Tutoriel JEE

Développement d'une application JEE selon le patron de conception MVC

Dr. Ramzi FARHAT

Objectif :

Créer une application JEE selon le modèle de conception MVC, tels que : (i) le Modèle sera développé sous forme d'EJBs, le Contrôleur sera développé sous forme d'une Servlet et la Vue sera développée sous forme d'une page JSP.

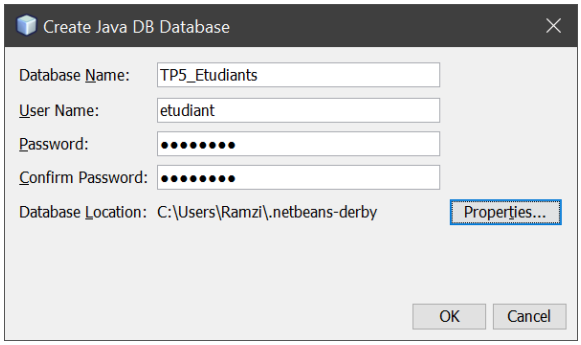
Nous allons utiliser **NetBeans 8.1** entant que EDI, **GlassFish 4.1** entant que serveur d'application JEE et **Java DB** (Derby) entant que serveur de bases de données relationnelles.

Remarque :

Il faut installer une autre version de GlassFish que celle installée par défaut (4.1.1) avec NetBeans 8.1 qui comporte des bugs au niveau de la console d'administration.

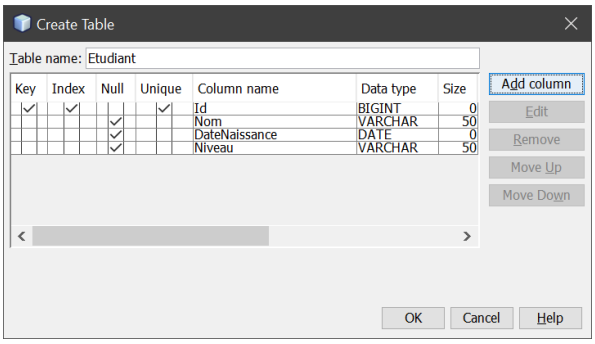
Création de la base de données

Pour créer une base de données, dans l'anglet **Services** dans NetBeans choisissez **Java DB** sous **Databases**. Puis, dans le menu contextuel choisissez **Create Database** tout en en spécifiant le nom (*TP5_Etudiants*) et les paramètres de connexion.



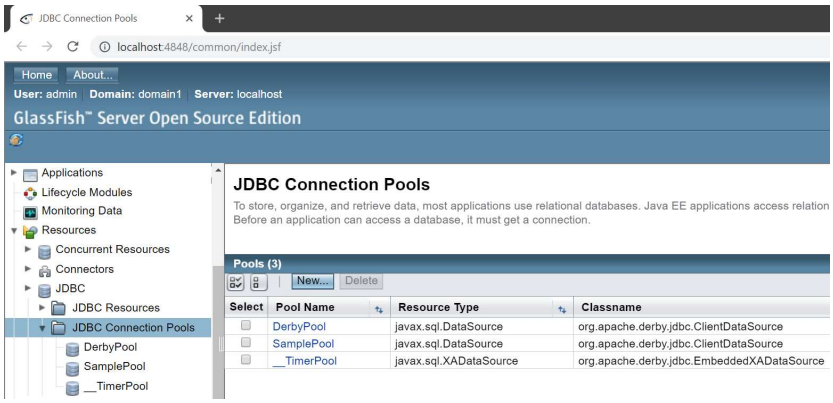
Dans cette base de données créez la table *Etudiant* ayant le schéma suivant :

Etudiant (Id : BigInt, Nom : Varchar(50), DateNaissance : Date, Niveau : Varchar(50))

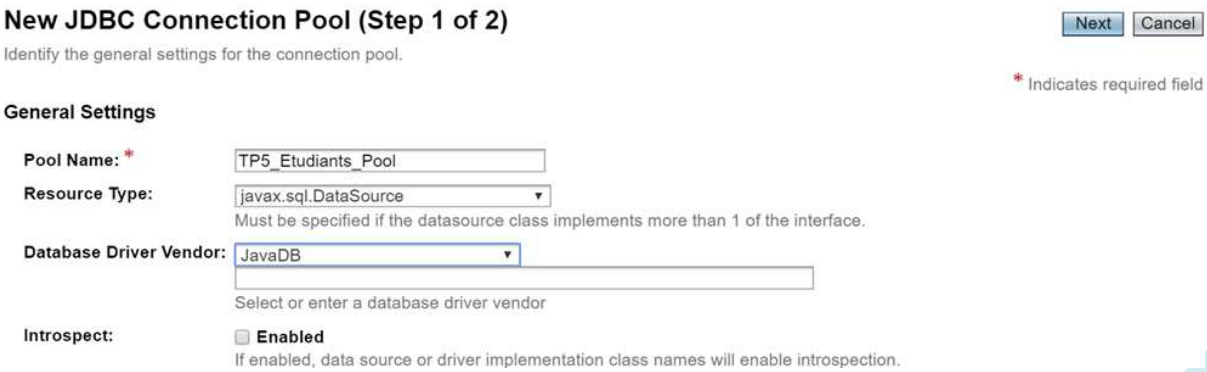


Maintenant, nous allons créer un **pool** de connexion à la base de données au niveau du serveur **GlassFish** pour faire abstraction du SGBD au niveau du code source et surtout au niveau du fichier de configuration *persistence.xml*. De plus, l'usage d'un pool de connexion permet d'améliorer les performances de l'application. A ce niveau il faut vérifier que vous avez une version sans bugs de **GlassFish** (par exemple : la version 4.1), sinon téléchargez une nouvelle version et ajoutez là à la liste des serveurs dans l'anglet **Services** de NetBeans.

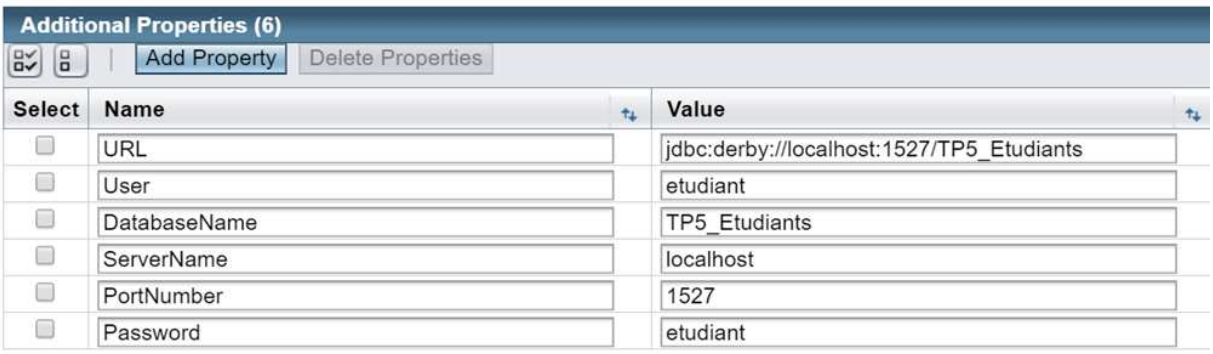
Démarrez le serveur via l'action **start** dans le menu contextuel, puis lancez la console d'administration via l'action **View Domain Admin Console** dans le menu contextuel du serveur. Puis dans le menu d'administration choisissez **JDBC Connection Pool** :



