

Rapport Projet 2

Introduction

Nous sommes satisfaits de pouvoir vous rendre ce projet puisqu'il représente la preuve que nous avons assimilé les concepts vus en cours et mis en pratique dans notre web application.

Tout ce qui était demandé dans le cahier des charges est présent dans notre projet, à savoir :

- Afficher la liste des élèves de la base SQL
- Ajouter un nouvel élève
- Modifier un élève
- Supprimer un élève
- Vérifier les inputs d'un utilisateur lors de la création ou modification d'un élève
- Faire fonctionner notre application soit en JDBC soit en JPA
- Déployer notre application

Pour gérer les 2 versions différentes JDBC et JPA nous avons créé **deux paquets** dans « src », un paquet qui gèrera JDBC et l'autre JPA. Nous allons d'abord voir le fonctionnement de celui qui fonctionne en JDBC.

WebStudentBook JSF/JDBC

Database

La base de données MySQL

Nous avons décidé d'utiliser la même base de données « studentdb » que dans les TD précédents.

Un élève est donc représenté par son :

- **ID** qui est un int et la clé primaire
- **First_name** qui est un string de moins de 45 caractères
- **Last_name** qui est un string de moins de 45 caractères
- **Email** qui est un string de moins de 45 caractères

StudentDbUtil

Cette classe nous permet de faire la **connexion avec la database** notamment grâce au fichier « context.xml ». Elle nous permet donc d'exécuter des requêtes SQL pour ajouter, modifier ou supprimer un élève dans la database mais aussi récupérer la liste des élèves par exemple.

Nous avons bien déclaré cette classe comme une « *Managed Bean* » pour qu'elle fasse bien partie des ressources JSF, mais aussi comme « *Application Scoped* » ce qui veut dire qu'elle sera créée une seule fois pour notre application, et qu'elle sera partagée par tous les utilisateurs.

Student

La classe Student nous permet de **créer un élève** afin de pouvoir, par la suite, modifier la database à l'aide de celui-ci.

On a eu besoin de **surcharger** les constructeurs de cette classe pour différentes raisons :

- On voulait pouvoir créer un objet Student au sein de l'application sans avoir son id étant donné que l'id d'un élève n'est créé qu'au moment de son enregistrement dans la DB.
- On voulait pouvoir créer un objet Student au sein de l'application sans argument afin de pouvoir l'initialiser dans le xhtml « add-student ».

Nous avons bien déclaré cette classe comme une « Managed Bean » pour qu'elle fasse bien partie des ressource JSF, mais aussi comme « Request Scoped » ce qui veut dire qu'elle sera créée à chaque fois qu'une nouvelle « request » sera créée.

StudentManager

Cette classe nous permet de faire la **connexion entre la DB et l'interface utilisateur** (xhtml). Contrairement aux servlets, cette unique classe nous permet de gérer toutes les pages xhtml. Nous n'avons donc plus besoin de créer un servlet par page xhtml.

Cette classe est donc composé de 6 fonctions :

- Les 5 premières permettent de **gérer la DB** :
 - Ajout, suppression ou modification d'un élève
 - Charger un élève à partir de son ID
 - Charger la liste des élèves
- Une fonction qui permet de **vérifier** qu'un email respect le bon format

Nous avons bien déclaré cette classe comme une « Managed Bean » pour qu'elle fasse bien partie des ressource JSF, mais aussi comme « Session Scoped » ce qui veut dire qu'elle sera créée une fois pour la session de navigation de l'utilisateur, elle est unique pour cet utilisateur.

WebStudentBook JSF/JPA

Nous allons maintenant vous présenter notre partie JPA.

Database

La base de données MySQL

Nous avons utilisé la même DB que pour JDBC.

StudentDbUtil1

C'est dans cette première classe qu'il y a une grosse différence par rapport à la partie précédente. Comme auparavant cette classe nous permet de faire la connexion avec la DB mais cette fois ci grâce au fichier « persistence.xml ». Elle nous permet donc **d'exécuter des requêtes SQL** pour ajouter, modifier ou supprimer un élève dans la DB mais aussi récupérer la liste des élèves par exemple. La particularité de JPA, c'est que nous n'avons **pas besoin de créer des instructions SQL** pour exécuter des requêtes SQL. En effet nous utilisons le langage JPQL (Java Persistence Query Language).

Student1

C'est dans cette seconde classe que se situe l'autre grosse différence avec la partie précédente. Cette classe nous permet de **créer l'entité Student**. Nous indiquons donc tout d'abord à notre serveur que notre bean Student va devenir une entité avec l'annotation « @Entity », on indique aussi la table avec « @Table = « student » ».

