

Création d'une Application Web / JSP

Rédacteurs : Alexandre Baillif, Philippe Lacomme et Raksmei Phan

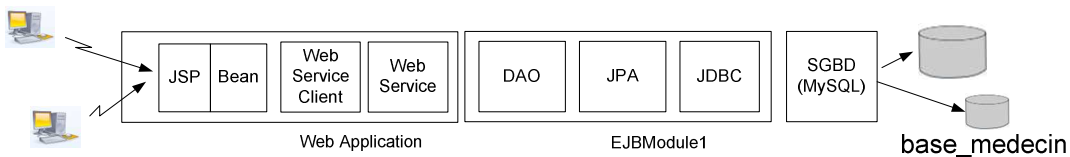
Date : juillet 2010

Avertissement : ce document est une reprise d'une partie d'un document écrit par Serge Tahé. En particulier la base de données utilisée.

Nous proposons deux solutions utilisant l'**EJBModule1** et l'**EJBModule2** et une solution standalone.

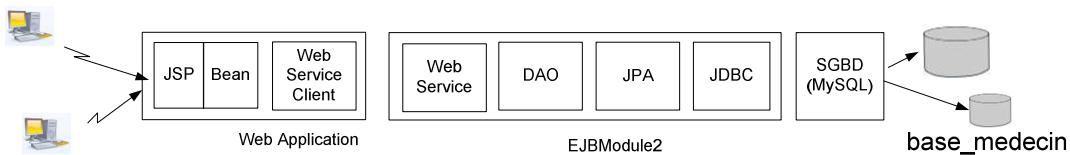
NB : Il faut savoir qu'il existe de nombreuses variantes et donc de nombreuses architectures possibles, mais celles présentées ici conviennent dans la grande majorité des cas.

Solution 1.



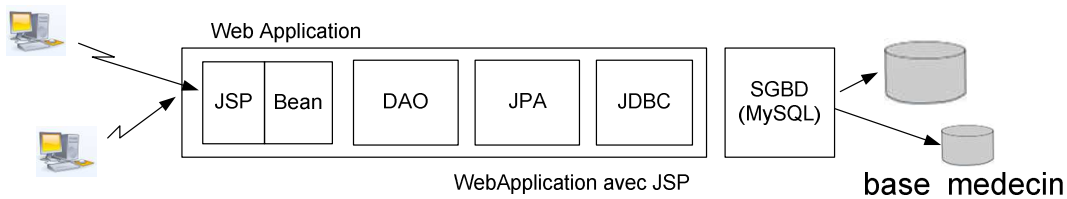
Cette solution permet de créer une application web à partir d'un EJB existant. Cette solution permet de réutiliser un EJB présent sur un serveur à travers un web service. Ici le web service et le client sont dans la même application, mais on peut très bien imaginer un client dans une application différente.

Solution 2.



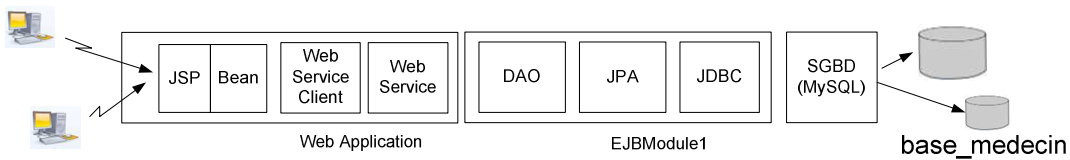
Cette solution permet de créer une application web à partir d'un web service existant. Cette solution est la plus portable car elle permet de faire tourner une application web de manière distante et indépendante de l'application métier (EJB). Cette solution est à préférer dans le cas de grosses applications (une variante sans le concept d'EJB peut très bien exister).

Solution 3.

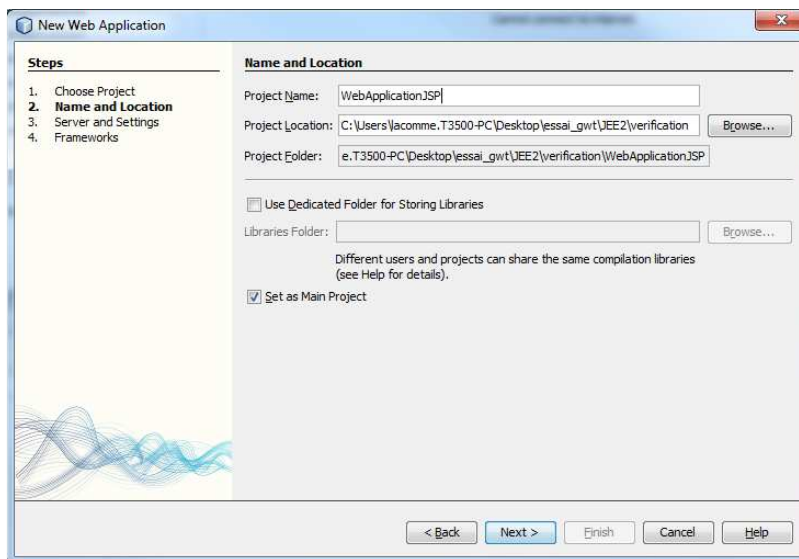
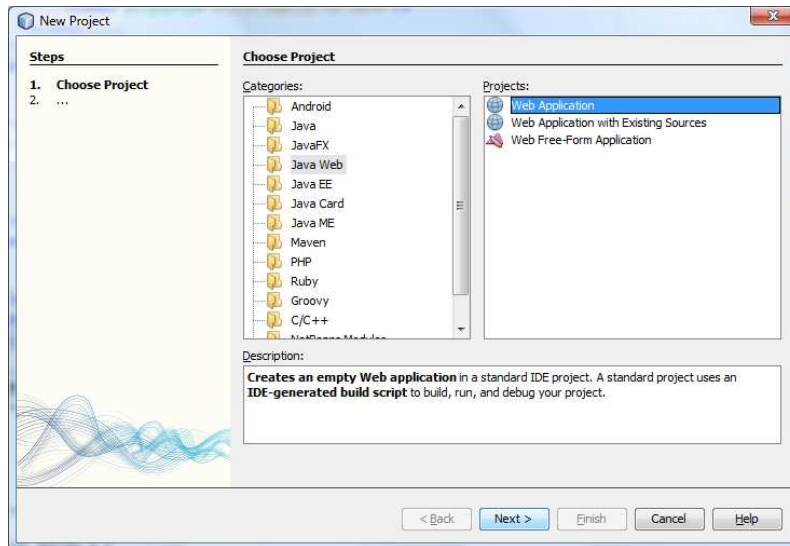


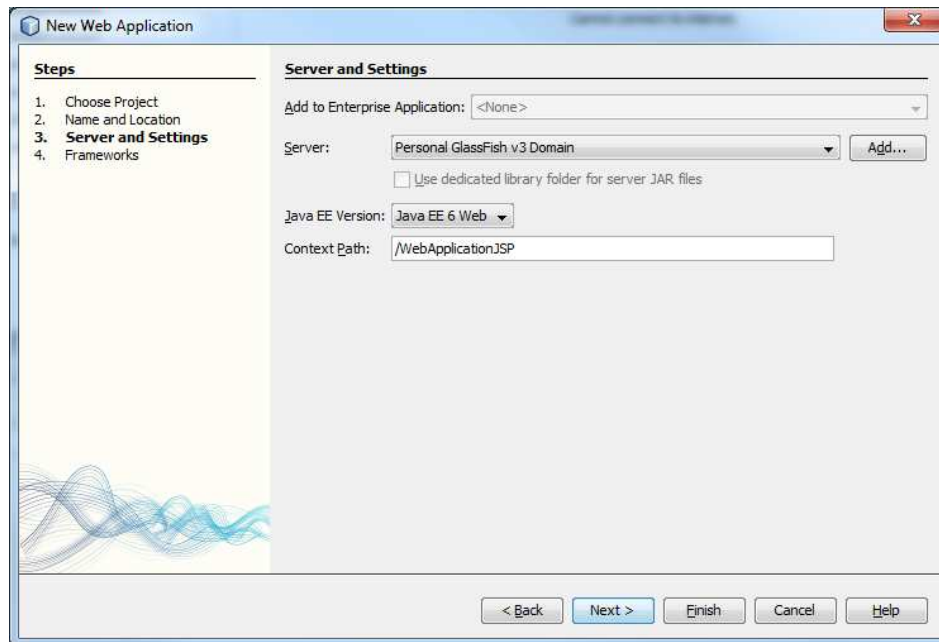
Cette solution permet de créer une application web sans avoir à utiliser d'EJB ou de web service. Elle possède une architecture plus simple (une seule application) et convient pour de petits projets qui tournent sur un seul serveur.

Solution 1.

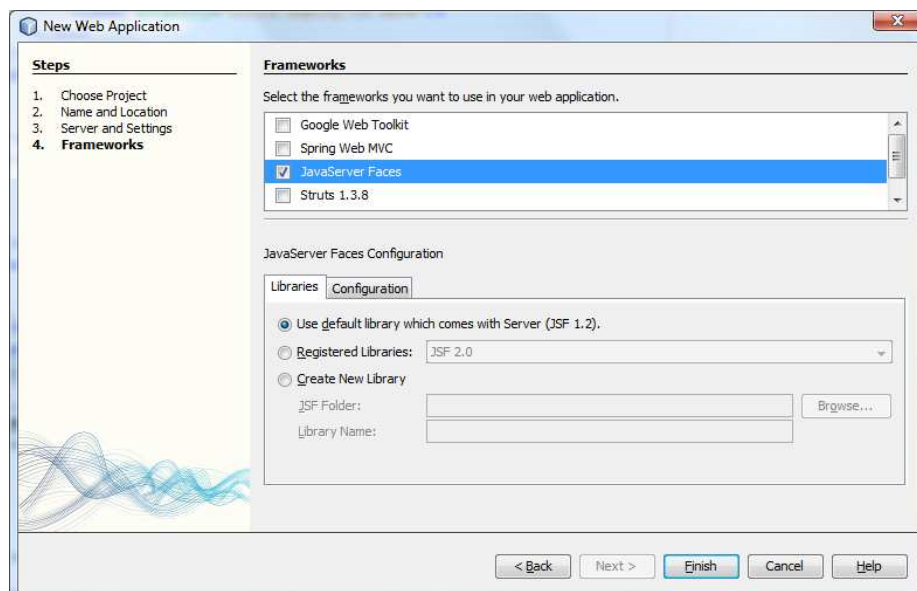


1) Conception de l'application client

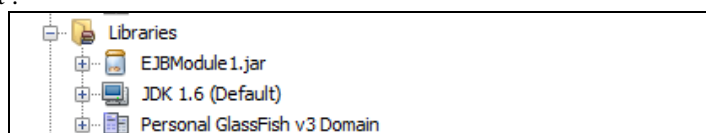




On peut choisir comme framework Java Server Faces par exemple.
Pour le serveur Glassfish, utiliser celui dont le port est disponible.

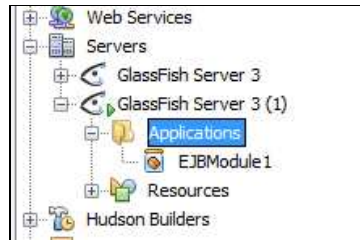


Faire un clic droit sur la partie « librairies » de l'application Web et choisir « Add Jar »
Choisir ensuite EJBModule1.jar. La partie Libraries de WebApplicationJSP devrait se présenter alors comme suit :

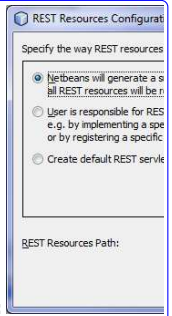


2) Définition d'un client web

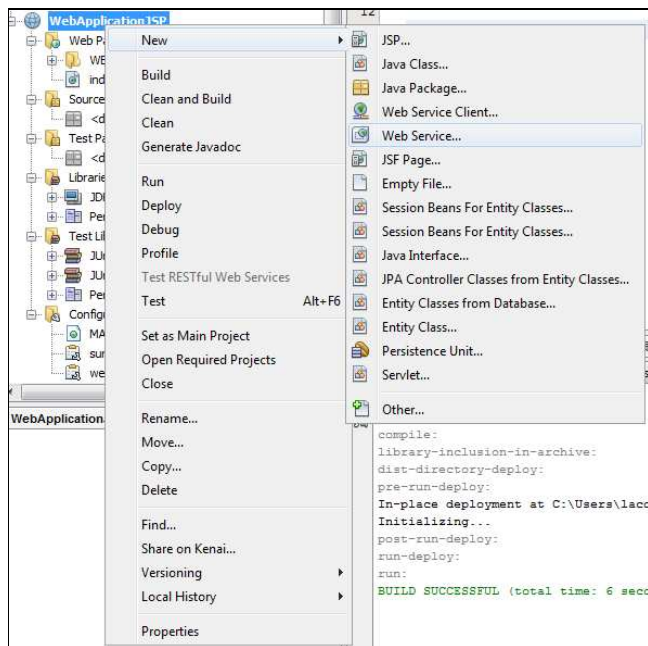
2.1. Vérifier dans la partie Services que l'EJBModule1 a bien été déployé. Vous devez obtenir un écran comme celui-ci :



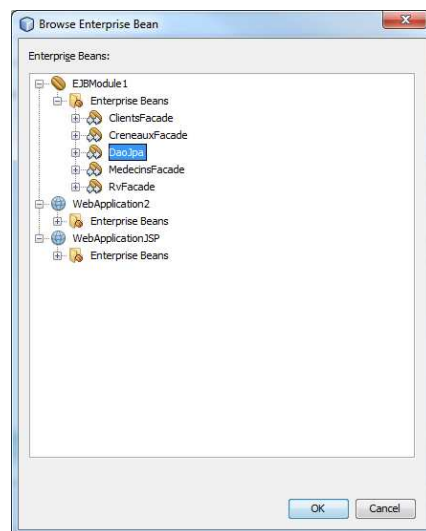
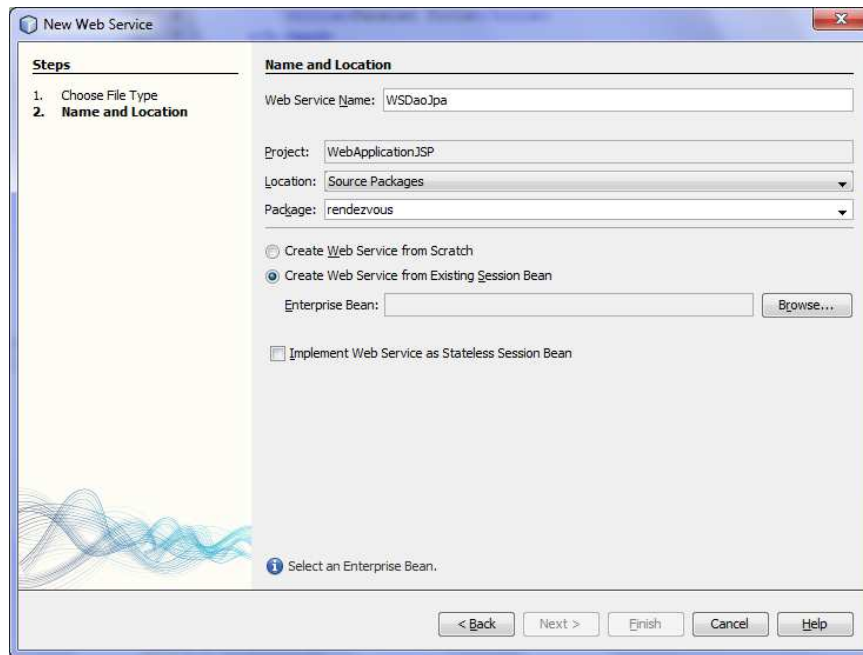
Supprimé :
(Sous JEE 6)

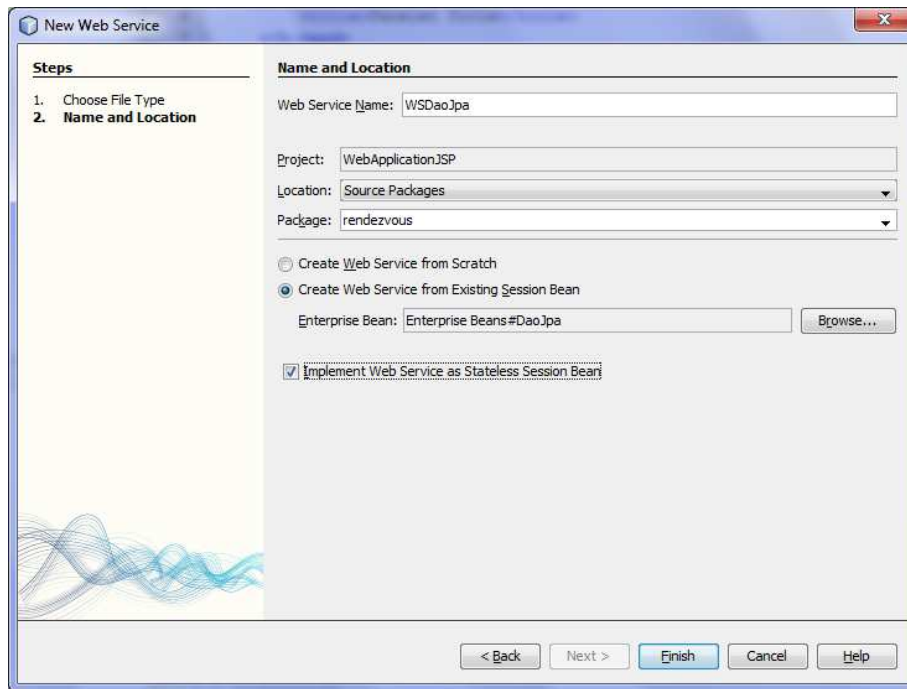


2.2. Faire un clic droit et ajouter un web service client.



Aller ajouter une session Bean en cliquant sur *Browser*.
Choisir comme nom de package **rendezvous** par exemple et nommer le WSDaoJpa.