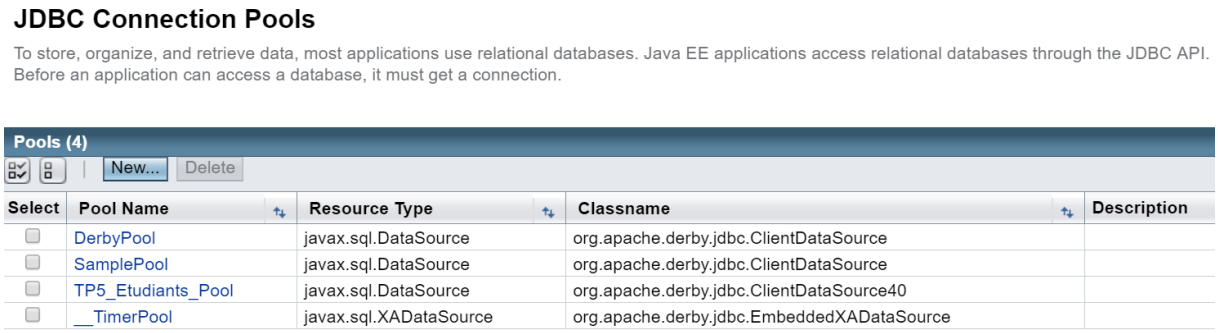
Cliquez sur **new** et créez un **pool** de connexion nommé *TP5_Etudiants_Pool* qui est une ressource de type **javax.sql.DataSource** et comme base de données **JavaDB** :



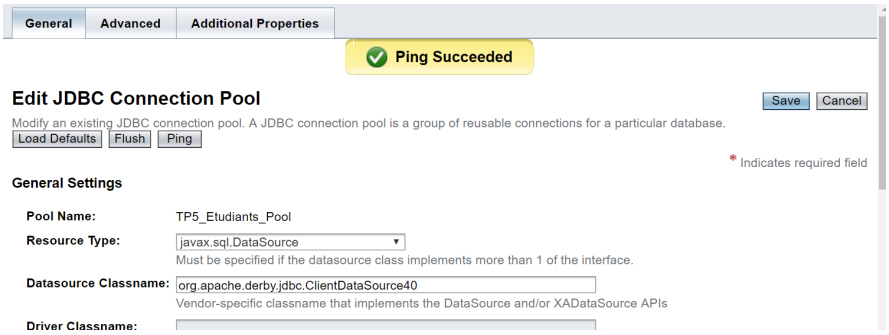
Puis, choisissez **next** et spécifiez les propriétés suivantes tout en **supprimant** les autres propriétés :



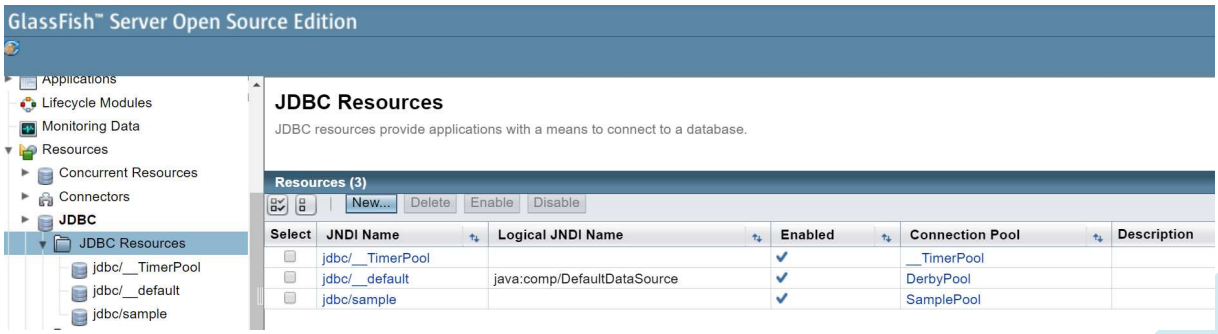
Vous devez voir votre **pool** apparaître dans la liste des pools de connexion gérés par le serveur :



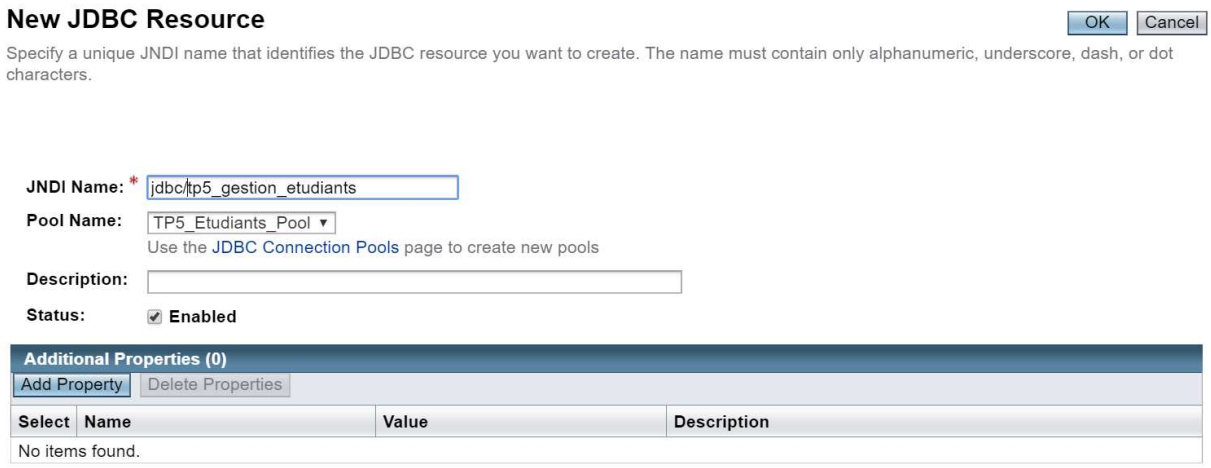
Maintenant il faut tester votre **pool** en cliquant sur son nom et en choisissant **ping** :



L'étape suivante est de créer une source JDBC qui va être référencée dans votre projet et qui est associée au pool que vous venez de créer :



Il suffit maintenant de donner un nom **JNDI** à votre ressource (à utiliser dans votre projet pour faire référence à cette source) et de lui associer au pool que vous venez de créer :