Enfin nous devons **déterminer tous les attributs** avec « @Column (name = « ... ») » lorsque celui-ci porte un nom différent dans la DB et dans la classe. Aussi les annotations « @id et @GeneratedValue » nous permettent de définir la clé primaire.

StudentManager1

Cette classe est exactement la même que pour JDBC, excepté le fait qu'elle utilise Student1 et pas Student, et StudentDbUtil1 et pas StudentDbUtil.

Nous allons maintenant vous présenter le rendu de notre application.

CSS

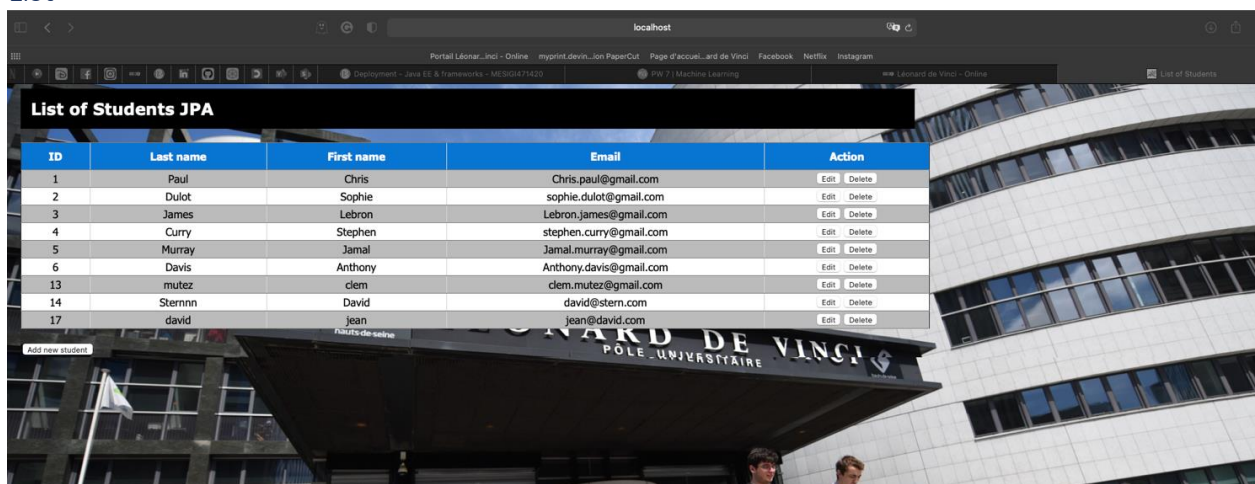
Avant de vous montrer nos résultats, nous tenons à préciser que nous avons inclus un CSS dans notre projet qui reprend celui vue dans les TD précédents.

XHTML

Pour les pages xhtml, chaque page existe en **deux versions, une JDBC et une JPA** (les versions JPA ont le même nom que celle JDBC mais avec un « 1 » en plus à la fin). Sinon le contenu des 2 versions est le même sauf pour l'appel des fonctions, la redirection des pages et la création de l'entité élève (soit de Student (JDBC) soit Student1 (JPA)).

Dans la suite de la présentation nous allons vous montrer la version JPA des pages xhtml, pour accéder à la version JDBC il suffit d'enlever les 1 à la fin des pages xhtml. Vous pourrez bien voir la différence car il y aura écrit JDBC au lieu de JPA sur chaque page.

List



ID	Last name	First name	Email	Action
1	Paul	Chris	Chris.paul@gmail.com	Edit Delete
2	Dulot	Sophie	sophie.dulot@gmail.com	Edit Delete
3	James	Lebron	Lebron.james@gmail.com	Edit Delete
4	Curry	Stephen	stephen.curry@gmail.com	Edit Delete
5	Murray	Jamal	Jamal.murray@gmail.com	Edit Delete
6	Davis	Anthony	Anthony.davis@gmail.com	Edit Delete
13	mutez	clem	clem.mutez@gmail.com	Edit Delete
14	Sternnn	David	david@stern.com	Edit Delete
17	david	jean	jean@david.com	Edit Delete