Ouvrir le fichier WSDaoJpa.java et créer un web service comme suit :

```
package rendezvous;

import javax.ejb.EJB;
import javax.jws.*;
import jpa.*;
import dao.*;
import dao.IdaoLocal;
import java.util.*;

@WebService()
public class WSDaoJpa implements Idao {
    @EJB
    private IdaoLocal dao;

    // web service numero 1
    // liste des clients
    @WebMethod
    public List<Clients> getAllClients() {
        return dao.getAllClients();
    }

    // web service numero 2
    // liste des clients
    @WebMethod
    public List<Medecins> getAllMedecins() {
        return dao.getAllMedecins();
    }

    // liste des créneaux horaires d'un médecin donné
    // medecin : le médecin
    @WebMethod
    public List<Creneaux> getAllCreneaux(Medecins medecin) {
        return dao.getAllCreneaux(medecin);
    }

    // liste des Rv d'un médecin donné, un jour donné
    // medecin : le médecin, jour : le jour
    @WebMethod
    public List<Rv> getRvMedecinJour(Medecins medecin, String jour) {
        return dao.getRvMedecinJour(medecin, jour);
    }
}
```

```

// ajout d'un Rv, jour : jour du Rv
// creneau : créneau horaire du Rv, client : client pour lequel est pris le Rv
@WebMethod
public Rv ajouterRv(String jour, Creneaux creneau, Clients client) {
    return dao.ajouterRv(jour, creneau, client);
}

// suppression d'un Rv, rv : le Rv supprimé
@WebMethod
public void supprimerRv(Rv rv) {
    dao.supprimerRv(rv);
}

// récupérer un client donné
@WebMethod
public Clients getClientById(Long id) {
    return dao.getClientById(id);
}

// récupérer un médecin donné
@WebMethod
public Medecins getMedecinById(Long id) {
    return dao.getMedecinById(id);
}

// récupérer un Rv donné
@WebMethod
public Rv getRvById(Long id) {
    return dao.getRvById(id);
}

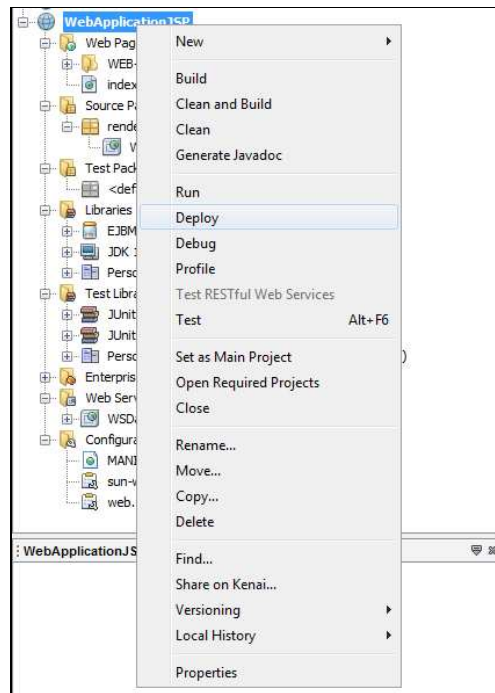
// récupérer un créneau donné
@WebMethod
public Creneaux getCreneauById(Long id) {
    return dao.getCreneauById(id);
}
}

```

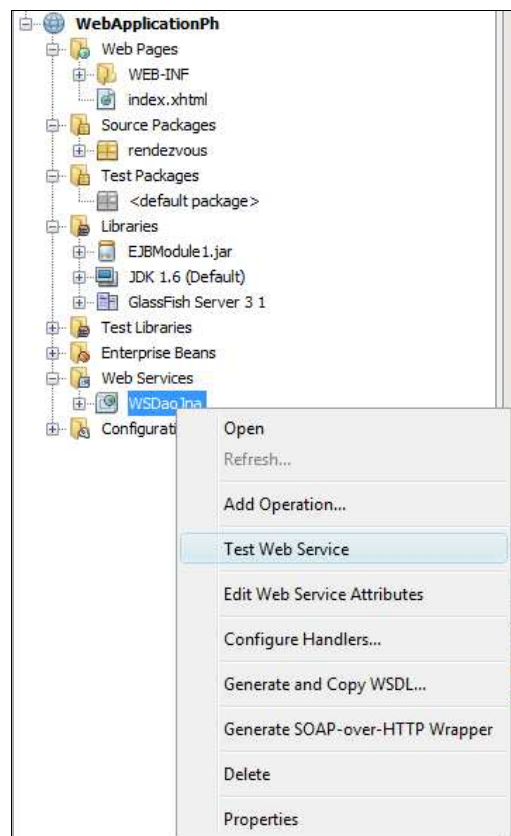
2.3. Compiler et déployer

Aller dans l'onglet Services et par un clic droit sur EJBModule1 faire Undeploy.

Faire un clic droit sur WebApplicationJSP puis « Clean and Build » et « Deploy ».



Il est alors possible de tester les services web en faisant un clic droit sur Web Service :

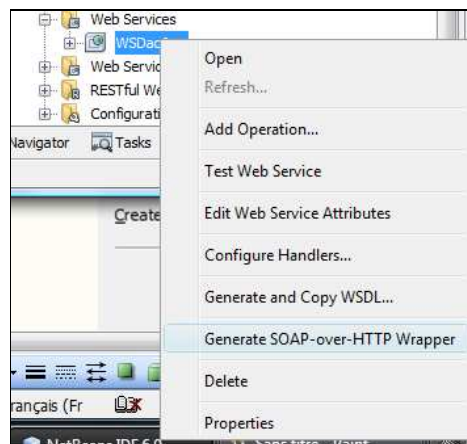


Le résultat devrait être :

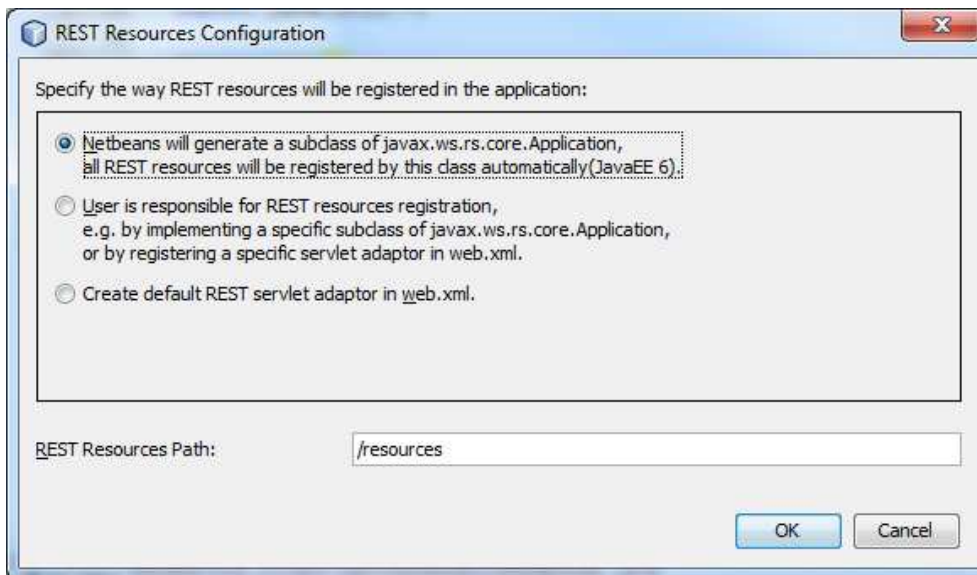


2.4. Générer les services SOAP

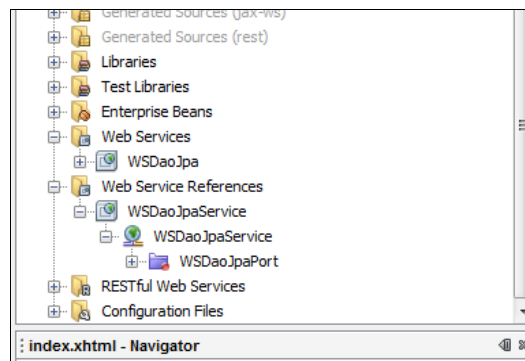
Aller sur Web Services et faire un clic droit sur WSDaoJpa.



Et faire Generate SOAP.



Finalement le projet ressemble à ce qui suit :



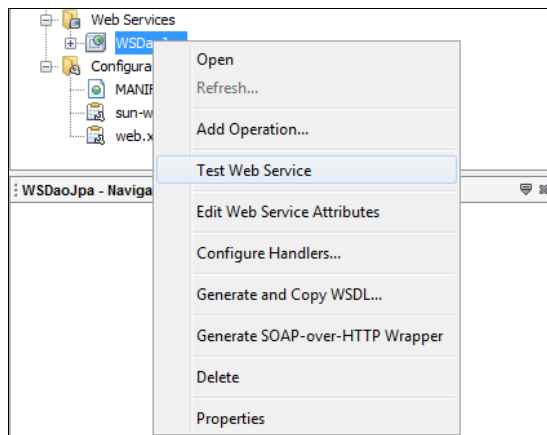
Remarquons la partie **Web Service References** :



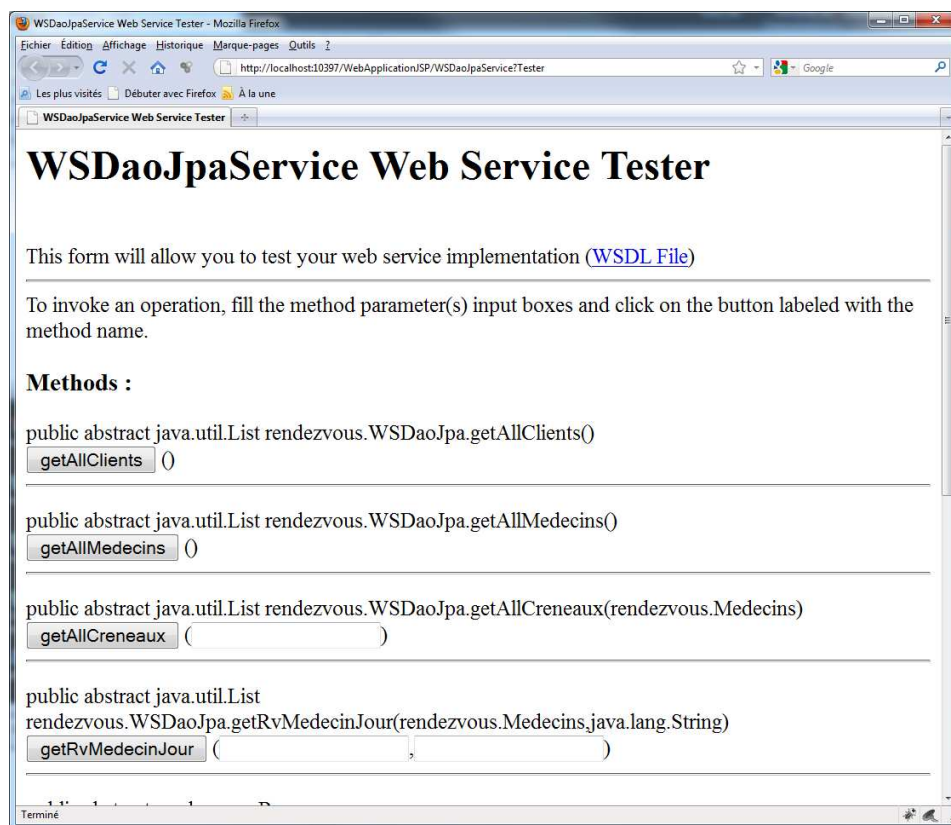
2.5. Tester le Web Service

Il faut déployer l'application.

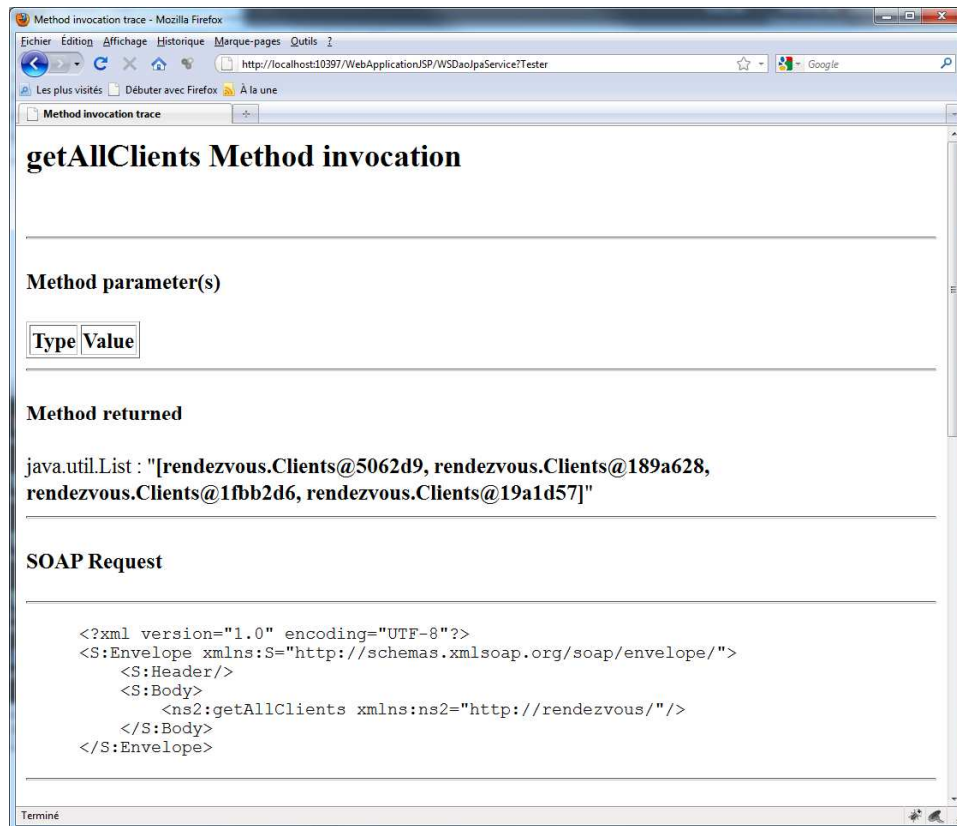
Faire un clic droit sur **WSDaoJpa** et choisir **Test Web Service**



Vous devriez obtenir ceci :

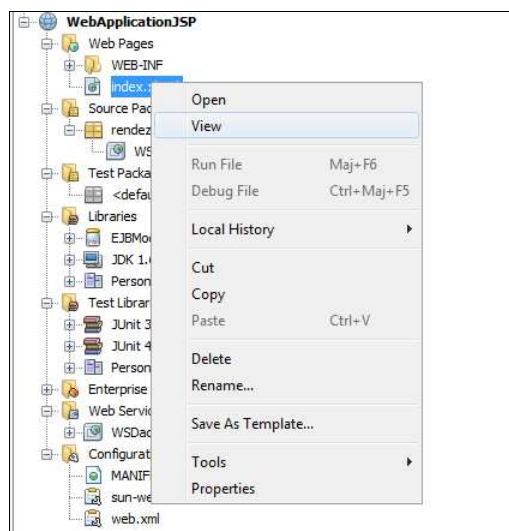


Et un clic sur **getAllClients** doit donner :

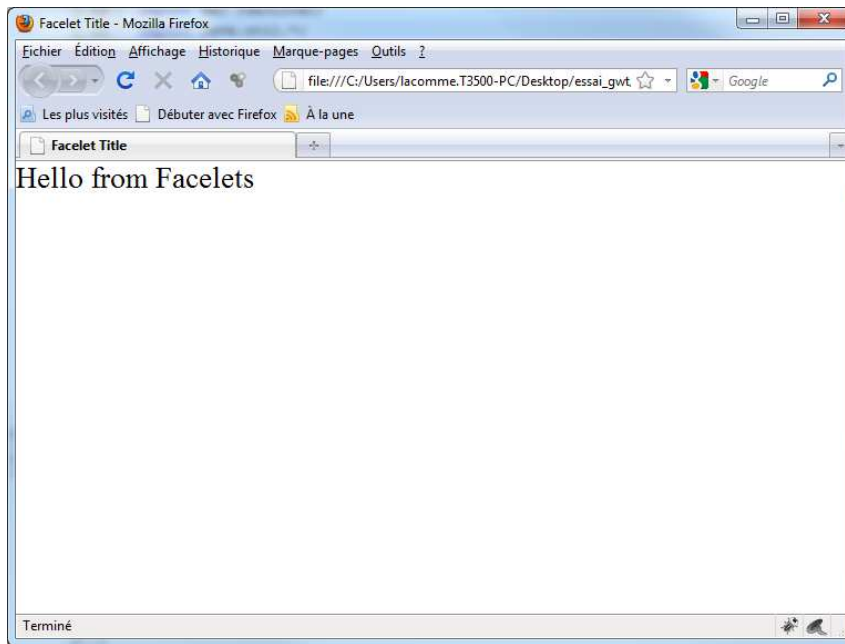


2.6. Vérifions que le fichier **index.xhtml** « fonctionne »

Faire un clic droit sur **index.xhtml** et choisir **view**.