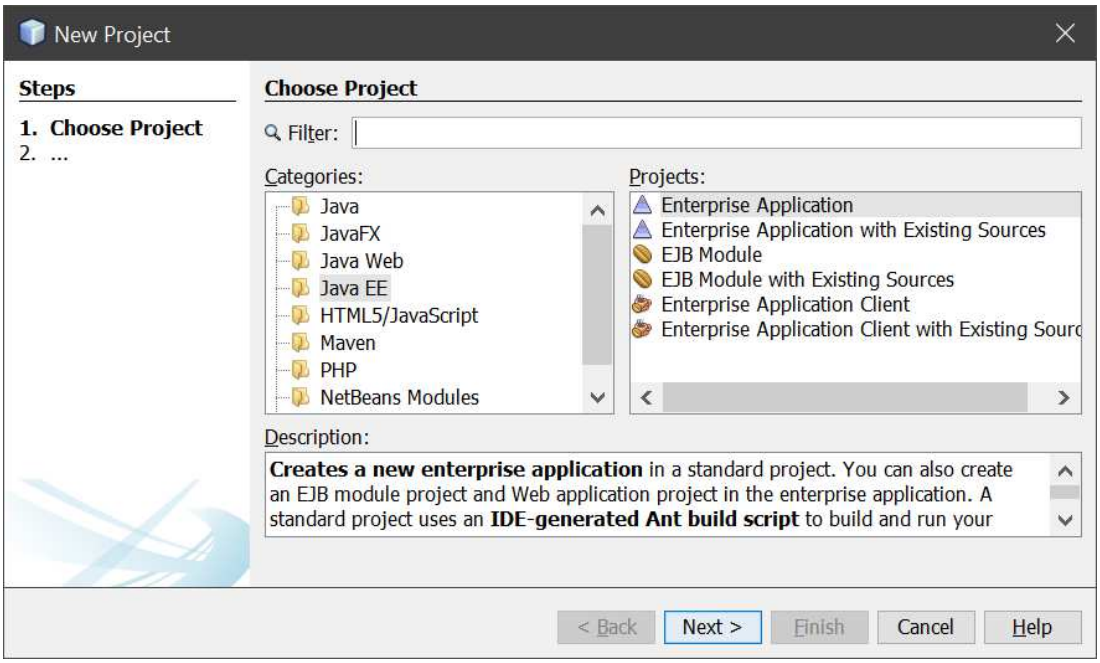


Vous avez terminé votre travail d'administrateur et vous allez entamer votre travail de programmation !

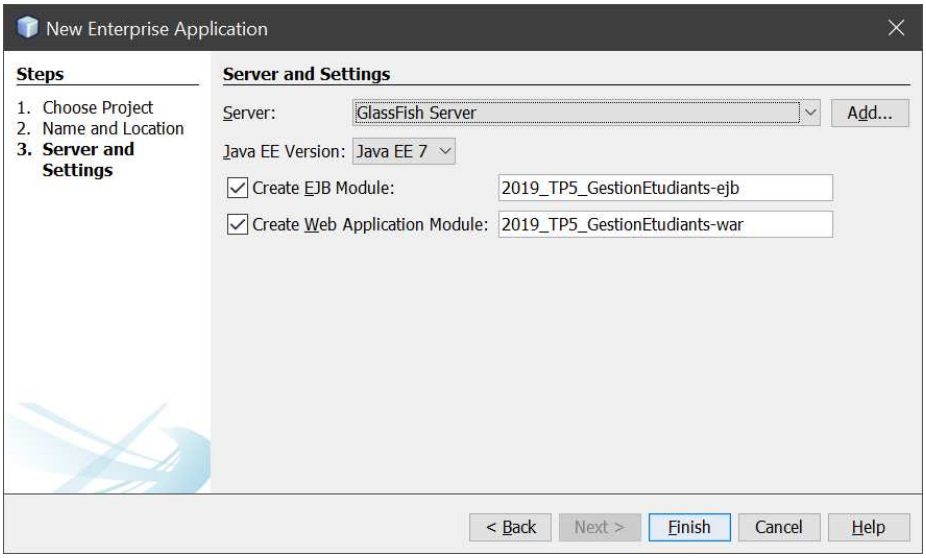
Création du projet

Il s'agit d'un projet JEE qui comporte deux sous projet : (1) un projet EJB qui comporte les éléments qui vont résider dans le conteneur EJB et (2) un projet Web qui comporte les éléments qui vont résider dans le conteneur Web.

Donc dans « New Project » choisissez un projet « Entreprise Application » dans la catégorie « Java EE »



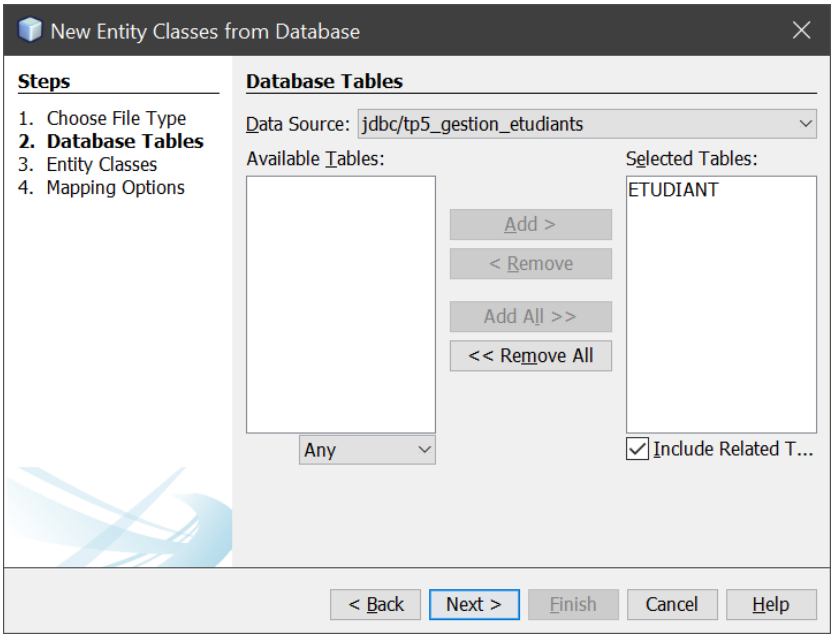
Nommez-le par exemple *2019_TP5_GestionEtudiants* et surtout choisissiez le bon serveur **GlassFish** (qui contient votre ressource JDBC), tout en choisissant la création des deux modules EJB et Web.