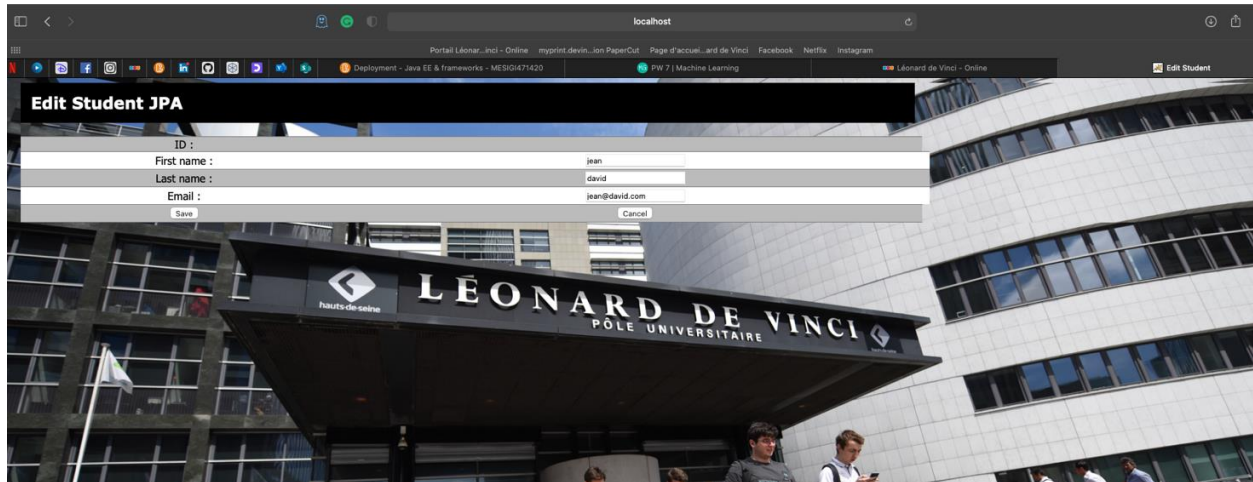
[Add new student](#)

Dès qu'on se connecte à <http://localhost:8080/WebStudentBookJSF-JPA/> on est bien dirigé vers la page ListStudent1.xhtml qui récupère et affiche la liste des étudiants. On peut alors choisir soit **d'ajouter, modifier ou supprimer un élève**.

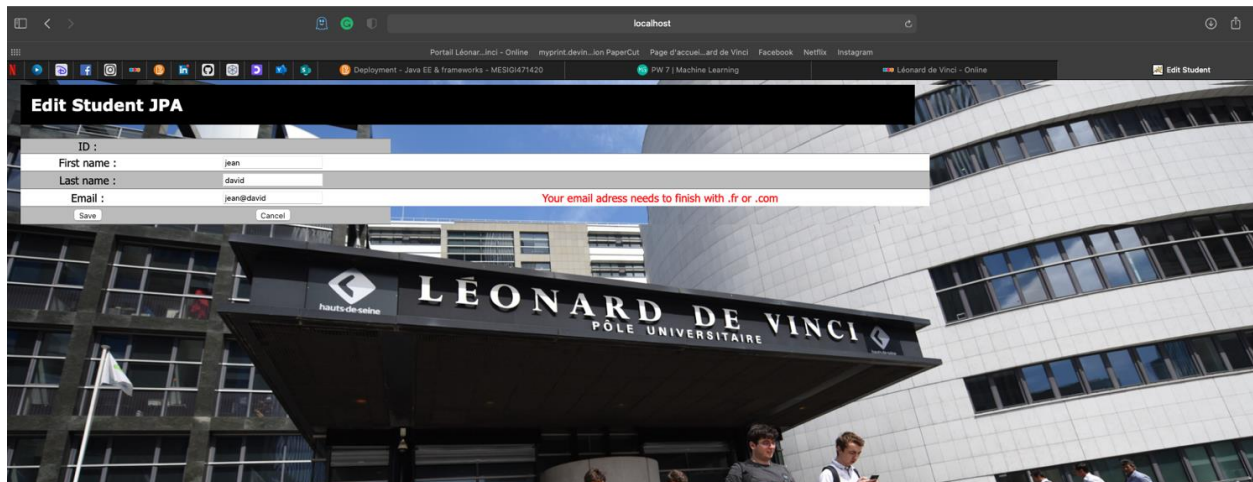
Le bouton *add* nous retourne vers la page « Add-student1.xhtml ».

Le bouton *edit* fait appel à la fonction « loadStudent() » de « StudentManager1 » qui crée une Request avec l'élève sélectionné et redirige vers la page « Edit-student1.xhtml ».
Enfin le bouton *delete* fait appel à la fonction « deleteStudent() » de « StudentManager1 » et permet de supprimer l'élève sélectionné.