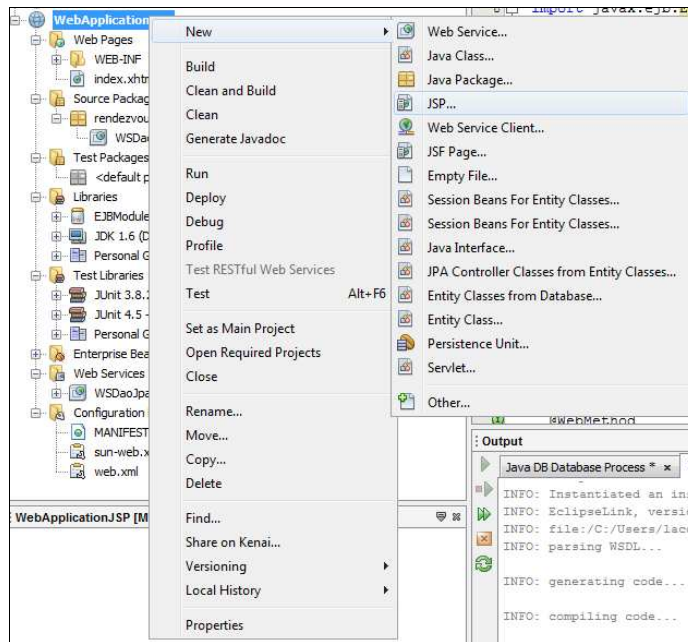
Le résultat doit être le suivant :



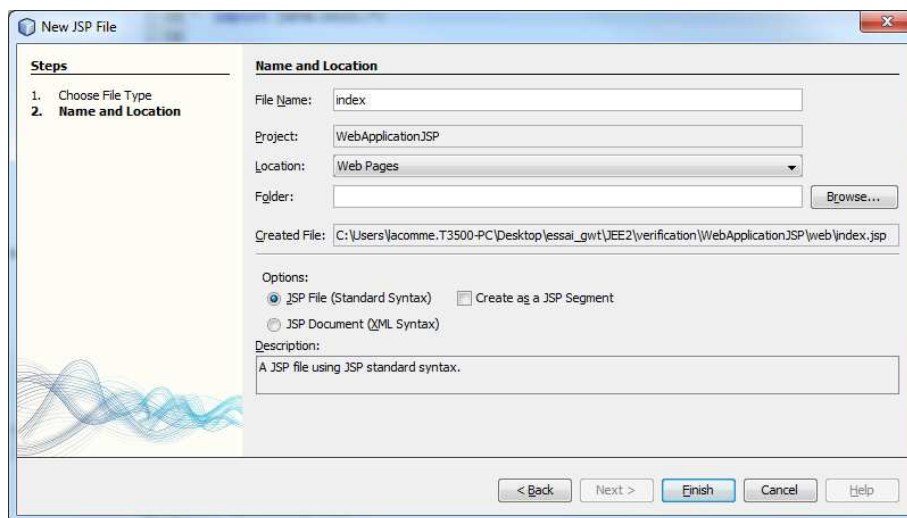
3) Création d'un fichier JSP

3.1. Créer un fichier nommé **index.jsp**

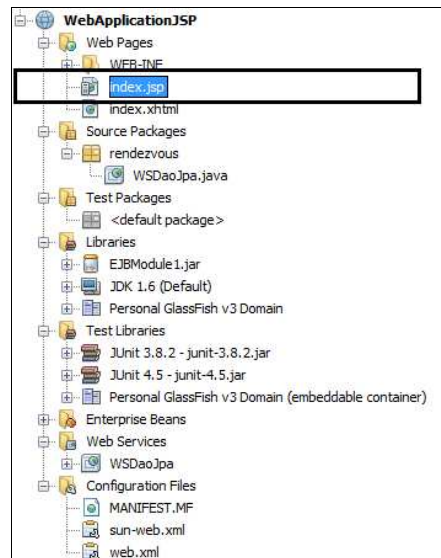
Faire un click droit sur WebApplicationJSP et choisir JSP.



Dans les options, choisir JSP File (Standard Syntax).



Un fichier nommé index.jsp apparaît dans la partie Web Pages du projet.



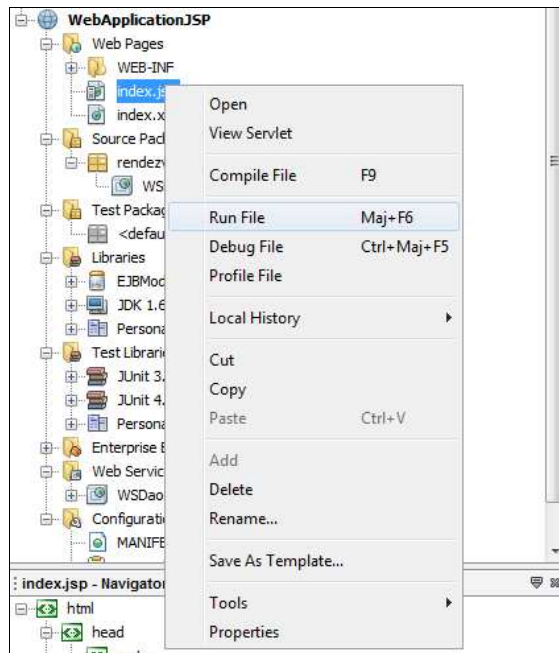
La page contient uniquement un « Hello World ».

```

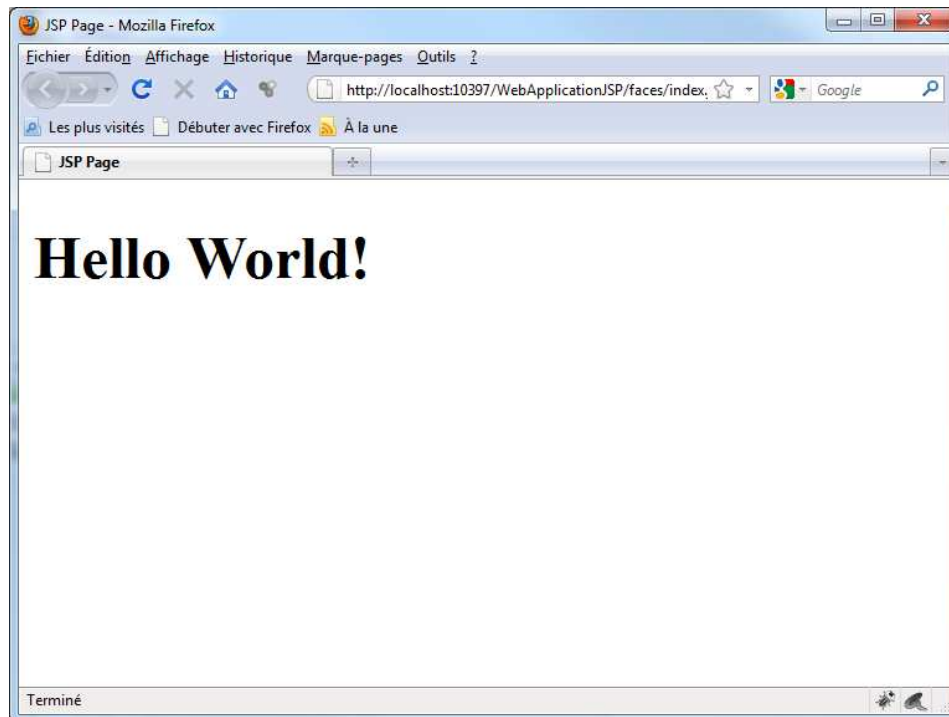
1  <%--
2      Document    : index
3      Created on  : 25 juil. 2010, 12:03:05
4      Author     : lacomme
5  --%>
6
7  <%%page contentType="text/html" pageEncoding="UTF-8"%>
8  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
9      "http://www.w3.org/TR/html4/loose.dtd">
10
11 <html>
12 <head>
13     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
14     <title>JSP Page</title>
15 </head>
16 <body>
17     <h1>Hello World!</h1>
18 </body>
19 </html>
20

```

Faire un clic droit sur **index.jsp** et choisir **Run File**.



Le résultat est alors le suivant :



3.2. Ajouter du code Java dans un fichier JSP

Les balises `<%` et `%>` permettent l'insertion de code Java.

Considérons par exemple le code suivant (les ajouts sont en rouge) :

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

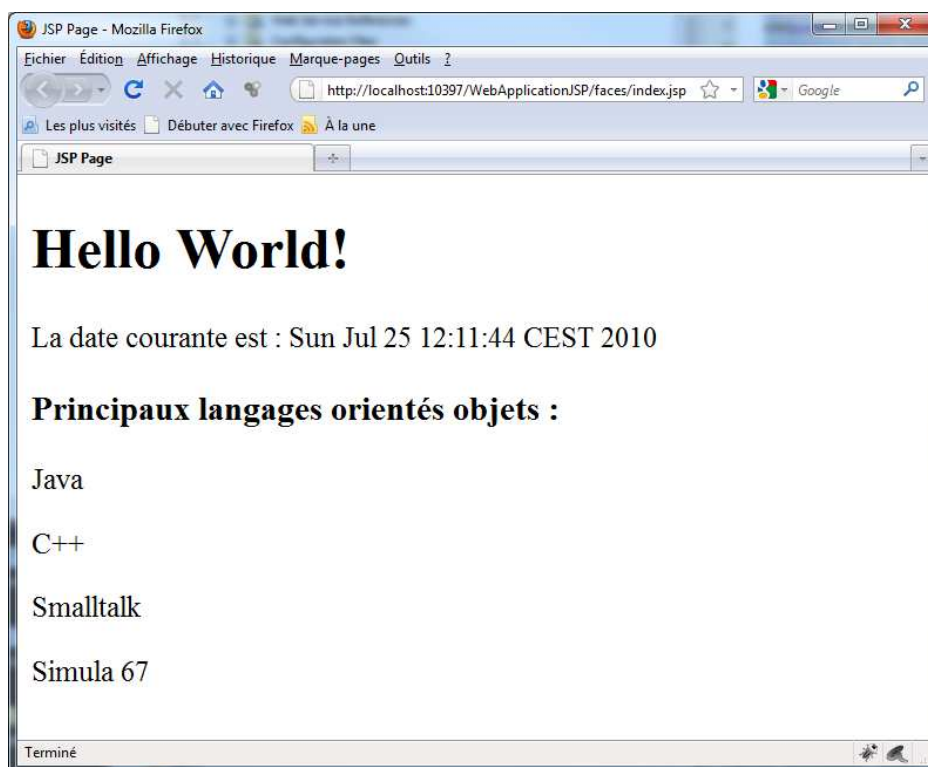
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h1>Hello World!</h1>

    La date courante est : <%= new java.util.Date() %>

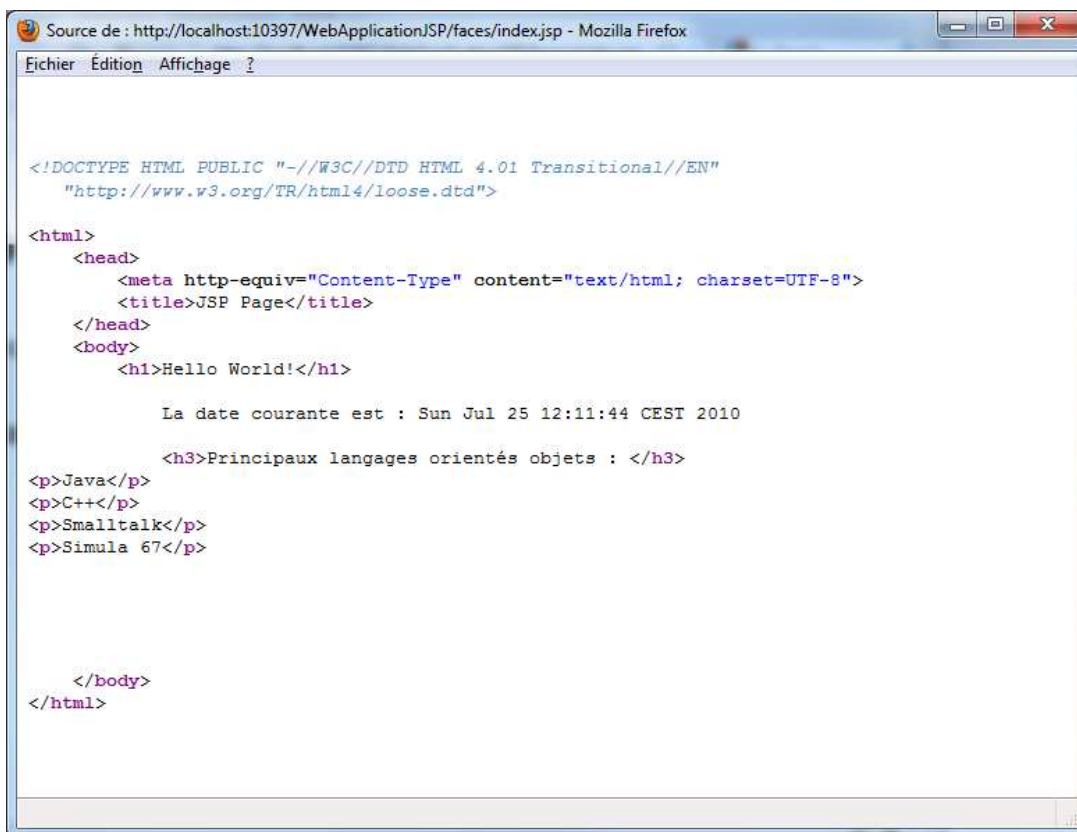
    <%
    String[] languages = {"Java", "C++", "Smalltalk", "Simula 67"};
    out.println("<h3>Principaux langages orientés objets : </h3>");
    for (int i=0; i < languages.length; i++) {
      out.println("<p>" + languages[i] + "</p>");
    }
    %>

  </body>
</html>
```

Le résultat sera :



Si on visualise le code de la page, on constate que le code Java n'apparaît pas ce qui montre clairement que le fichier JSP est interprété par le serveur et non pas par le client.



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h1>Hello World!</h1>

    La date courante est : Sun Jul 25 12:11:44 CEST 2010

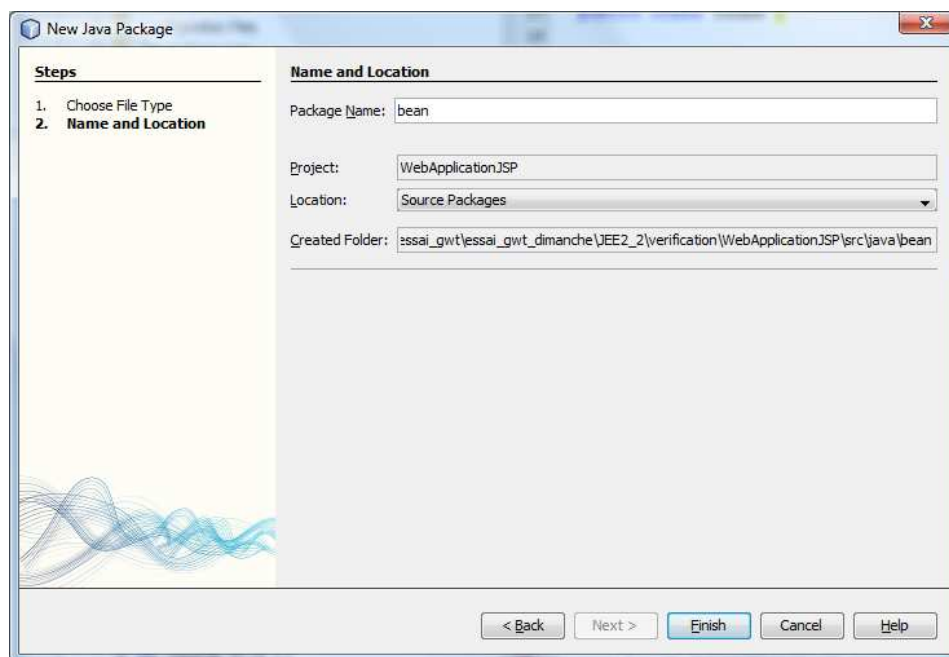
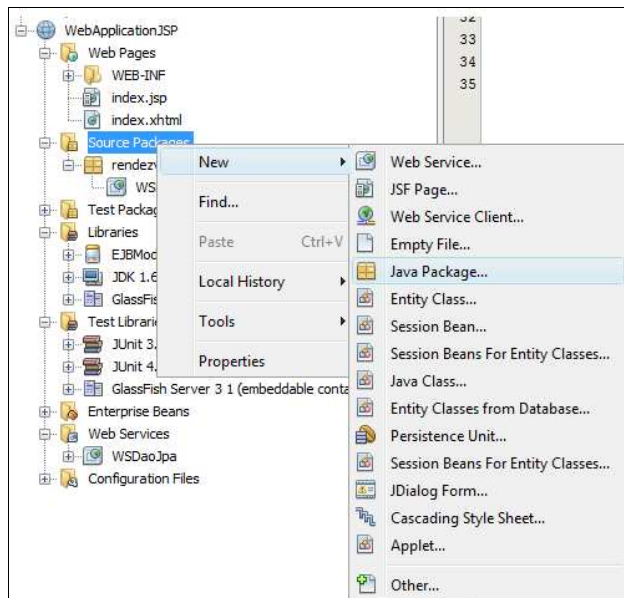
    <h3>Principaux langages orientés objets : </h3>
    <p>Java</p>
    <p>C++</p>
    <p>Smalltalk</p>
    <p>Simula 67</p>

  </body>
</html>
```

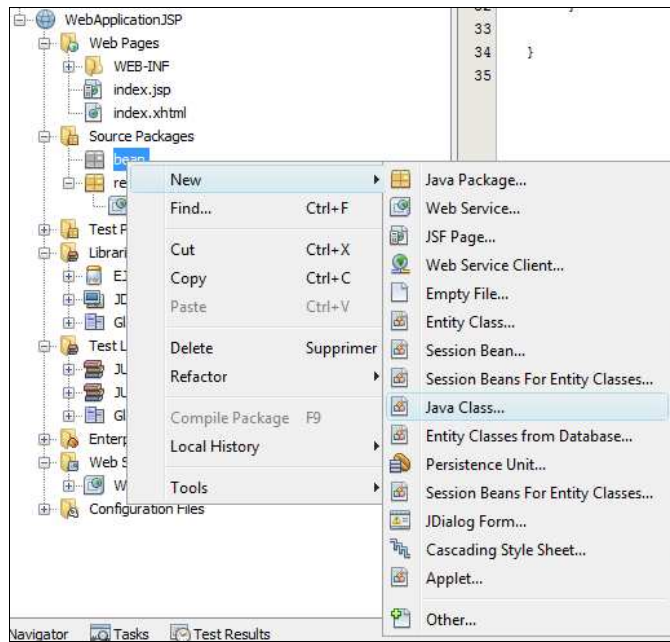
3.3. Création d'un fichier jsp pour consulter la base de données

Nous allons créer un package nommé **Bean** dans lequel nous allons définir une classe nommée **IndexClient** qui va servir d'intermédiaire vers JPA.