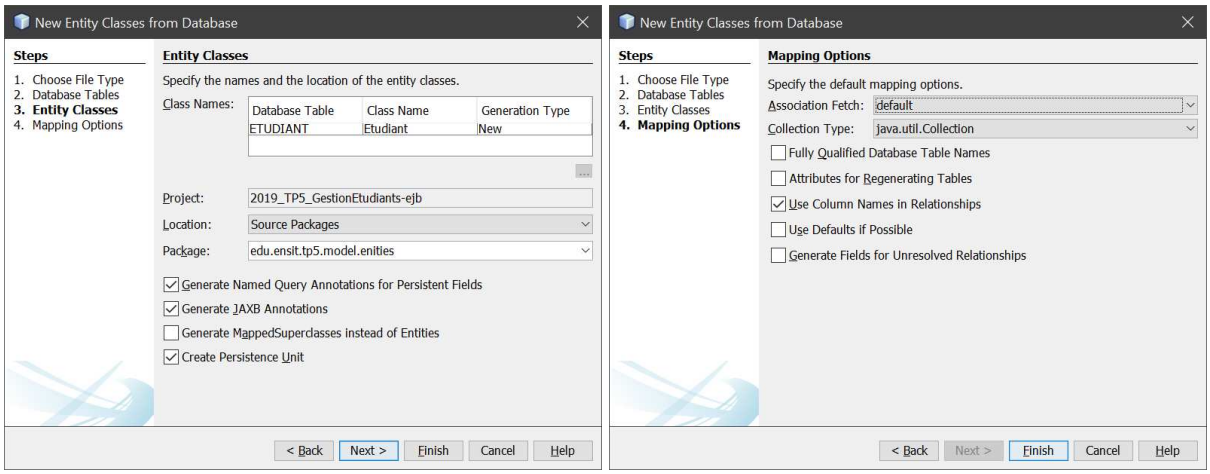


Module EJB

Dans l'anglet « **Projects** » vous allez avoir trois projets : un projet principal JEE, un projet EJB pour le module EJB et un projet Web pour le module Web. Nous nous intéressant au projet qui concerne le module EJB : *2019_TP5_GestionEtudiants-ejb*.

Créez un **EJB Entity** à partir de la base de données (pour avoir la génération automatique du code source) dans le package *edu.ensit.tp5.modele.entities*. Dans le choix de la data source, spécifiez le nom **JNDI** défini dans le serveur pour votre source de données :





Maintenant, il faut faire quelques modifications dans l'**EJB Entity** permettant de répondre à nos besoins. Parmi, c'est le fait de restreindre les valeurs qu'on peut sauvegarder dans la colonne « Niveau ». Alors il faut créer une énumération dans le package *edu.ensit.tp5.modele.entities* comme suit :

```
package edu.ensit.tp5.modele.entities;
public enum NiveauEtude {
    GInfo_1, GInfo_2, GInfo_3;
}
```

Maintenant, vous devez ouvrir le code source de votre **EJB Entity** *Etudiant.java* pour apporter les modifications suivantes : **(1)** Ajouter à la variable « *id* » l'annotation permettant de le générer automatiquement, **(2)** changer le type de la variable « *niveau* » de *String* à *NiveauEtude* tout en ajoutant l'annotation appropriée relatives aux valeurs de type énumération et **(3)** finalement mettre à jour le code des mutateurs et accesseurs de la variable *niveau* pour s'aligner au changement de type :

```
43      @Id
44      @Basic(optional = false)
45      @NotNull
46      @Column(name = "ID")
47      @GeneratedValue(strategy = GenerationType.AUTO)
48      private Long id;
```

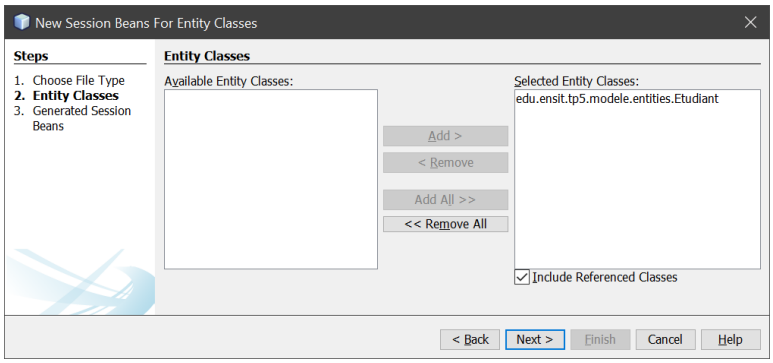
```
52      @Size(max = 50)
53      @Column(name = "NIVEAU")
54      @Enumerated(EnumType.STRING)
55      private NiveauEtude niveau;
```

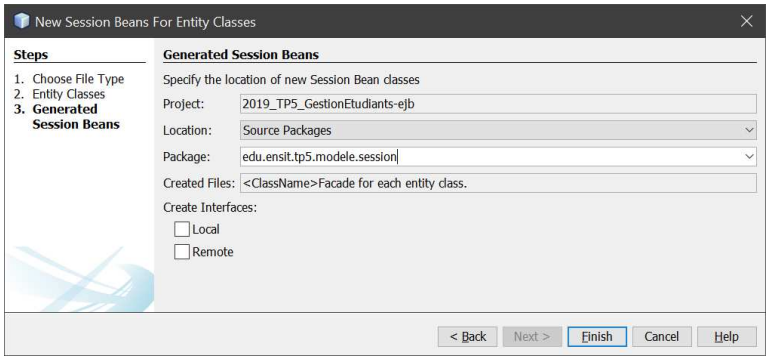
```
88      public NiveauEtude getNiveau() {
89          return niveau;
90      }
91
92      public void setNiveau(NiveauEtude niveau) {
93          this.niveau = niveau;
94      }
```

Maintenant, nous avons utilisé la spécification **JPA** pour faire la correspondance Objet/Relationnel et dorénavant nous allons agir sur l'**EJB Entity** au lieu d'agir directement sur notre table relationnelle.

Justement, pour pouvoir agir (c'est-à-dire appliquer les opération CRUD) nous avons besoin d'un **EJB Session Stateless** chargé d'offrir ces services. Donc dans le même projet (*2019_TP5_GestionEtudiants-ejb*) nous allons créer notre EJB.

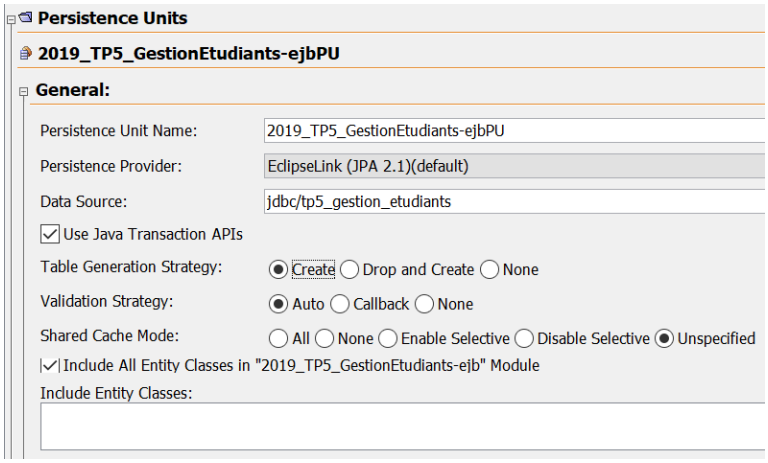
Choisissez dans « **new file** » l'entrée « **New Session Beans For Entity classes** » et choisissez l'entité **Etudiant**. Le code doit résider dans le package « *edu.ensit.tp5.session* » :





Ceci va permettre de générer les classes ***AbstractFacade.java*** et sa fille ***EtudiantFacade.java*** offrant toutes les opérations nécessaires dans notre application pour gérer l'EJB Entity *Etudiant.java* et derrière la table correspondante *Etudiant*.