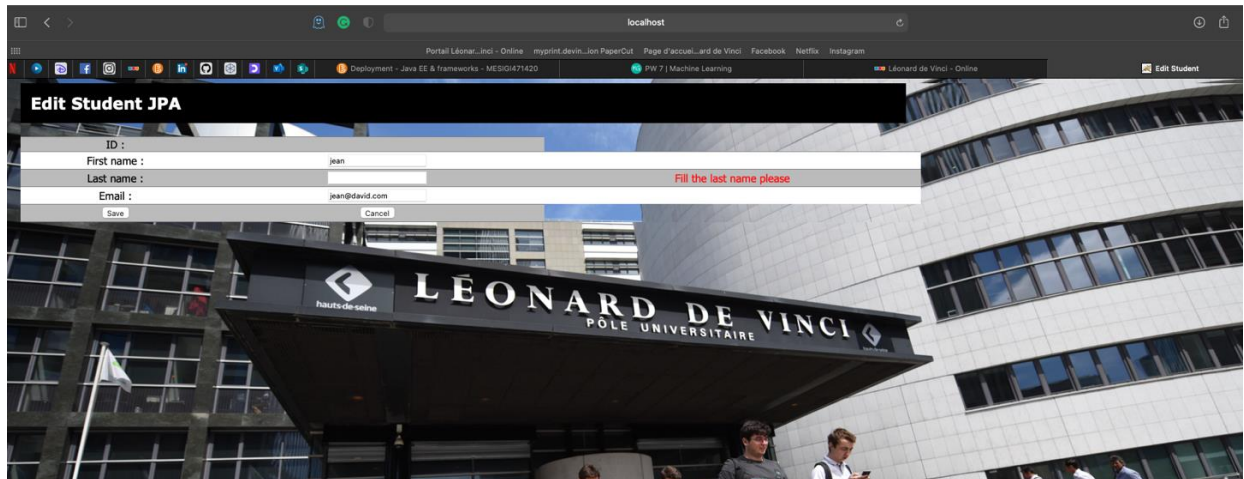
Edit



Voici comment se présente la page pour modifier un élève. On ne peut pas voir l'ID car nous avons considéré qu'un élève **ne pouvait pas le modifier**, c'est pourquoi dans le.xhtml il est appelé en « inputHidden ». Nous avons été obligés de l'appeler de cette manière pour pouvoir ensuite le rentrer dans la DB.



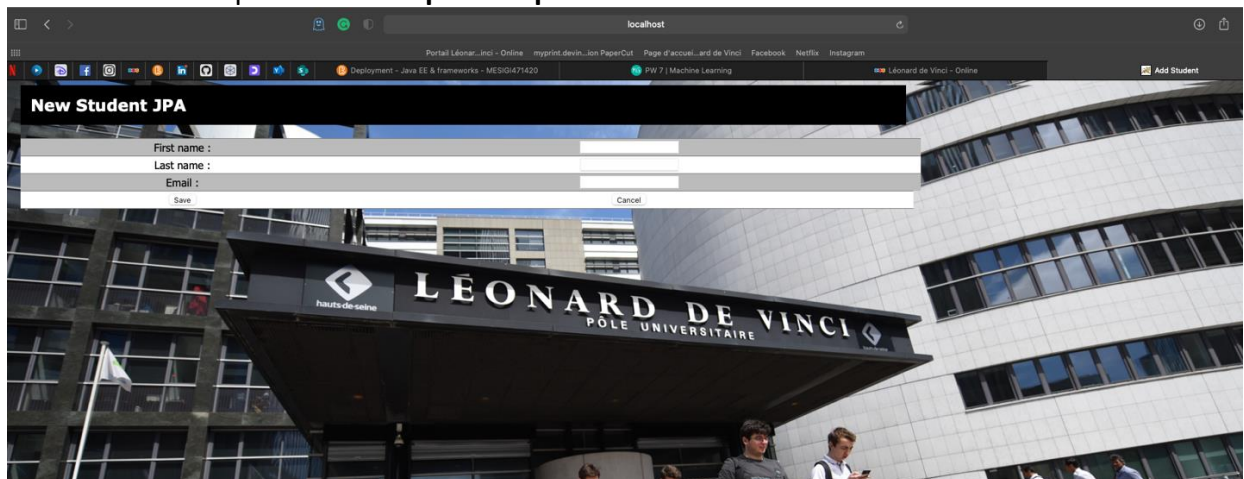
Nous pouvons voir ici que si un élève essaye de rentrer un mauvais email, un **message d'erreur** lui indique l'erreur commise. Nous avons créé la fonction « validateEmail » dans la classe « StudentManager1 » pour effectuer une **validation** « customiser » afin de rentrer plusieurs facteurs de validation (présence d'une @, se finis par .com ou .fr, etc...)