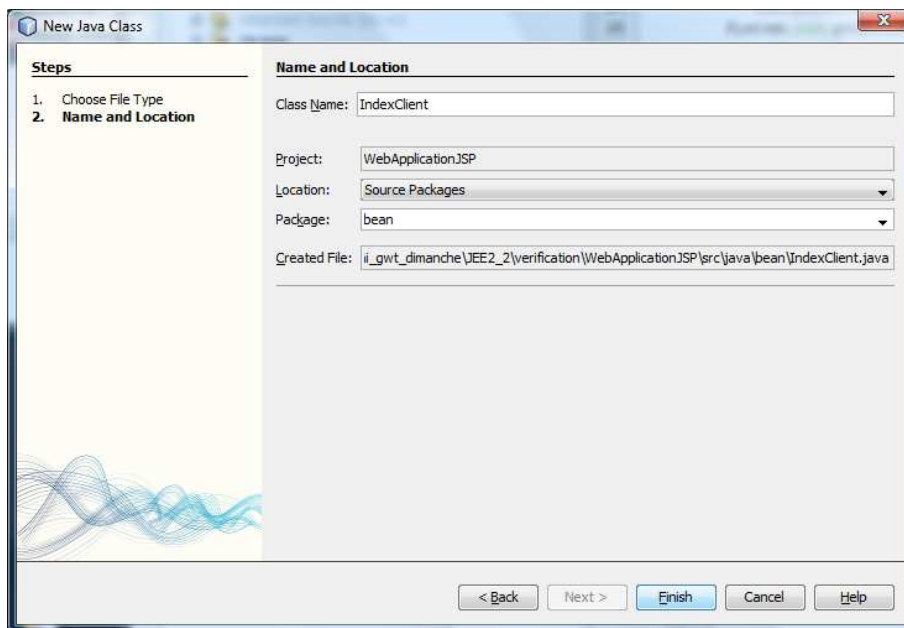
Faire un clic droit sur **Source Package** et choisir **Java Package**.



Faire ensuite un clic droit sur **bean** puis choisir **Java Class**.



Choisir comme nom **IndexClient**.



On peut alors définir la classe IndexClient comme suit :

```
package bean ;

import java.util.ArrayList;
import java.util.List;
import rendezvous.WSDaoJpaPort;
import rendezvous_client.WSDaoJpaService;
import rendezvous_client.Clients;

public class IndexClient {

    private List<Clients> clients;

    public void setClients(List<Clients> clients) {
        this.clients = clients;
    }

    public List<Clients> getClients()
    {
        WSDaoJpaService ws = new WSDaoJpaService ();
        System.out.println("--- affichage liste des clients ---");
        List<Clients> myArr = new ArrayList<Clients>();
        myArr = ws.getWSDaoJpaPort().getAllClients();
        return myArr;
    }
}
```

3.4. Modification de la page index.jsp

```
<%@page import="bean.IndexClient, rendezvous_client.Clients, java.util.*, RendezVous.*" %>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>

<script language="javascript">
    function getClients()
    {
        <%
            IndexClient ind = new IndexClient();
            StringBuffer val = new StringBuffer();
            for (Clients c : ind.getClients()) {
                val.append(c.getNom()+" <br/> ");
            }
            %>
            var chaine = "<%= new String(val)%>";
            <!--var elt = document.getElementById("res");
            elt.value = chaine;--%>
            document.getElementById("text").innerHTML= chaine;
        }
    }
</script>
</head>
<body>
<h1>Hello World!</h1>
    La date courante est : <%= new java.util.Date() %>

    <%
        String[] languages = {"Java", "C++", "Smalltalk", "Simula 67"};
        out.println("<h3>Principaux langages orientés objets : </h3>");
        for (int i=0; i < languages.length; i++) {
            out.println("<p>" + languages[i] + "</p>");
        }
    %>

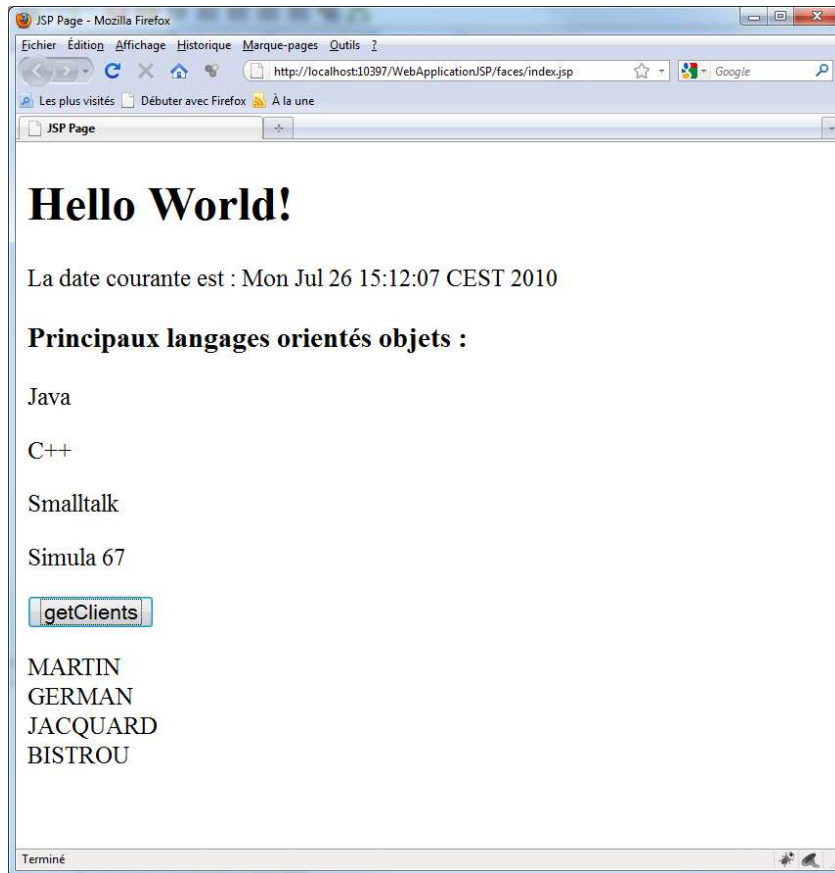
    <input type="submit" value="getClients" onclick="getClients()"/>
</body>
</html>
```

```
<p id="text">

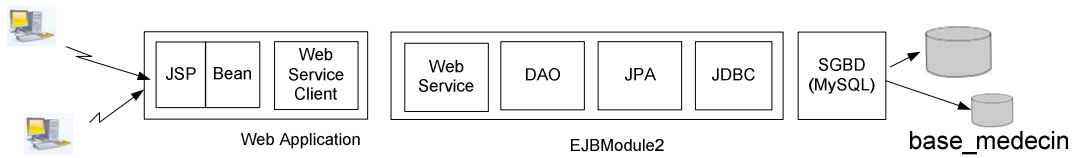
</p>

</body>
</html>
```

Ce qui donne comme résultat d'exécution :

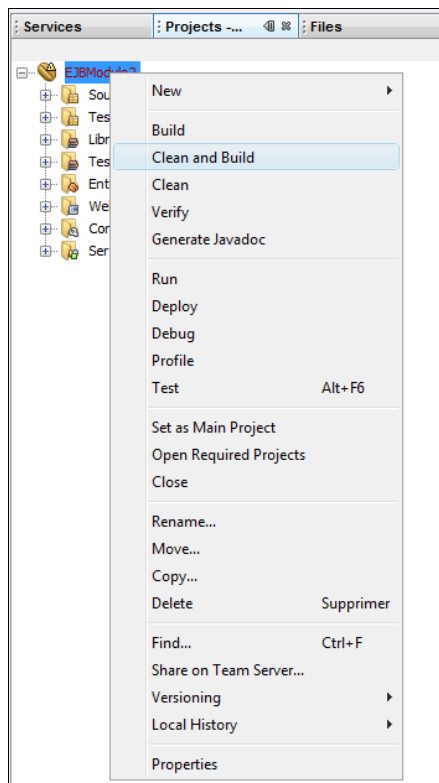


Solution 2.

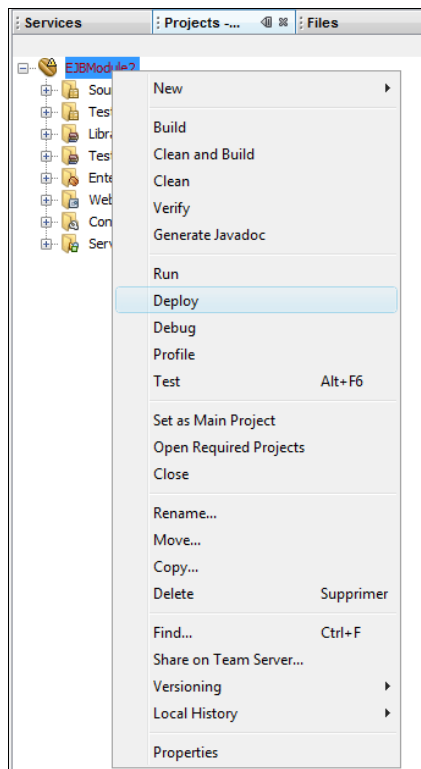


1) Déployer l'EJBModule2

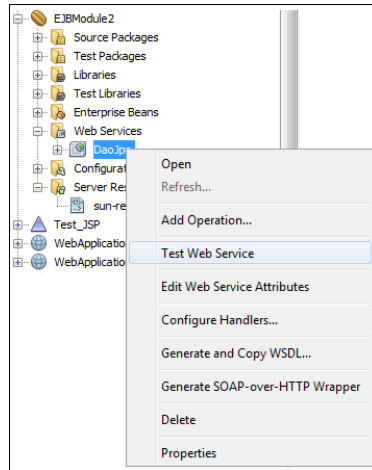
Vérifier que l'EJBModule2 compile.



Déployer l'EJB.

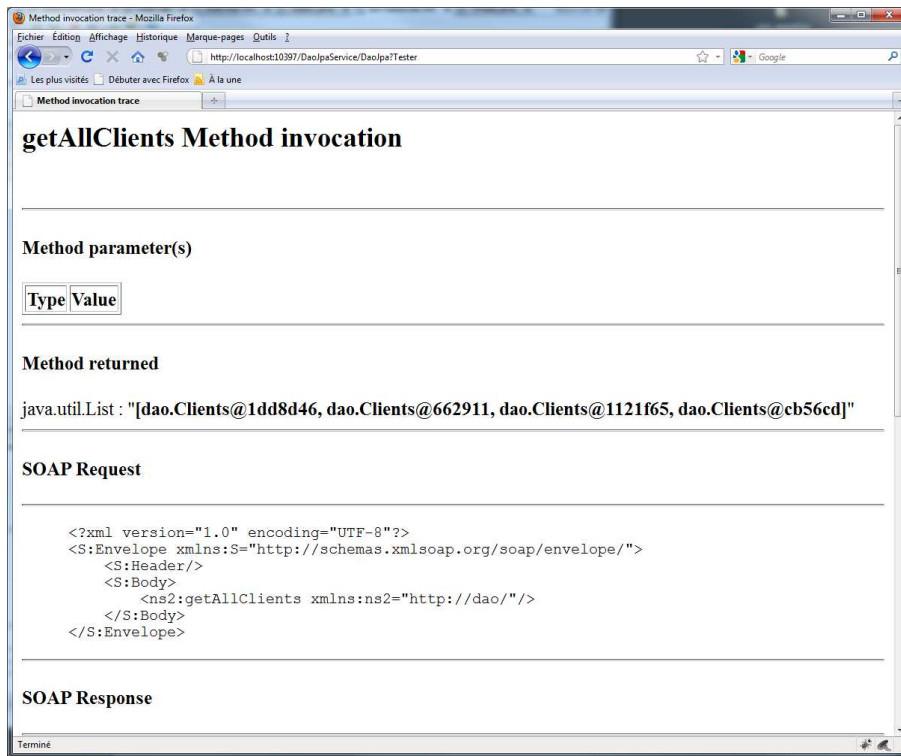


Faire un clic droit sur **DaoJpa** et choisir **Test Web Service**.

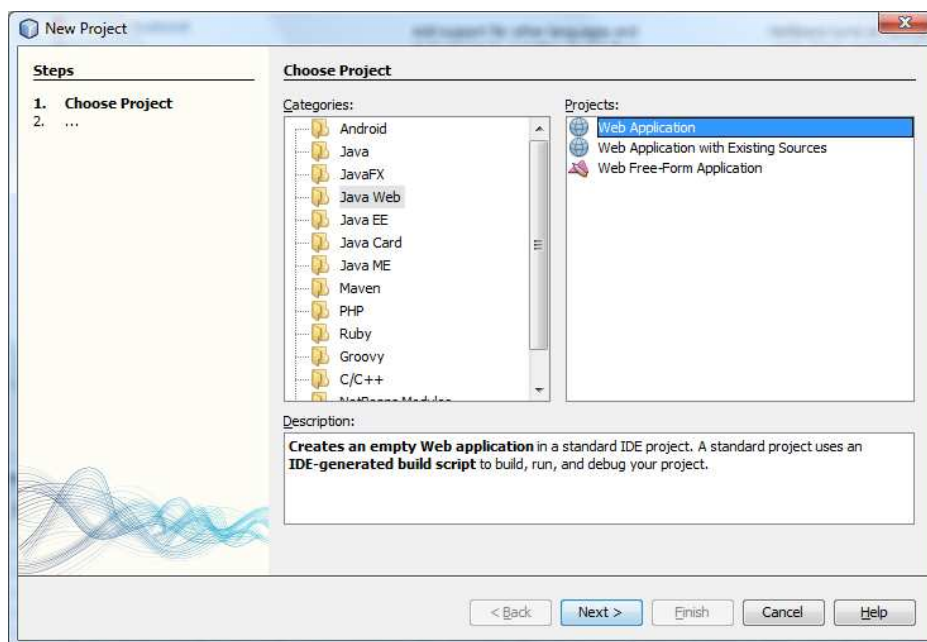


Vous devez obtenir ceci :





2) Créer une web application



New Web Application

add support for other languages and
add more source code

Steps

1. Choose Project
2. **Name and Location**
3. Server and Settings
4. Frameworks

Name and Location

Project Name:

Project Location:

Project Folder:

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☒ Set as Main Project

< Back Next > Finish Cancel Help

New Web Application

add support for other languages and
add more source code

Steps

1. Choose Project
2. Name and Location
3. **Server and Settings**
4. Frameworks

Server and Settings

Add to Enterprise Application:

Server:

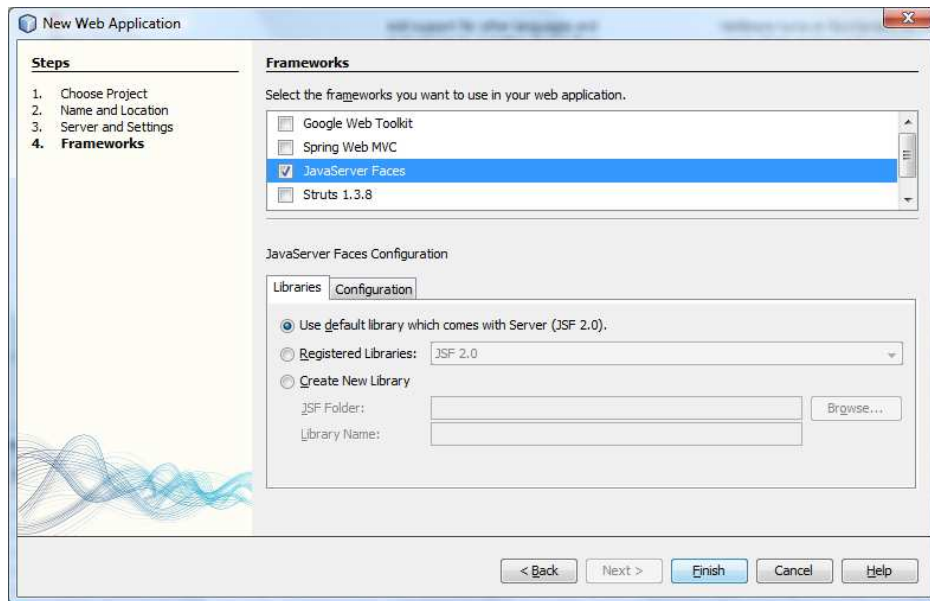
☐ Use dedicated library folder for server JAR files

Java EE Version:

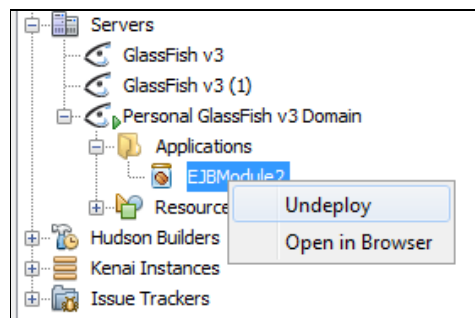
☐ Enable Contexts and Dependency Injection

Context Path:

< Back Next > Finish Cancel Help

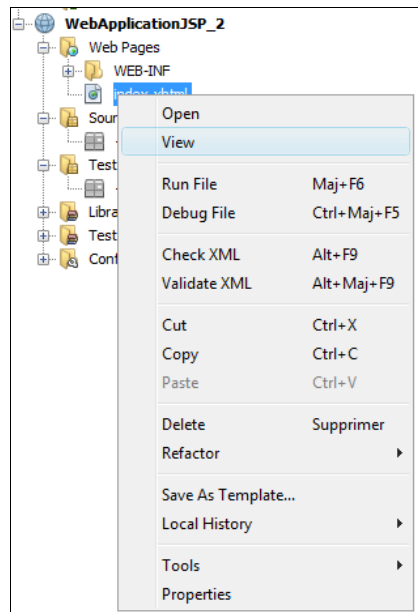


Pour terminer, supprimer EJBModule2 des applications déployées sur le serveur GlassFish.