Dernière chose pour le bon fonctionnement du projet, il va falloir modifier le fichier ***persistence.xml*** de façon à permettre la création des tables comme le montre la capture d'écran ci-dessous. En fait, pour générer automatiquement la valeur de la clé primaire de la table étudiant, le « **Persistence Provider** » a besoin de créer une table dans la base de données nommée *SEQUENCE*. Donc il faut lui permettre de créer des tables lors du premier déploiement sur le serveur. Donc choisissez **Create** pour **Table Generation Starategy**. Une fois la table créée vous pouvez la remettre à **None** (ce qui est recommandé).

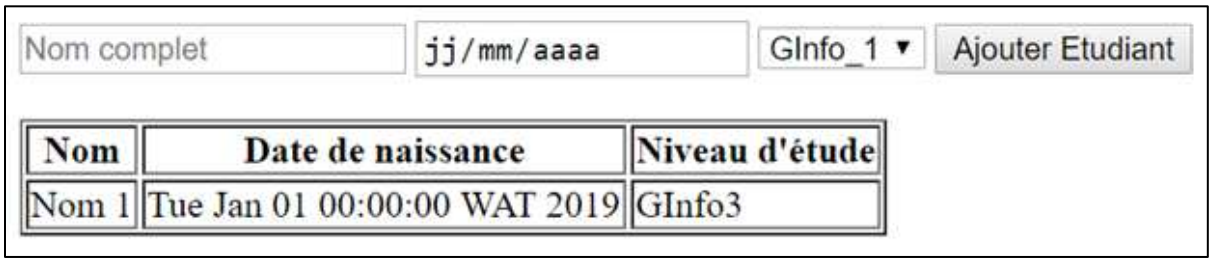


Maintenant vous pouvez, dans le projet principal « *2019_TP5_GestionEtudiants* » lancer **Clean and Build** puis **Run** pour s'assurer que tout va bien et qu'il n'y a pas d'erreurs. De plus, vous devez vérifier la présence de la table *SEQUENCE* dans votre base de données.

Notre modèle est près, il reste à développer le module Web permettant à l'utilisateur d'interagir avec l'application pour gérer les étudiants.

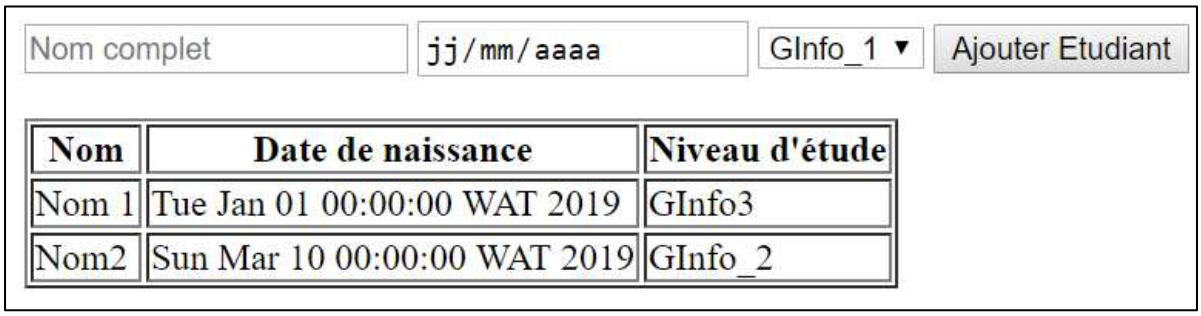
Module Web

L'objectif est de créer une interface Web qui contient en haut un menu permettant d'ajouter un étudiant et en dessous une liste des étudiants présents dans la base de données :



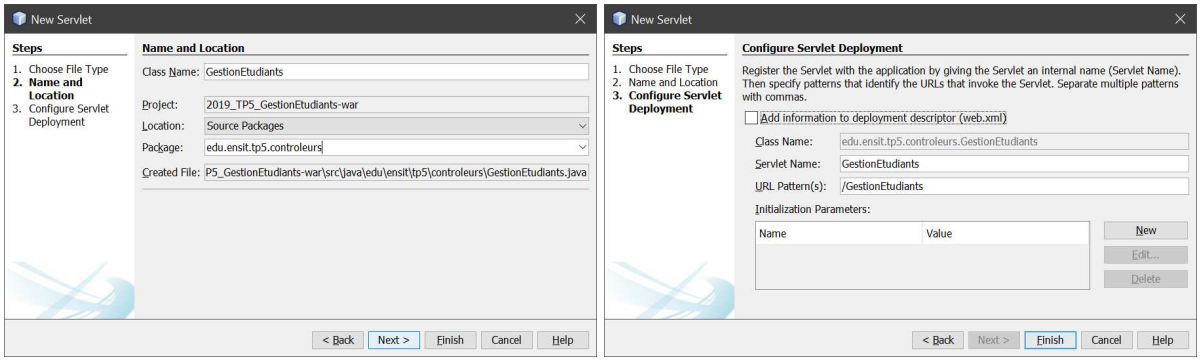
A chaque ajout d'un étudiant s'il y a des erreurs de saisi alors l'interface va apparaitre avec les erreurs affichées en dessus du formulaire d'ajout :

S'il n'y a pas des erreurs donc l'étudiant est ajouté à la base de données et ses données apparaissent dans le tableau des étudiants :



Pour commencer, conformément au patron de conception MVC2, nous devons avoir un contrôleur qui va se charger de gérer toutes les requêtes http provenant à notre application Web.

Créez dans le projet *2019_TP5_GestionEtudiants-war* une servlet ***GestionEtudiants.java*** :



Nettoyez le code source pour ne garder que les méthodes *doGet()* et *doPost()* :

```
1  package edu.ensit.tp5.controleurs;
2
3  import java.io.IOException;
4  import javax.servlet.ServletException;
5  import javax.servlet.annotation.WebServlet;
6  import javax.servlet.http.HttpServlet;
7  import javax.servlet.http.HttpServletRequest;
8  import javax.servlet.http.HttpServletResponse;
9
10 @WebServlet(name = "GestionEtudiants", urlPatterns = {"/GestionEtudiants"})
11 public class GestionEtudiants extends HttpServlet {
12
13     @Override
       protected void doGet(HttpServletRequest request, HttpServletResponse response)
15             throws ServletException, IOException {
16     }
17
18     @Override
       protected void doPost(HttpServletRequest request, HttpServletResponse response)
20             throws ServletException, IOException {
21     }
22 }
```