Nous pouvons voir ici que si l'élève oublie de rentrer son nom, un message d'erreur apparait lui indiquant de rentrer son nom. Nous avons créé la même sécurité pour le prénom.

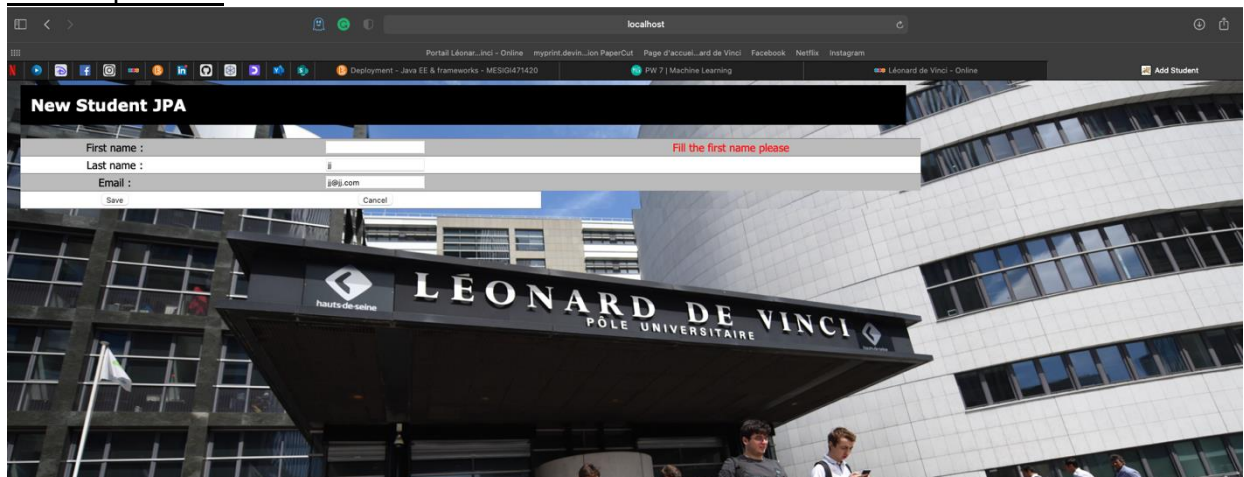
Add

Voici comment se présente la page pour modifier un élève. On ne peut pas voir l'ID car nous avons considéré qu'un élève ne **pouvait pas le créer lui-même**.

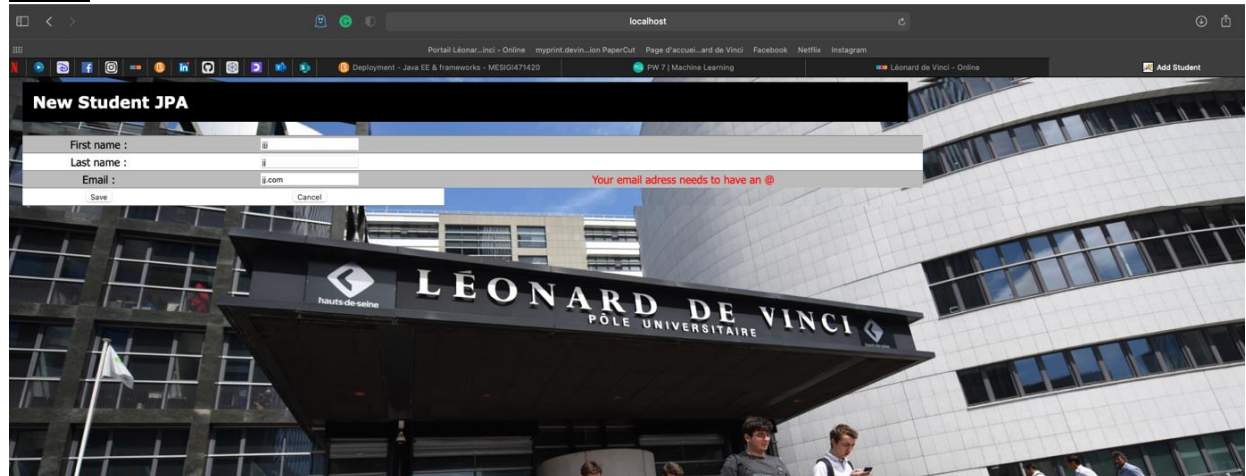


Nous avons aussi appliqué les mêmes systèmes de validation pour le nom, le prénom et l'email pour la page add.

Nom et prénom:



Email:



Deployment

Enfin nous avons déployé notre application en exportant notre projet dans un fichier war.