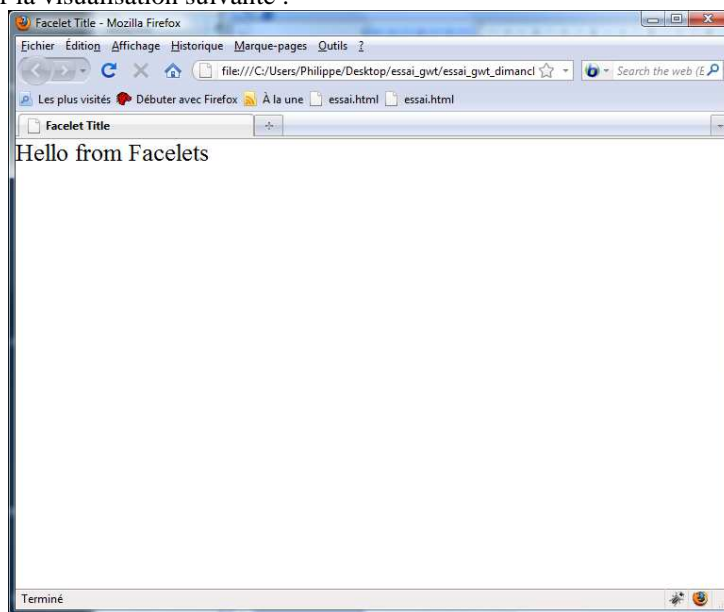


3) Création d'un service web client

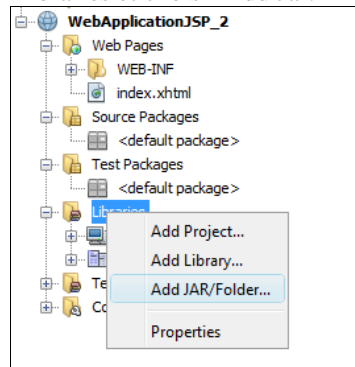
3.1. Remarquons le fichier **index.xml** qu'on peut visualiser simplement par un clic droit.



Ceci doit donner la visualisation suivante :

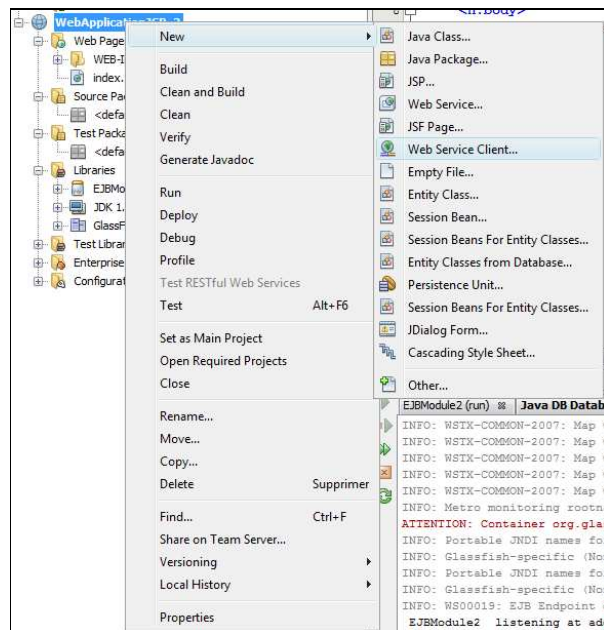


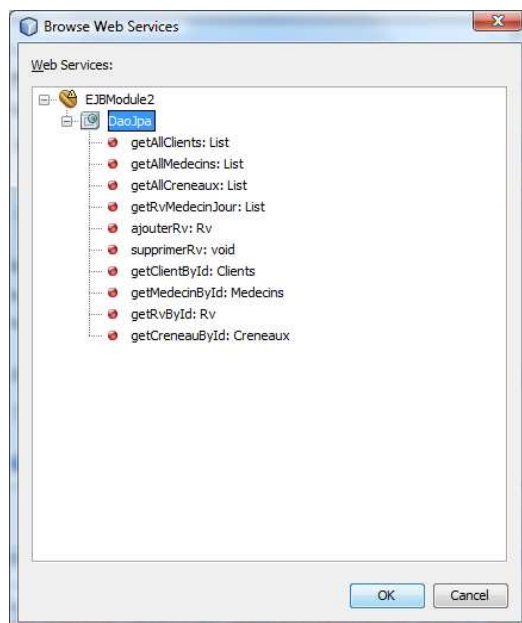
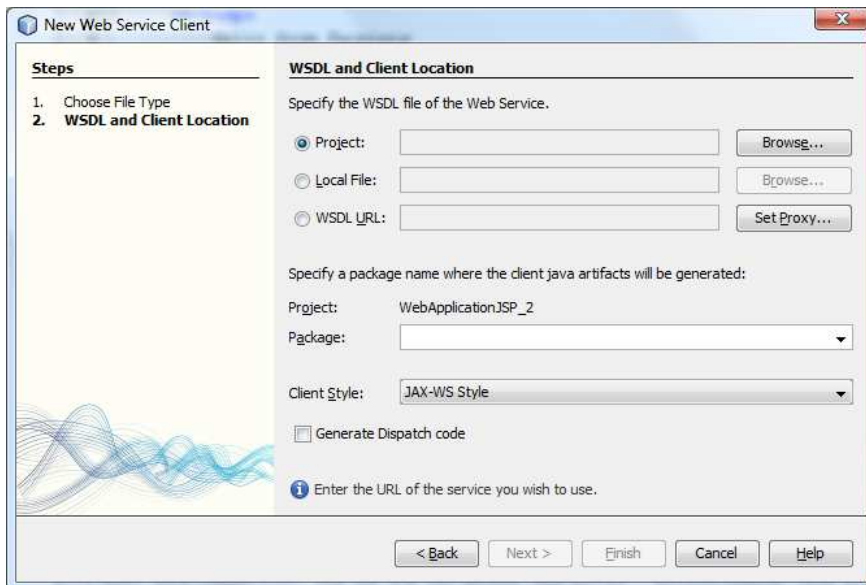
3.2. Ensuite faire un clic droit sur Libraries et choisir Add Jar.



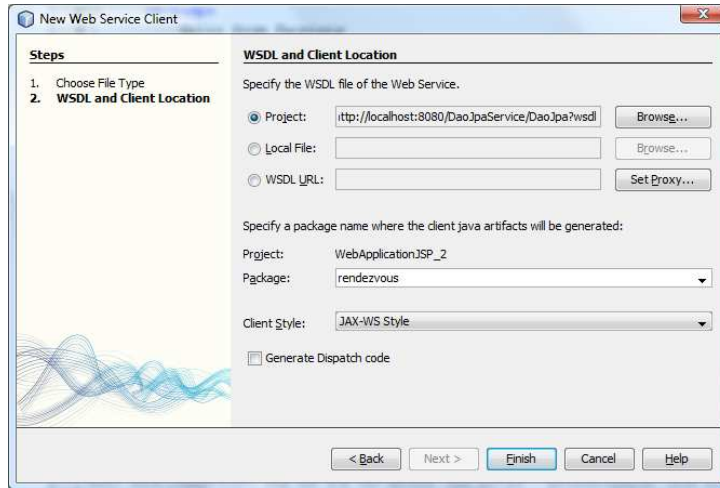
Choisir ensuite le .jar de l'EJBModule2.

3.3. Création d'un web service client

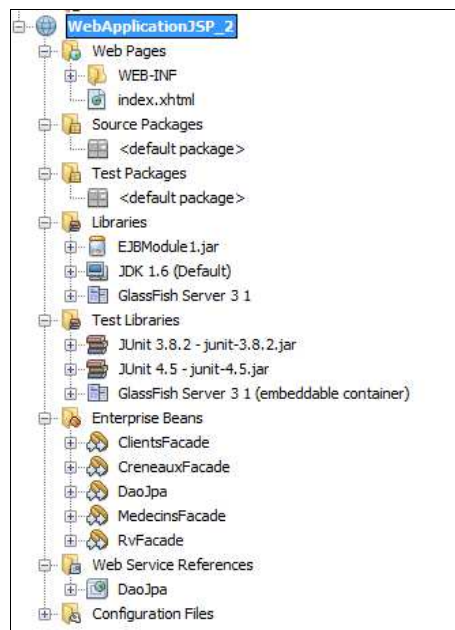




Attention, pensez à redéployer le EJBModule2 avant.



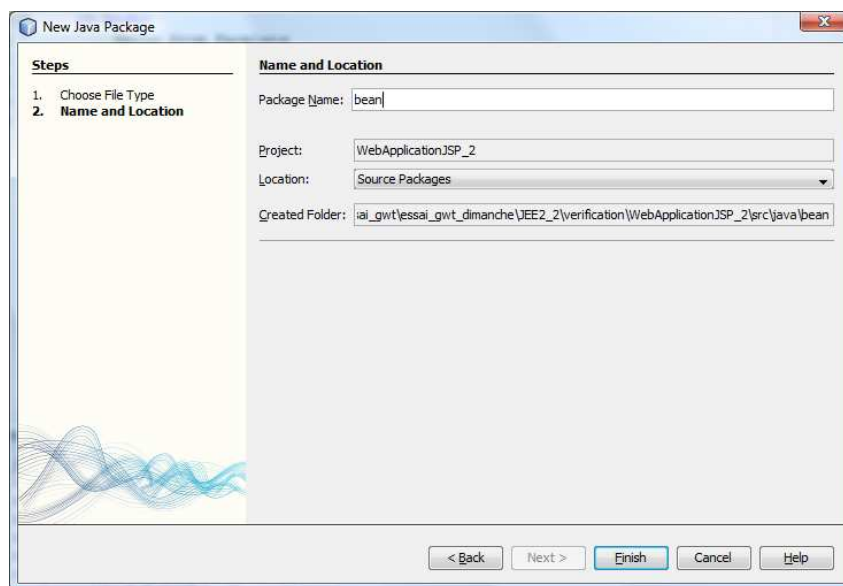
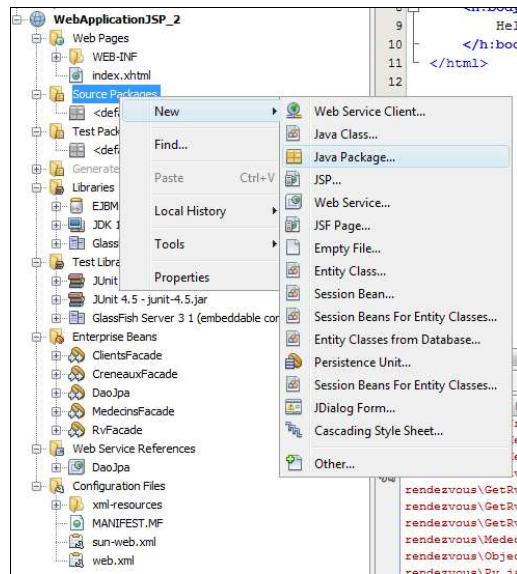
Le projet doit ressembler à ce qui suit :



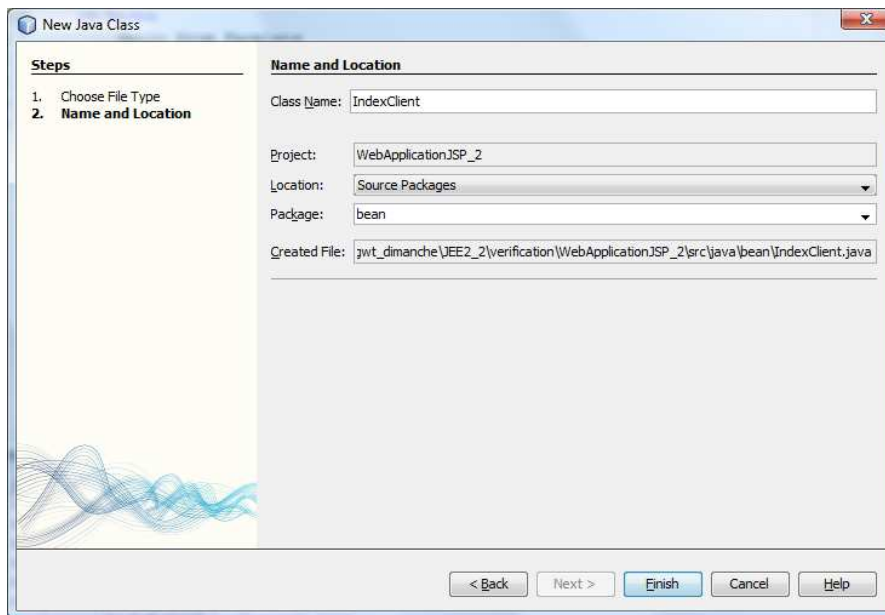
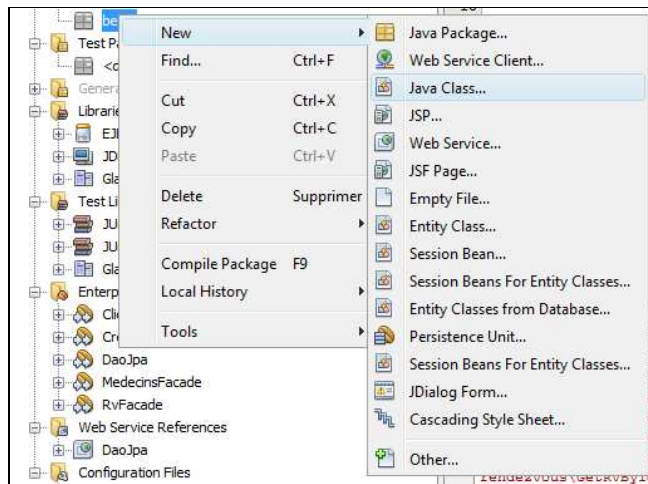
3.4. Création d'un package bean et d'une classe IndexClient

Nous allons créer un package nommé **Bean** dans lequel nous allons définir une classe nommée **IndexClient** qui va servir d'intermédiaire vers JPA.

Faire un clic droit sur **Source Package** et choisir **Java Package**.