Lors de l'appel de la servlet via la méthode GET (via un lien hypertexte) nous allons : **(1)** ajouter un attribut de requête ***listeEtudiants*** contenant la liste de tous les étudiants de la table Etudiant, puis **(2)** afficher une vue nommée ***gestionEtudiants.jsp*** sous ***WEB-INF***. Donc nous avons besoin de l'EJB session ***EtudiantFacade*** qui offre la méthode ***findAll()*** permettant de récupérer cette liste. L'instance de l'EJB ***EtudiantFacade*** est récupérée via une injection de dépendance :

```
16  @WebServlet(name = "GestionEtudiants", urlPatterns = {"/GestionEtudiants"})
17  public class GestionEtudiants extends HttpServlet {
18
19      // Récupérer l'EJB Session EtudiantFacade
20      @EJB
21      EtudiantFacade etudiantFacade;
22
23      @Override
        protected void doGet(HttpServletRequest request, HttpServletResponse response)
25              throws ServletException, IOException {
26          // Envoyer la liste des étudiants
27          request.setAttribute("listeEtudiants", etudiantFacade.findAll());
28          // Appler la vue WEB-INF/gestionEtudiants.jsp
29          request.getRequestDispatcher("WEB-INF/gestionEtudiants.jsp").forward(request, response);
30
31      }
```

Maintenant, noua allons éditer la page d'accueil de l'aplication Web ***index.html*** pour ajouter un lien hypertexte qui permet d'appeler le contrôleur :

```
7       </head>
8       <body>
9           <a href="GestionEtudiants">Gestion Etudiants</a>
10      </body>
```

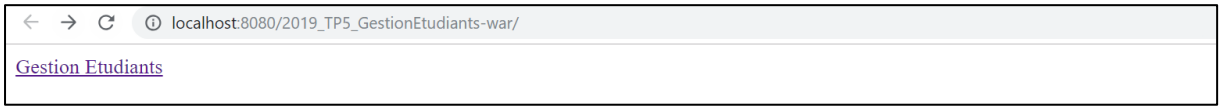
Ensuite, il faut créer la page ***gestionEtudiant.jsp*** sous le répertoire ***WEB-INF***. Elle va contenir un tableau permettant d'afficher tous les étudiants en utilisant l'attribut ***listeEtudiant*** (via l'**EL**).

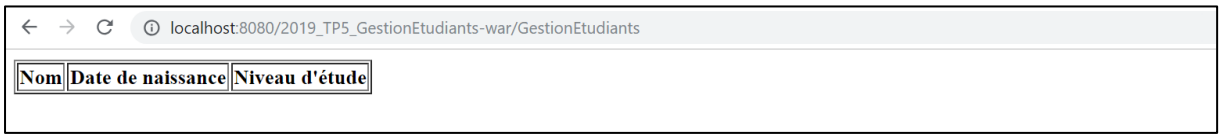
Une boucle permet d'afficher pour chaque élément dans ***listeEtudiant*** une ligne dans le tableau (via la bibliothèque **JSTL core**) :

```
1   <%@page contentType="text/html" pageEncoding="UTF-8"%>
2   <%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
3   <!DOCTYPE html>
4   <html>
5       <head>
6           <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7           <title>Gestion Etudiants</title>
8       </head>
9       <body>
10          <table border="1">
11              <tr>
12                  <th>Nom</th>
13                  <th>Date de naissance</th>
14                  <th>Niveau d'étude</th>
15              </tr>
```

```
16              <c:forEach var="etudiant" items="${listeEtudiants}">
17              <tr>
18                  <td>${etudiant.nom}</td>
19                  <td>${etudiant.datenaissance}</td>
20                  <td>${etudiant.niveau}</td>
21              </tr>
22              </c:forEach>
23          </table>
24
25      </body>
26  </html>
```

Maintenant pour tester, je vous recommande d'aller au projet principal ***2019_TP5_GestionEtudiants*** pour lancer **Clean and Build** puis **Run**.





Revenez au code source de la page ***gestionEtudiant.jsp*** pour ajouter avant le tableau un formulaire d'ajout d'un étudiant tel que les valeurs de la liste déroulante **niveau** sont générés via un attribut de requête ***niveauxEtude*** :

```
    <form action="GestionEtudiants" method="POST">
        <input type="text" name="nom" placeholder="Nom complet"/>
        <input type="date" name="dateNaissance"/>
        <select name="niveauEtude">
            <c:forEach var="niveau" items="${niveauxEtude}">
                <option value="${niveau}">${niveau}</option>
            </c:forEach>
        </select>
        <input type="submit" value="Ajouter Etudiant"/>
    </form>
    <br>
```

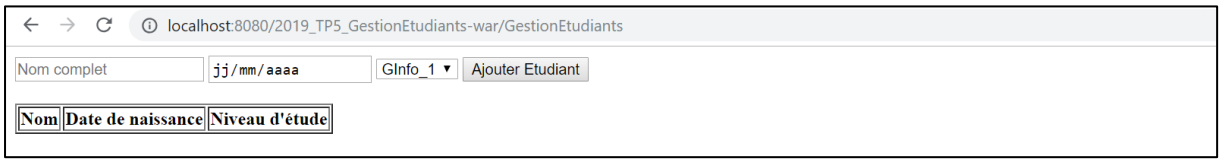
Pour fournir la liste des niveaux d'étude autorisés il va falloir, dans un premier temps, ajouter une méthode statique dans l'énumértion ***NiveauEtude.java*** permettant de retourner une liste des valeurs des instances de l'énumération :

```
9       public static List<String> getNiveauxEtude(){
10          List<String> l = new LinkedList<String>();
11          for(NiveauEtude niveau : NiveauEtude.values()){
12              l.add(niveau.toString());
13          }
14          return l;
15      }
```

Il suffit maintenant d'ajouter cette liste de niveaux d'étude comme attribut de requête dans la méthode ***doGet()*** du contrôleur pour l'utiliser dans la vue :

```
16  @WebServlet(name = "GestionEtudiants", urlPatterns = {"/GestionEtudiants"})
17  public class GestionEtudiants extends HttpServlet {
18
19      // Récupérer l'EJB Session EtudiantFacade
20      @EJB
21      EtudiantFacade etudiantFacade;
22
23      @Override
        protected void doGet(HttpServletRequest request, HttpServletResponse response)
25              throws ServletException, IOException {
26          // Envoyer la liste des étudiants
27          request.setAttribute("listeEtudiants", etudiantFacade.findAll());
28          request.setAttribute("niveauxEtude", NiveauEtude.getNiveauxEtude());
29          // Appler la vue WEB-INF/gestionEtudiants.jsp
30          request.getRequestDispatcher("WEB-INF/gestionEtudiants.jsp").forward(request, response);
31
32      }
```

Encore, un **Clean and Build** et un **Run** sur le projet principal, puis on teste :



Maintenant il faut traiter le code qui va permettre d'ajouter un étudiant à la base de données l'orsqu'on clique sur le bouton « **Ajouter Etudiant** ». En fait, il s'agit d'une requête POST donc il va falloir faire le traitement dans la méthode **doPost()** du contrôleur. Toutefois, pour laisser le code du contrôleur assez lisible et pour écrire un code modulaire, nous allons créer une classe spécialisée dans le traitement du formulaire d'ajout d'étudiants permettant de **(1)** vérifier les données du formulaire et de les convertir et de **(2)** préparer les éventuels messages d'erreurs.

