arriver sur une page qui affiche les informations de votre choix.

FACES 2.3



Pour les personnes qui veulent aller plus loin, voici comment créer des composants avec JSF 2.3.

CRÉATION D'UN PROJET 2.3



Pour créer un projet JSF 2.3, il faut créer un projet web avec les dépendances suivantes :

 JSF 2.3

 Primefaces 6.2

 CDI 1.2

GLASSFISH

Nous allons utiliser Glassfish pour créer notre projet.
Il faut donc télécharger Glassfish 5.0 et l'installer.
nous le téléchargerons ici :

<https://javaee.github.io/glassfish/download>

Pour l'intégrer dans eclipse, il faut aller dans Window >
Preferences > Server > Runtime Environments > Add > Glassfish
5.0.



CRÉATION D'UN PROJET 2.3

Pour créer un projet JSF 2.3, il faut créer un projet web dynamique web.

Il ne faut pas oublier :

- ☕ de cocher la case "Generate web.xml deployment descriptor" pour générer le fichier web.xml.
- ☕ d'ajouter les dépendances JSF 2.3, Primefaces 6.2 et CDI 1.2.
- ☕ de générer le fichier faces-config.xml && un fichier de lancement de CDI.



APPLICATIONCONFIGURATION



```
@ApplicationScoped  
@FacesConfig(version = FacesConfig.Version.JSF_2_3)  
public class ApplicationConfiguration {  
}
```

DIFFERENCES ENTRE 2.2 ET 2.3

Faces 2.3 permet de créer des composants avec des annotations.

Il faut créer une classe qui hérite de `UIComponentBase`.

Ces composants face 2.3 sont des composants de type CDI.

Ils peuvent donc être injectés dans les autres composants.

```
@FacesComponent("composant")
public class Composant extends UIComponentBase {
    @Override
    public String getFamily() {
        return "composant";
    }
}
```

```
<composant />
```



CONCLUSION

JSF est un framework qui permet de créer des applications web dynamiques.

Il permet de créer des pages web avec des composants.

Nous avons vu comment créer des composants, les lier à des variables, les afficher, les modifier, les valider.