Faire un clic droit sur **Source Package** sur **bean** et choisir Java Class



Ouvrir le fichier IndexClient.java et insérer le code suivant :

```
package bean;

import java.util.ArrayList;
import java.util.List;
import rendezvous.DaoJpaService;
import rendezvous.Clients;

public class IndexClient {

    private List<Clients> clients;

    public void setClients(List<Clients> clients) {
        this.clients = clients;
    }
}
```

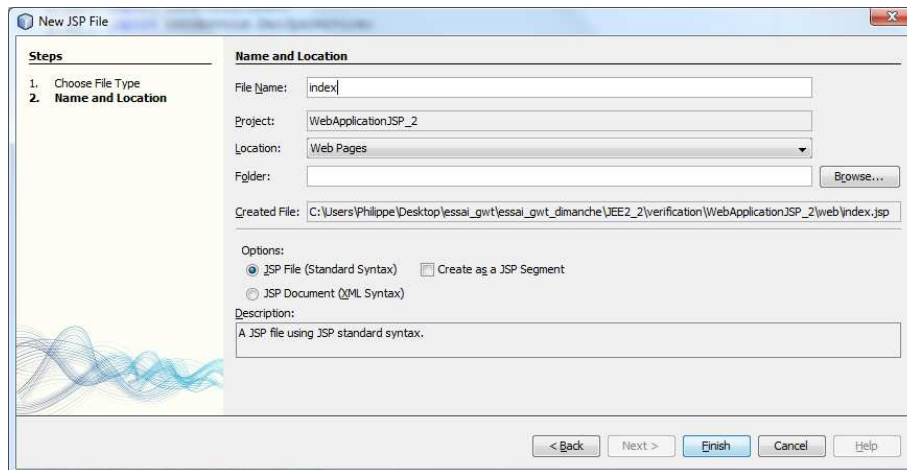
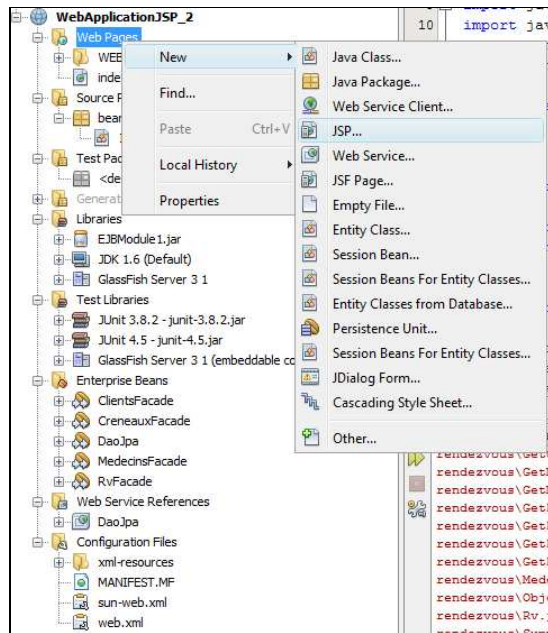
```

public List<Clients> getClients()
{
    DaoJpaService ws = new DaoJpaService ();
    System.out.println("--- affichage liste des clients ---");
    List<Clients> myArr = new ArrayList<Clients>();
    myArr = ws.getDaoJpaPort().getAllClients();
    return myArr;
}
}

```

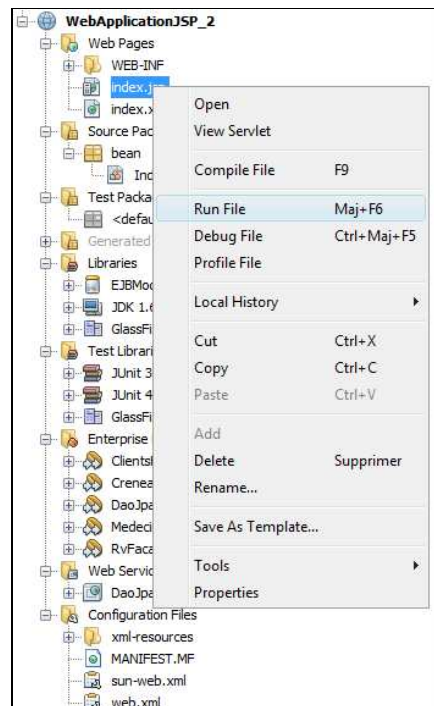
3.5. Création d'une page JSP

Faire un clic droit sur Web Page et choisir JSP.

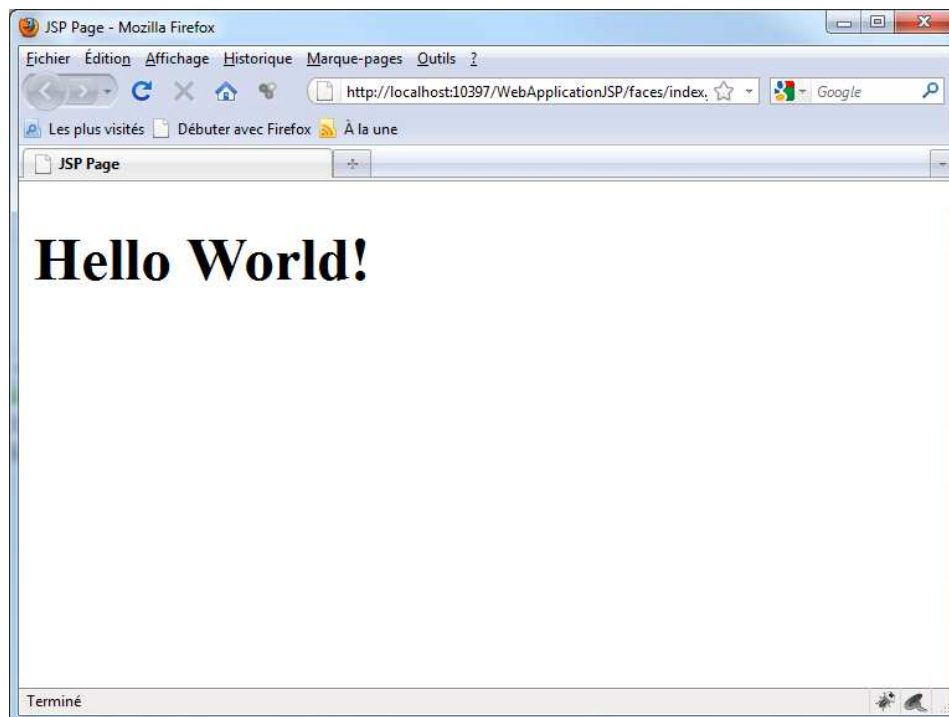


Il faut ensuite, faire « undeploy » de l'EJBModule2 et arrêter le serveur GlassFish.

Faire un clic droit sur index.jsp et choisir Run File.



Le résultat est alors le suivant :



3.6. Modification de la page JSP

Modifiez la page comme suit :

```
<%@page import="bean.IndexClient, rendezvous.Clients, java.util.*, rendezvous.*" %>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>

    <script language="javascript">
      function getClients()
      {
        <%
          IndexClient ind = new IndexClient();
          StringBuffer val = new StringBuffer();
          for (Clients c : ind.getClients()) {
            val.append(c.getNom()+" <br/> ");
          }
        %>
        var chaine = "<%= new String(val)%>";
        <!--var elt = document.getElementById("res");
        elt.value = chaine;--%>
        document.getElementById("text").innerHTML= chaine;
      }
    </script>
  </head>
  <body>
    <h1>Hello World!</h1>
    La date courante est : <%= new java.util.Date() %>

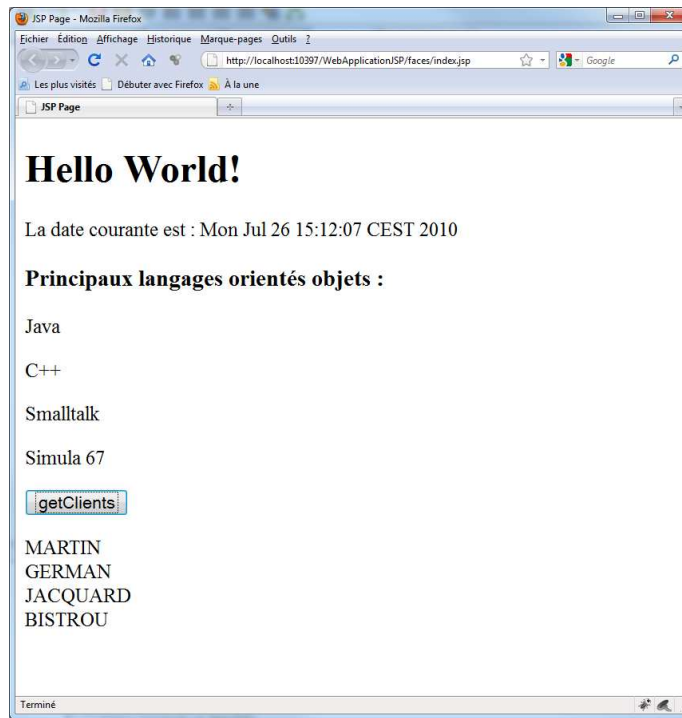
    <%
      String[] langages = {"Java", "C++", "Smalltalk", "Simula 67"};
      out.println("<h3>Principaux langages orientés objets : </h3>");
      for (int i=0; i < langages.length; i++) {
        out.println("<p>" + langages[i] + "</p>");
      }
    %>

    <input type="submit" value="getClients" onclick="getClients()"/>

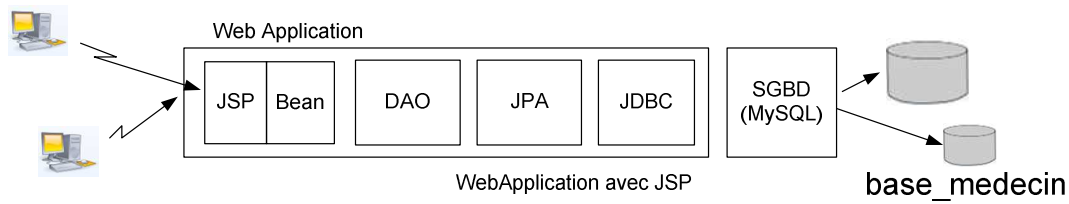
    <p id="text">

    </p>
  </body>
</html>
```

Ce qui donne comme résultat d'exécution :

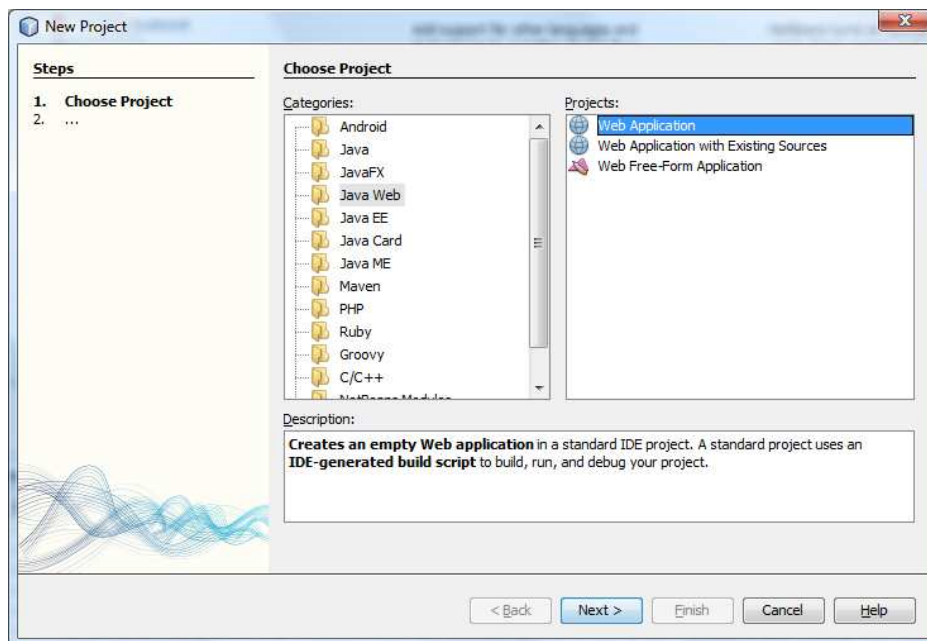


Solution 3.



La particularité de cette solution est que l'on n'utilise pas le concept d'EJB. En effet, les beans et pages web vont directement venir se greffer sur la couche DAO. Cette solution peut paraître plus longue car on recrée tout depuis le début (à partir de JPA) mais en fait, elle est rapide à mettre en place.

1) Créer une web application



New Web Application

Steps

1. Choose Project
2. **Name and Location**
3. Server and Settings
4. Frameworks

Name and Location

Project Name:

Project Location:

Project Folder:

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☒ Set as Main Project

< Back Next > Finish Cancel Help

New Web Application

Steps

1. Choose Project
2. Name and Location
3. **Server and Settings**
4. Frameworks

Server and Settings

Add to Enterprise Application:

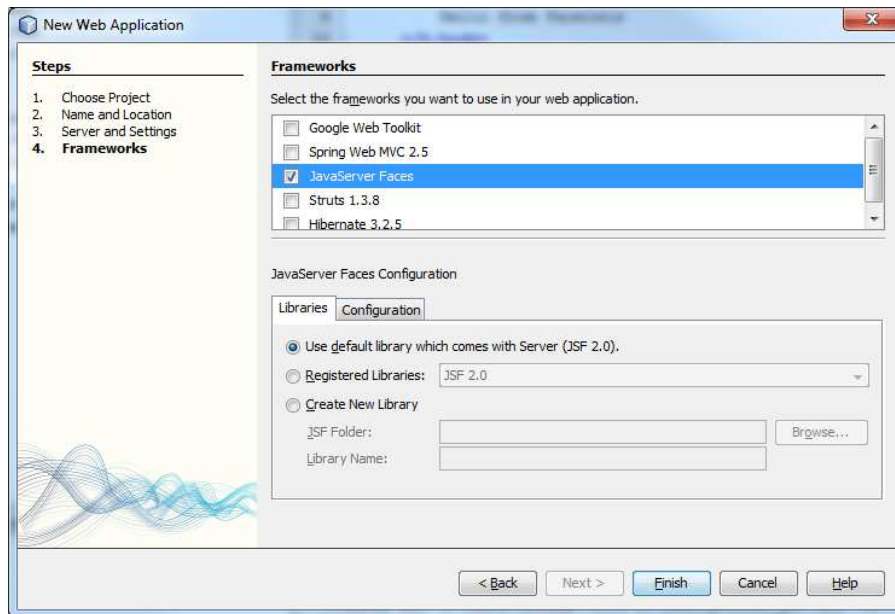
Server:

☐ Use dedicated library folder for server JAR files

Java EE Version:

Context Path:

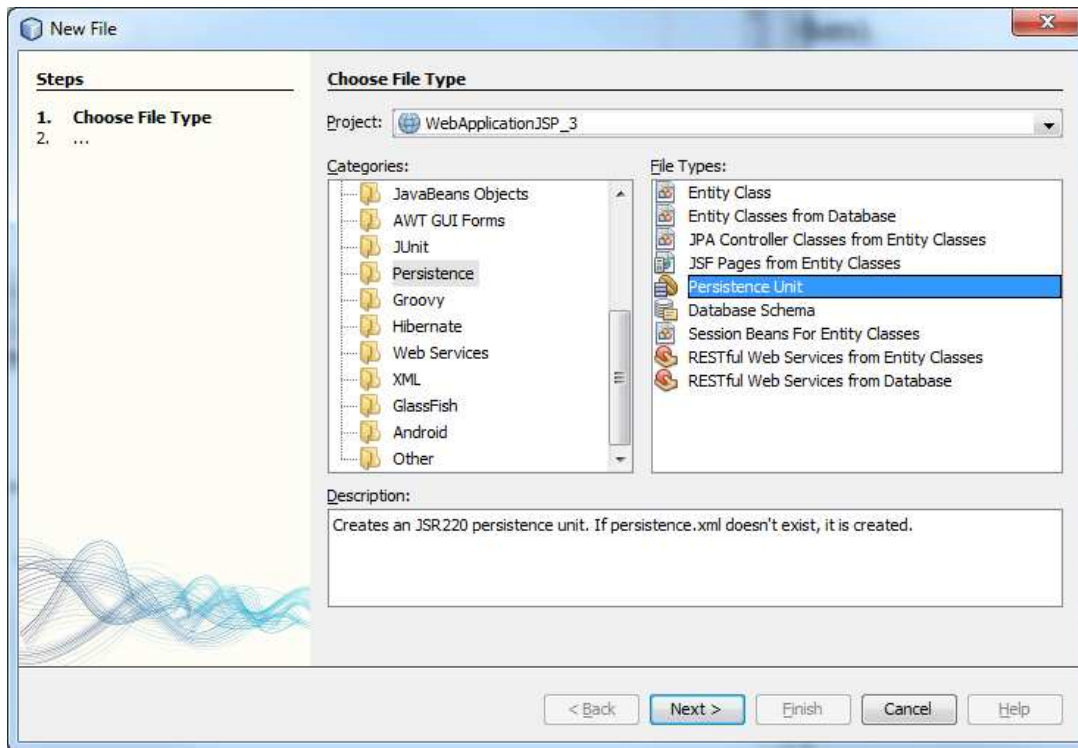
< Back Next > Finish Cancel Help



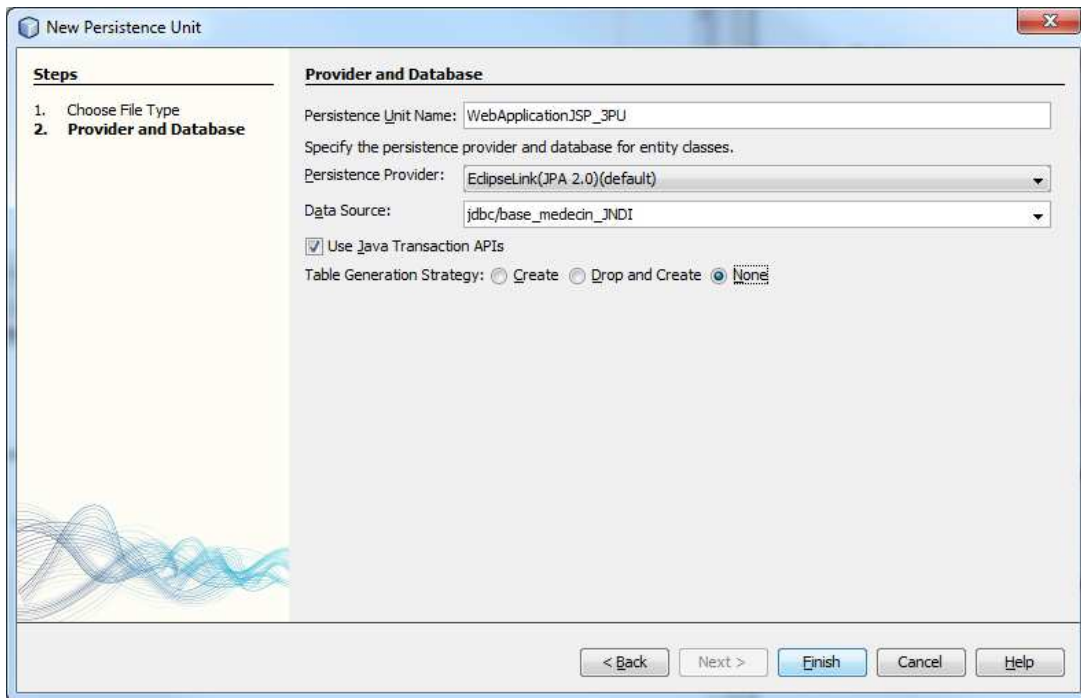
2) Créer la couche dao et jpa

Cette partie reprend certains point du cours 1.