Créez dans le projet ***2019_TP5_GestionEtudiants-war*** la classe ***FormulaireAjoutEtudiant.java*** dans le package ***edu.ensit.tp5.formulaires***. Cette classe contient des variables qui correspondent aux trois champs du formulaire : ***nom*** de type ***String***, ***dateNaissance*** de type ***Date*** et ***niveauEtude*** de type ***NiveauEtude***. En plus, une quatrièmre variable ***erreurs*** de tyepe ***Map<String, String>*** va permettre d'associer le nom du paramètre qui comporte une erreur à un message d'erreur. Il faut générer les accesseurs et les mutateurs pour ces variables. Ensuite, pour lancer la conversion et la collecte des éventuelles erreurs on va passer par un constructeur qui reçoit les différents paramètres sous forme de chaînes de caractères puis il va appeler les mutateurs des variables pour faire la conversion. Chaque mutateur va déclencher une exception utilisateur en ca de problème. Donc il faut créer cette

exception utilisateur, par exemple dans le même fichier source, on peut créer la classe ***EtudiantException*** :

```
63    class EtudiantException extends Exception{
64        EtudiantException(String msg){
65            super(msg);
66        }
67    }
```

Pour aller en douceur nous allons juste commencer par la variable ***nom***. Il faut donc modifier le mutateur ***setNom()*** afin de déclarer qu'il reçoit un paramètre de type ***String*** et qu'il peut déclencher l'exception ***EtudiantException***. Dans le corps de la méthode nous allons déclencher l'exception avec un message d'erreur si le paramètre est null ou vide.

Puis dans le constructeur nous allons appeler ce mutateur pour affecter une valeur à la variable ***nom***. En cas d'exception (*nom* est un objet *null* ou c'est *une chaîne vide*) nous allons ajouter une entrée au Map ***erreurs*** avec comme ***clé*** « *nom* » et comme ***valeur*** le message d'erreur retourné par l'exception.

```
14    public class FormulaireAjoutEtudiant {
15        private String nom;
16        private Date dateNaissance;
17        private NiveauEtude niveauEtude;
          private Map<String, String> erreurs = new HashMap<String, String>();
19
20        public FormulaireAjoutEtudiant(String nom, String dateNaissance, String niveauEtude){
21            try{
                  this.setNom(nom);
23            }catch(EtudiantException e){
24                erreurs.put("nom", e.getMessage());
25            }
26        }
27        // Accesseurs et mutateurs
28        public String getNom() {
29            return nom;
30        }
31        public void setNom(String nom) throws EtudiantException{
32            if(nom == null || nom.trim().length()==0)
33                throw new EtudiantException("Nom complet doit être renseigné");
34            this.nom = nom.trim();
35        }
36
37        public Date getDateNaissance() { ... 3 lines }
```

Maintenant la méthode ***doPost()*** est plus simple à développer. Il suffit de créer une instance de la classe ***FormulaireAjoutEtudiant*** en passant au constructeur les valeurs récupéré comme paramètre de la requête (qui sont envoyés par le formulaire d'ajout d'un étudiant dans la vue JSP).

L'exécution du constructeur va permettre pour le moment d'initialiser le ***nom*** dans l'objet ***formulaireAjoutEtudiant*** s'il n'est pas vide, sinon de générer une erreur dans le Map ***erreurs***. Donc si le Map ***erreurs*** est vide on peut se permettre de créer un objet ***Etudiant*** et de l'initialiser

avec les valeurs des variables de l'objet ***formulaireAjoutEtudiant*** (il faut noter que puisqu'on a travaillé uniquement sur la variable nom dans FormulaireAjoutEtudiant, alors on va donner des constantes pour initialiser la *dateDeNaissance* et le *niveau* de l'objet **Etudiant**. On a pu les initialiser à null puisque dans la table **Etudiant** on a permis les valeurs null pour toutes les colonnes). Dans tous les cas on va envoyer trois attributs dans la requête à la vue ***gestionEtudiant.jsp*** : **(1)** l'objet ***formulaireAjoutEtudiant*** qui contient les éventuels erreurs, **(2)** la liste des étudiants pour afficher le tableau des étudiants et **(3)** les niveaux d'étude pour pouvoir construire la liste déroulante des niveaux d'étude.

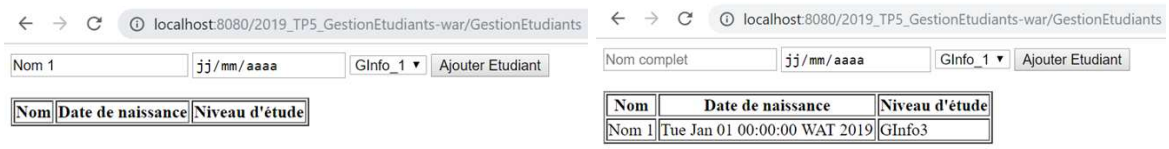
```
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
36          throws ServletException, IOException {
37          // On va déléger la vérification des erreurs à un objet de type FormulaireAjoutEtudiant
38          FormulaireAjoutEtudiant formulaireAjoutEtudiant = new FormulaireAjoutEtudiant(
39                  request.getParameter("nom"),
40                  request.getParameter("dateNaissance"),
41                  request.getParameter("niveauEtude"));
42          if (formulaireAjoutEtudiant.getErreurs().isEmpty()) {
43              Etudiant etudiant = new Etudiant();
44              etudiant.setNom(formulaireAjoutEtudiant.getNom());
45              etudiant.setDatenaissance(new Date("01/01/2019"));
46              etudiant.setNiveau(NiveauEtude.GInfo3);
47              etudiantFacade.create(etudiant);
48          }
49          request.setAttribute("formulaireAjoutEtudiant", formulaireAjoutEtudiant);
50          // Envoyer la liste des étudiants
51          request.setAttribute("listeEtudiants", etudiantFacade.findAll());
52          // Envoyer la liste des niveaux d'étude
53          request.setAttribute("niveauxEtude", NiveauEtude.getNiveauxEtude());
54          // Appeler la vue
55          request.getRequestDispatcher("WEB-INF/gestionEtudiants.jsp").forward(request, response);
56
57      }
```

Maintenant, il ne reste qu'à améliorer la **vue** afin d'ajouter le code permettant d'afficher les éventuelles erreurs juste au dessus du formulaire d'ajout d'un étudiant :

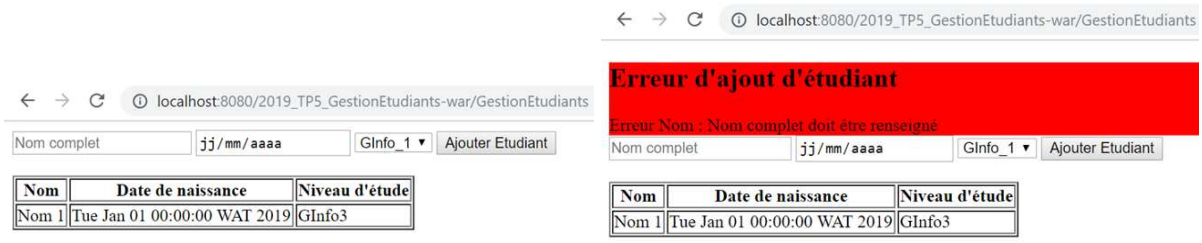
```
 9      <body>
10          <div style="background-color: red;">
11              <c:if test="${!empty formulaireAjoutEtudiant.erreurs}">
12                  <h2>Erreur d'ajout d'étudiant</h2>
13              </c:if>
14              <c:if test="${!empty formulaireAjoutEtudiant.erreurs['nom']}">
15                  Erreur Nom : ${formulaireAjoutEtudiant.erreurs['nom']}<BR>
16              </c:if>
17          </div>
```

Il est temps de tester (après un « **Clean and Build**» et un « **Run** » sur le projet principal).

Premier test sans erreurs :



Deuxième test avec un champs **Nom** vide :



Maintenant que tout fonctionne à merveille il suffit de généraliser le travail de conversion et de gestion des erreurs dans la classe formulaireAjoutEtudiant.java :

```
45      public void setDateNaissance(String dateNaissance)throws EtudiantException {
46          try {
47              // Il faut considérer le format retourné par le champs date dans l'HTML 5 : yyyy-MM-dd
48              this.dateNaissance = new SimpleDateFormat("yyyy-MM-dd").parse(dateNaissance);
49
50          } catch (ParseException ex) {
51              throw new EtudiantException("Date de naissance doit être renseignée sous forme jj/MM/aaaa");
52          }
53      }
54      public NiveauEtude getNiveauEtude() {...3 lines }
57      public void setNiveauEtude(String niveauEtude) {
58          this.niveauEtude = NiveauEtude.valueOf(niveauEtude);
59      }
```

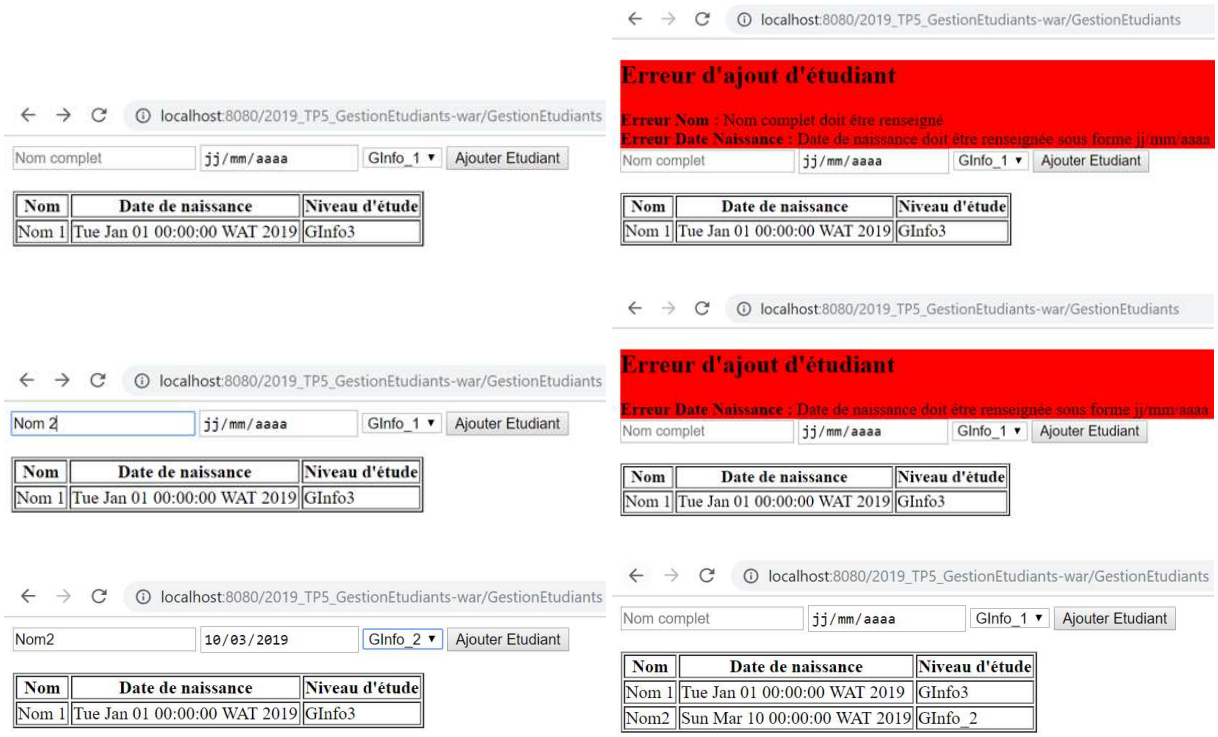
Puis, il suffit de mettre à niveau le code d'initialisation d'un objet Etudiant dans la méthode ***doPost()*** du contrôleur :

```
42              if (formulaireAjoutEtudiant.getErreurs().isEmpty()) {
43                  Etudiant etudiant = new Etudiant();
44                  etudiant.setNom(formulaireAjoutEtudiant.getNom());
45                  etudiant.setDatenaissance(formulaireAjoutEtudiant.getDateNaissance());
46                  etudiant.setNiveau(formulaireAjoutEtudiant.getNiveauEtude());
47                  etudiantFacade.create(etudiant);
48              }
```

Finalement, il va falloir prendre en compte les erreurs du paramètre date dans la vue :

```
10          <div style="background-color: red;">
11              <c:if test="${!empty formulaireAjoutEtudiant.erreurs}">
12                  <h2>Erreur d'ajout d'étudiant</h2>
13              </c:if>
14              <c:if test="${!empty formulaireAjoutEtudiant.erreurs['nom']}">
15                  <b>Erreur Nom :</b> ${formulaireAjoutEtudiant.erreurs['nom']}<BR>
16              </c:if>
17              <c:if test="${!empty formulaireAjoutEtudiant.erreurs['dateNaissance']}">
18                  <b>Erreur Date Naissance :</b> ${formulaireAjoutEtudiant.erreurs['dateNaissance']}<BR>
19              </c:if>
20          </div>
```

Puis il ne reste qu'à tester le code (après un « **Clean and Build**» et un « **Run** » sur le projet principal) :



Perspectives

Si vous voulez allez plus loin et mettre en pratique vos compétences en programmation JEE, vous pouvez améliorer l'application en ajoutant une quatrième colonne dans le tableau d'affichage des étudiants. Cette colonne doit comporter deux boutons : édition et suppression. L'édition va permettre d'afficher un formulaire avec les valeurs qui correspondent à la ligne courante avec la possibilité de les modifier et de mettre à jour l'enregistrement dans la table Etudiant (Une autre alternative, peut-être de montrer chaque ligne sous forme de champs éditables avec un bouton de mise-à-jour). La suppression permet de supprimer l'enregistrement qui correspond à la ligne en cours.

On peut également apporter plusieurs améliorations sur le code source et sur la structure du projet. Ceci n'a pas été fait pour des soucis pédagogiques.

Bon travail