Dans un **premier temps** nous allons créer une unité de persistance (faire Clic Droit sur WebApplicationJSP_3 et choisir New->Others).

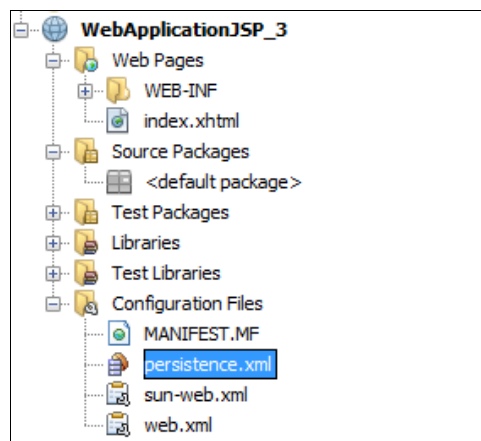


Choisir ensuite comme Data Source : base_medecin_JNDI
Et comme fournisseur de service de persistance EclipseLink.

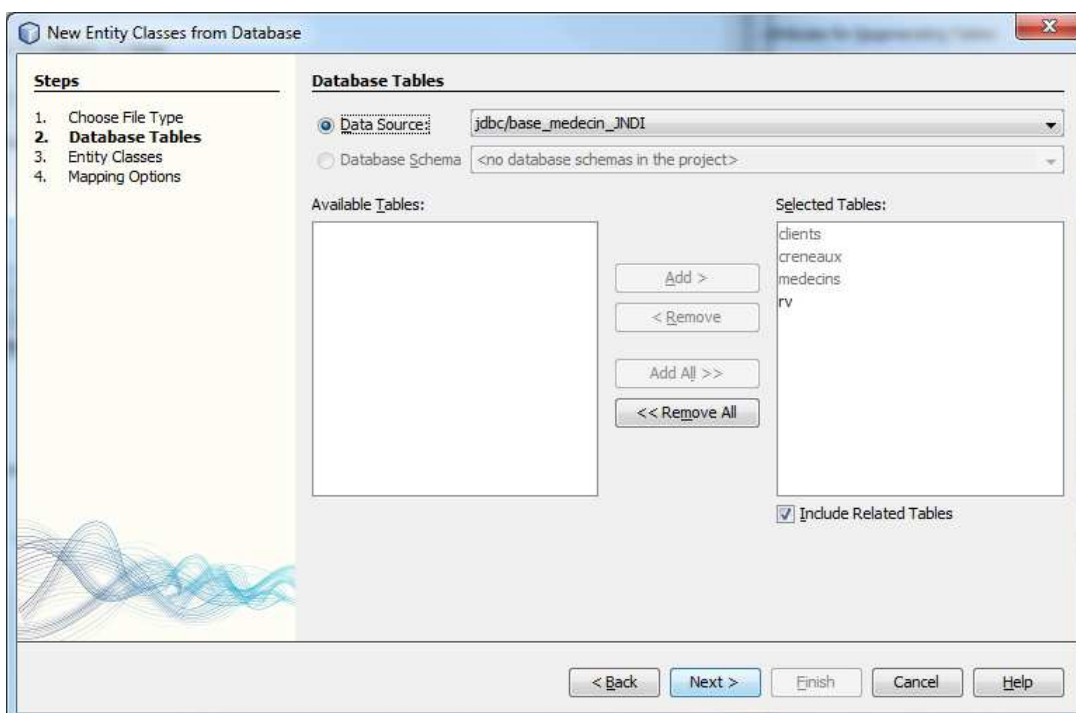
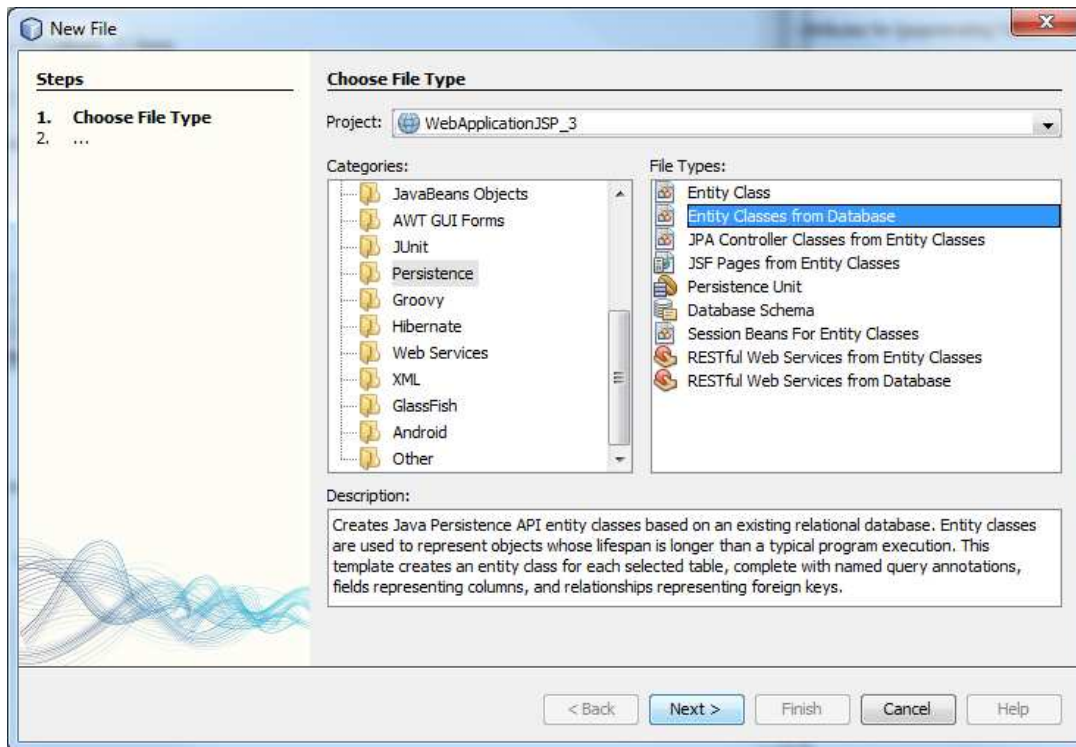


Attention à ne pas choisir une stratégie englobant la génération des tables. Elles existent déjà grâce au script SQL que nous avons utilisé au départ.

Dans la partie « Configuration Files », le fichier **persistence.xml** apparaît.



Dans un **deuxième temps**, nous allons créer des entités JPA. Comme précédemment faire Clic Droit / New / Other.



Etant donné qu'il s'agit de la génération d'entités JPA, on peut choisir JPA comme nom de package.

New Entity Classes from Database

Steps

1. Choose File Type
2. Database Tables
- 3. Entity Classes**
4. Mapping Options

Entity Classes

Specify the names and the location of the entity classes.

Class Names:

Database Table	Class Name
clients	Clients
creneaux	Creneaux
medecins	Medecins
rv	Rv

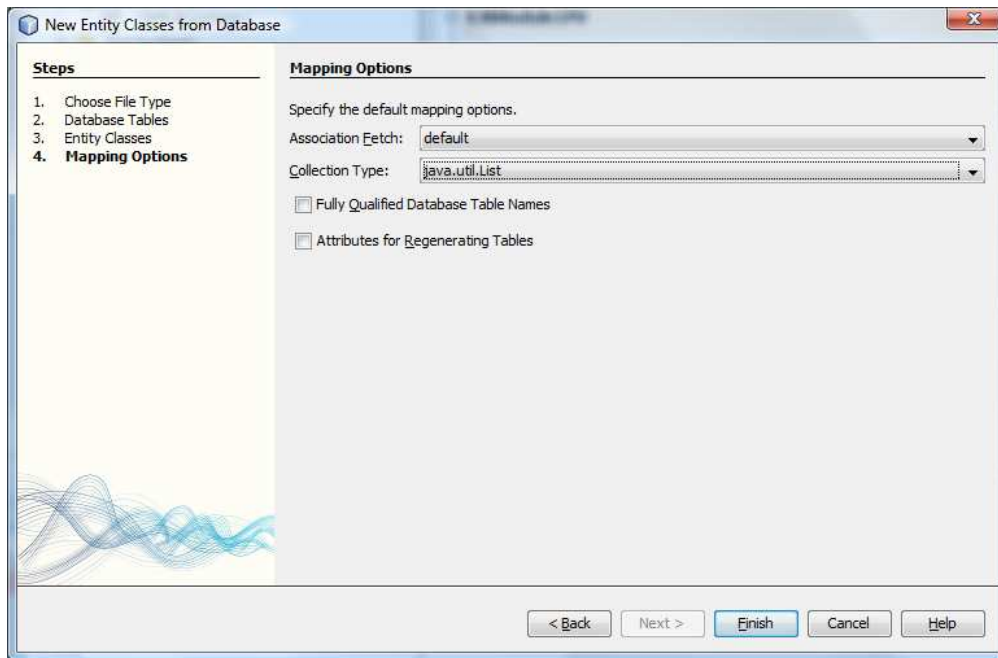
Project: WebApplicationJSP_3

Location: Source Packages

Package: jpa

☒ Generate Named Query Annotations for Persistent Fields

< Back Next > Finish Cancel Help

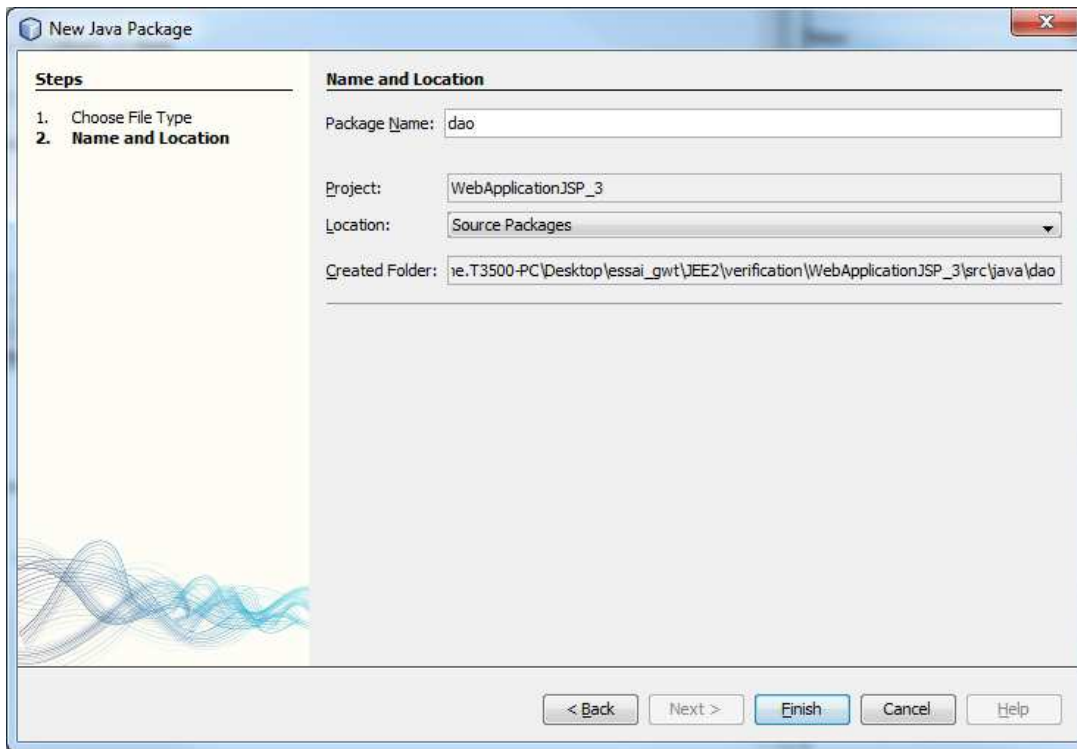


Veillez à bien choisir « **java.util.List** » pour Collection Type (pour avoir plus de lisibilité).

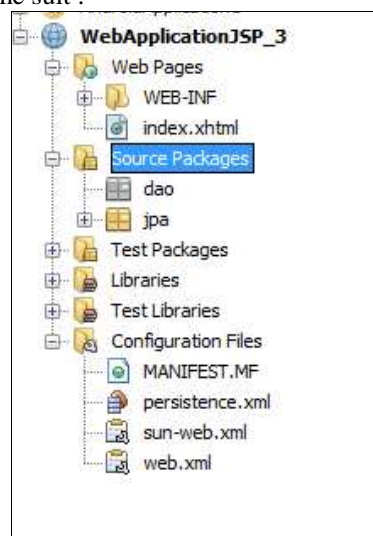
Dans la classe Rv nous ajoutons un nouveau constructeur permettant de créer un rendez vous en donnant une date, un client et un jour.

```
public Rv(Date jour, Clients client, Creneaux creneau)
{
    this.jour = jour ;
    this.idClient = client ;
    this.idCreneau = creneau ;
}
```

Enfin nous allons créer un package nommée **dao**, qui représente l'interface de l'EJB pour l'accès aux données.



Le projet se présente alors comme suit :



Ajoutons une classe **DaoJpa** qui contiendra l'intelligence métier.

Il faut rajouter le code suivant dans le fichier DaoJpa.java.

```

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package dao;

import java.text.SimpleDateFormat;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import jpa.Clients;
import jpa.Creneaux;
import jpa.Medecins;
import jpa.Rv;

/**
 *
 * @author lacomme
 */
public class DaoJpa {
    private EntityManager em;

    private EntityManagerFactory emf;

    private EntityTransaction tx ;

    public void init()
    {
        emf = Persistence.createEntityManagerFactory("WebApplicationJSP_3PU");

        em = emf.createEntityManager();

        tx = em.getTransaction();

        tx.begin();
    }

    public void close()
    {
        em.close();

        emf.close();
    }

    // liste des clients
    public List<Clients> getAllClients() {
        try {
            return em.createQuery("select c from Clients c").getResultList();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    // liste des m  decins
    public List<Medecins> getAllMedecins() {
        try {
            return em.createQuery("select m from Medecins m").getResultList();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    // liste des cr  neaux horaires d'un m  decin donn  
    // medecin : le m  decin
    public List<Creneaux> getAllCreneaux(Medecins medecin) {
        try {
            return em.createQuery("select c from Creneaux c join c.medecin m where m.id=:idMedecin").setParameter("idMedecin", medecin.getId()).getResultList();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}

```



```

// liste des Rv d'un mÃ©decin donnÃ©, un jour donnÃ©
// medecin : le mÃ©decin
// jour : le jour
public List<Rv> getRvMedecinJour(Medecins medecin, String jour) {
    try {
        return em.createQuery("select rv from Rv rv join rv.creneau c join c.medecin m where
m.id=:idMedecin and rv.jour=:jour").setParameter("idMedecin", medecin.getId()).setParameter("jour",
new SimpleDateFormat("yyyy:MM:dd").parse(jour)).getResultList();
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

// ajout d'un Rv
public Rv ajouterRv(String jour, Creneaux creneau, Clients client) {
    try {
        Rv rv = new Rv(new SimpleDateFormat("yyyy:MM:dd").parse(jour), client, creneau);
        em.persist(rv);
        return rv;
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

// suppression d'un Rv
// rv : le Rv supprimÃ©
public void supprimerRv(Rv rv) {
    try {
        em.remove(em.merge(rv));
    } catch (Exception e) {
        e.printStackTrace();
    }
}

// recuperer un client donnÃ©
public Clients getClientById(Long id) {
    try {
        return (Clients) em.find(Clients.class, id);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

// recuperer un mÃ©decin donnÃ©
public Medecins getMedecinById(Long id) {
    try {
        return (Medecins) em.find(Medecins.class, id);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

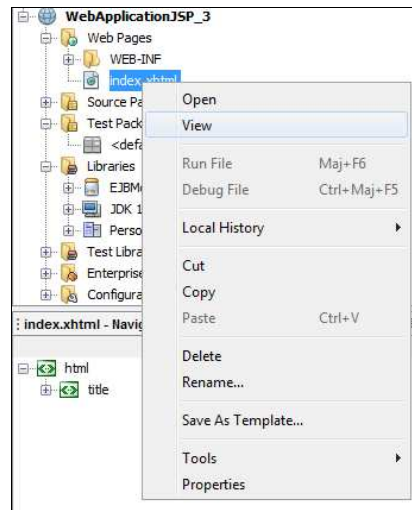
// recuperer un Rv donnÃ©
public Rv getRvById(Long id) {
    try {
        return (Rv) em.find(Rv.class, id);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

// recuperer un crÃ©neau donnÃ©
public Creneaux getCreneauById(Long id) {
    try {
        return (Creneaux) em.find(Creneaux.class, id);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
}

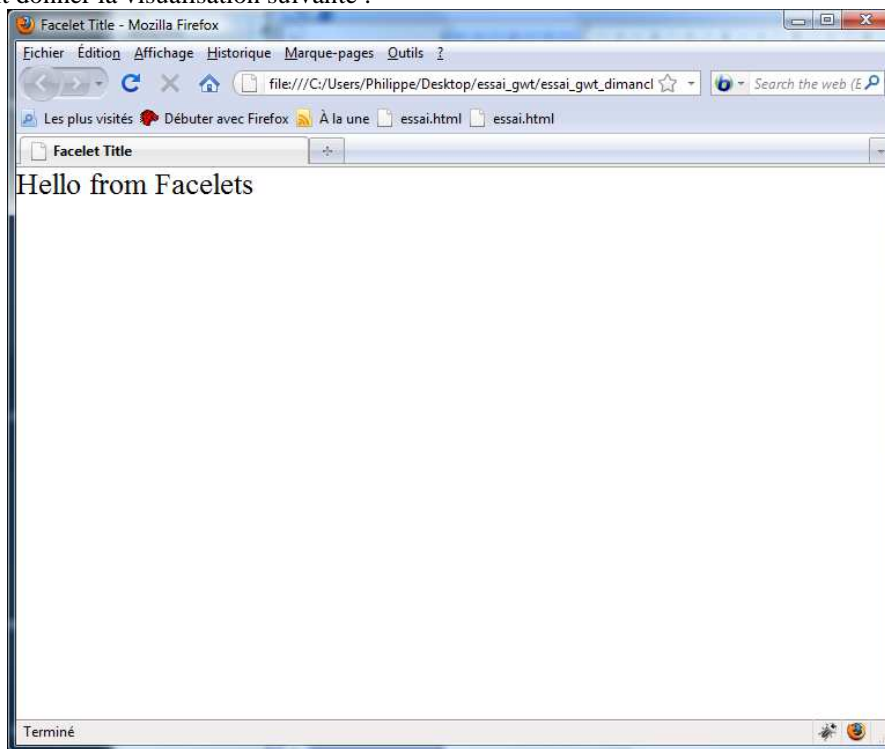
```

3) Ajout de page JSP

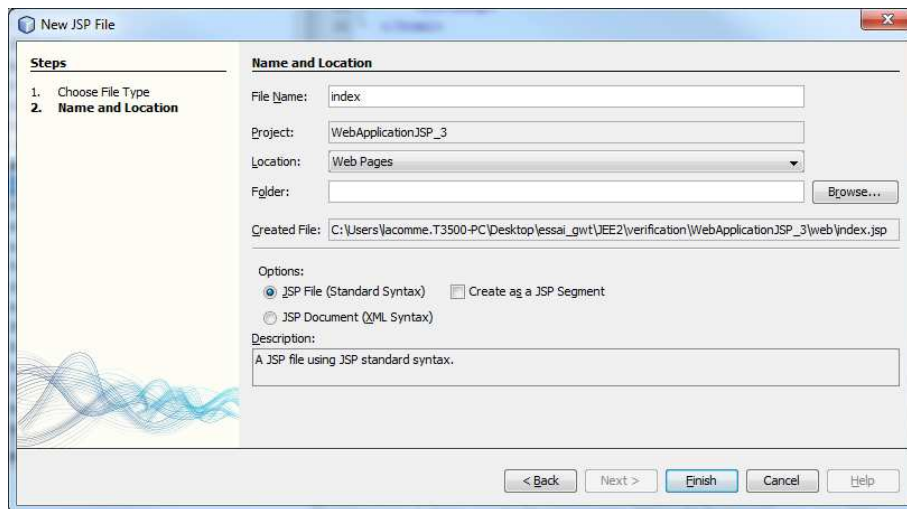
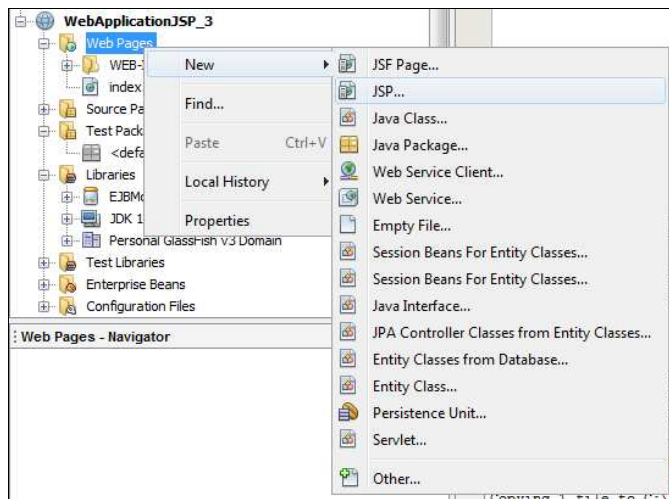
3.1. Remarquons le fichier **index.xml** qu'on peut visualiser simplement par un clic droit.



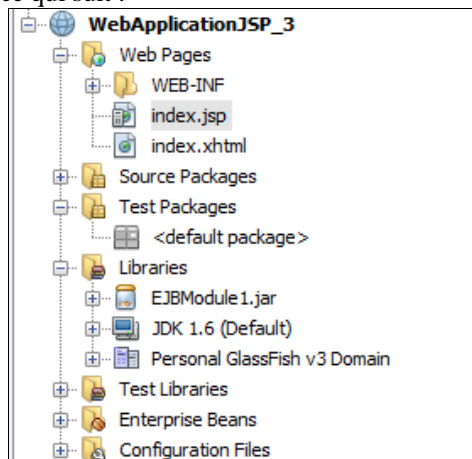
Ceci doit donner la visualisation suivante :



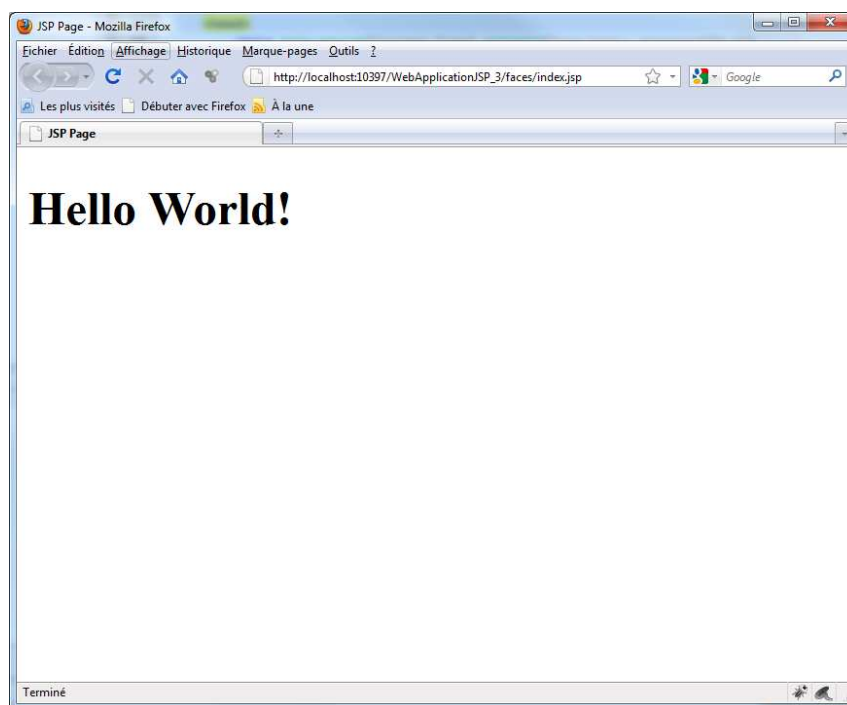
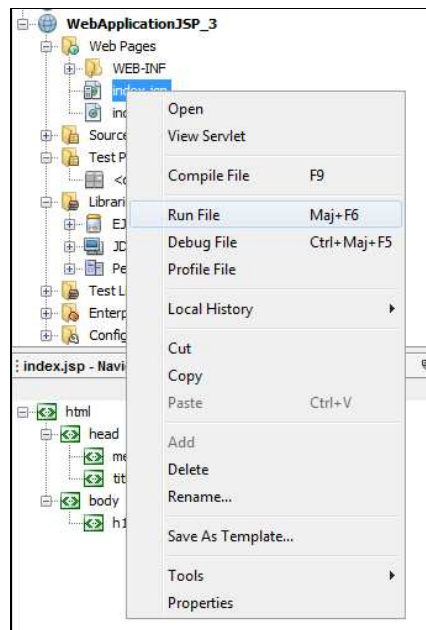
3.2. Ajout d'une page JSP



Le projet doit ressembler à ce qui suit :



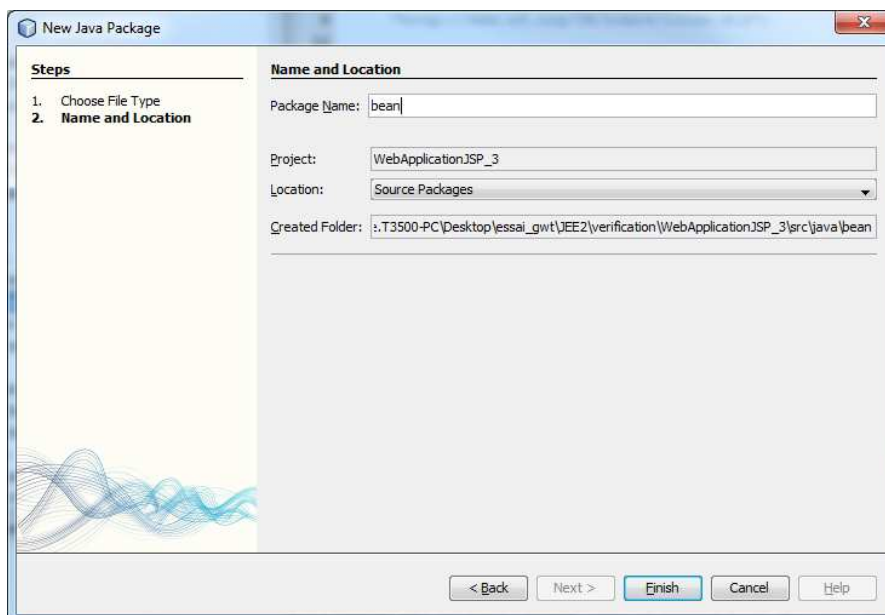
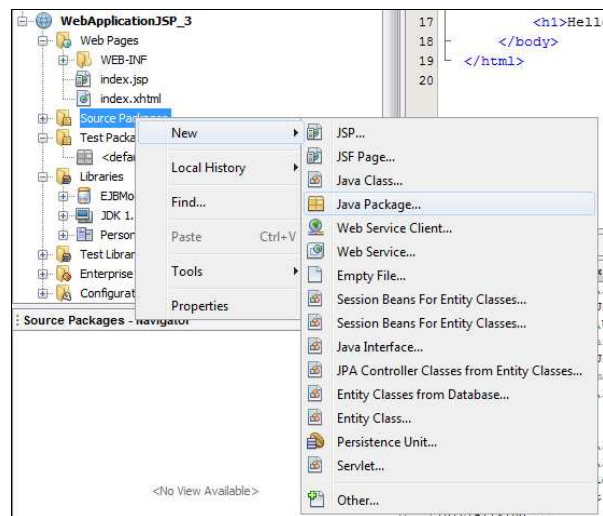
Il faut tester la page JSP en faisant un clic droit et Run File.



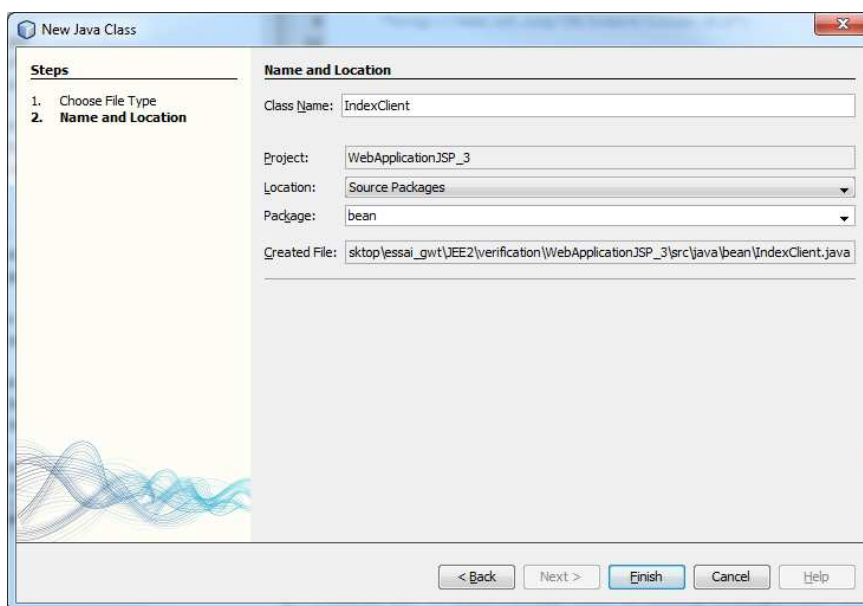
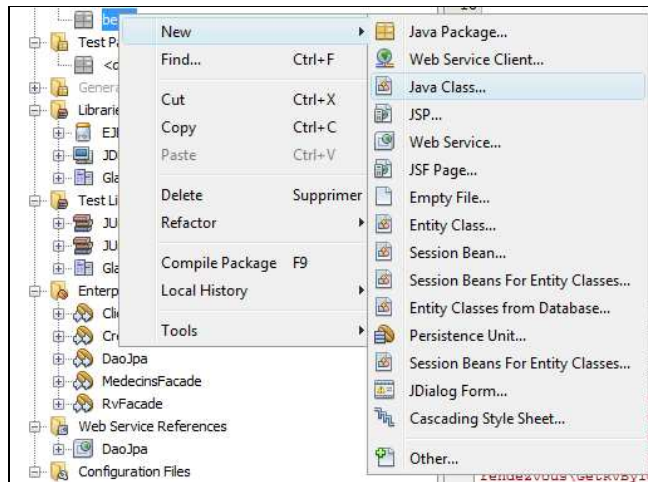
3.3. Création d'un package bean et d'une classe IndexClient

Nous allons créer un package nommé **Bean** dans lequel nous allons définir une classe nommée **IndexClient** qui va servir d'intermédiaire vers JPA.

Faire un clic droit sur **Source Package** et choisir **Java Package**.



Faire un clic droit sur **Source Package** sur bean et choisir Java Class



Ouvrir le fichier IndexClient.java et insérer le code suivant :

```

package bean;

import java.util.ArrayList;
import java.util.List;
import dao.DaoJpa;
import jpa.Clients;

public class IndexClient {

    private DaoJpa dao;

    private List<Clients> clients;

    public void setClients(List<Clients> clients) {
        this.clients = clients;
    }

    public List<Clients> getClients()
    {

        dao = new DaoJpa ();

        dao.init(); //On initialise la couche de persistance

        System.out.println("--- affichage liste des clients ---");
        List<Clients> myArr = new ArrayList<Clients>();
        myArr = dao.getAllClients();

        dao.close();
        return myArr;
    }
}

```

3.4. Modification de la page JSP

Modifiez la page comme suit :

```

<%@page import="bean.IndexClient,jpa.Clients, java.util.*" %>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>

<script language="javascript">
    function getClients()
    {
        <%
            IndexClient ind = new IndexClient();
            StringBuffer val = new StringBuffer();
            for (Clients c : ind.getClients()) {
                val.append(c.getNom()+" <br/> ");
            }
        %>
        var chaine = "<%= new String(val)%>";
        <!--var elt = document.getElementById("res");
        elt.value = chaine;-->
        document.getElementById("text").innerHTML= chaine;
    }
}

```

```

</script>
</head>
<body>
  <h1>Hello World!</h1>
  La date courante est : <%= new java.util.Date() %>

  <%
String[] langages = {"Java", "C++", "Smalltalk", "Simula 67"};
out.println("<h3>Principaux langages orientés objets : </h3>");
for (int i=0; i < langages.length; i++) {
  out.println("<p>" + langages[i] + "</p>");
}
%>

  <input type="submit" value="getClients" onclick="getClients()"/>

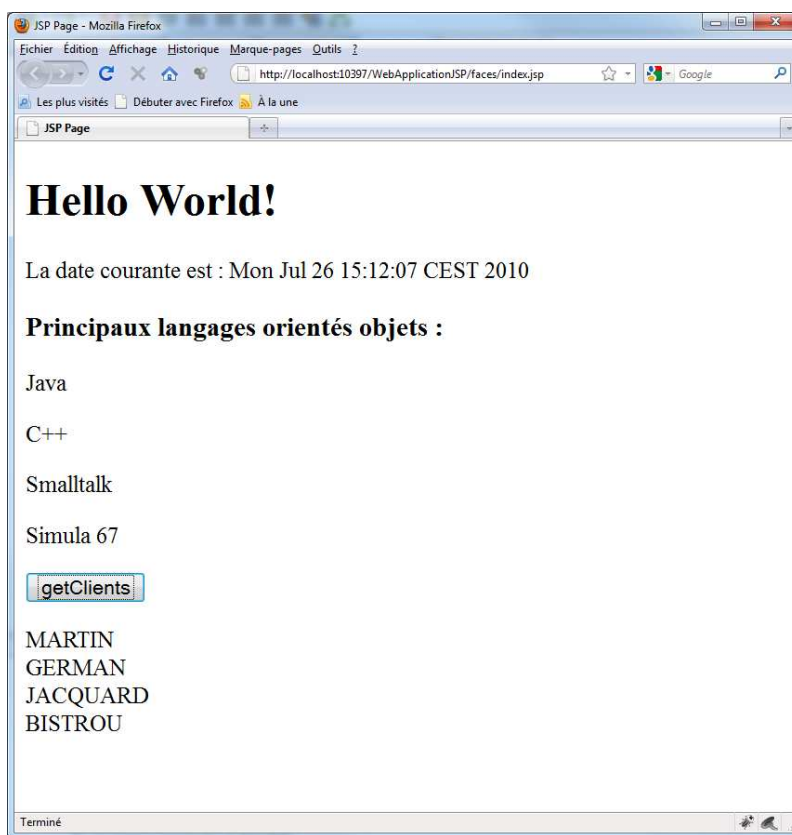
  <p id="text">

  </p>

</body>
</html>

```

Ce qui donne comme résultat d'exécution :



----- fin -----