Compte-Rendu : TP2 de JAVAFX

Réalisé par : Ayoub MAGHDAOUI

Filière : BDCC2

Année : 2021-2022

Contenu :

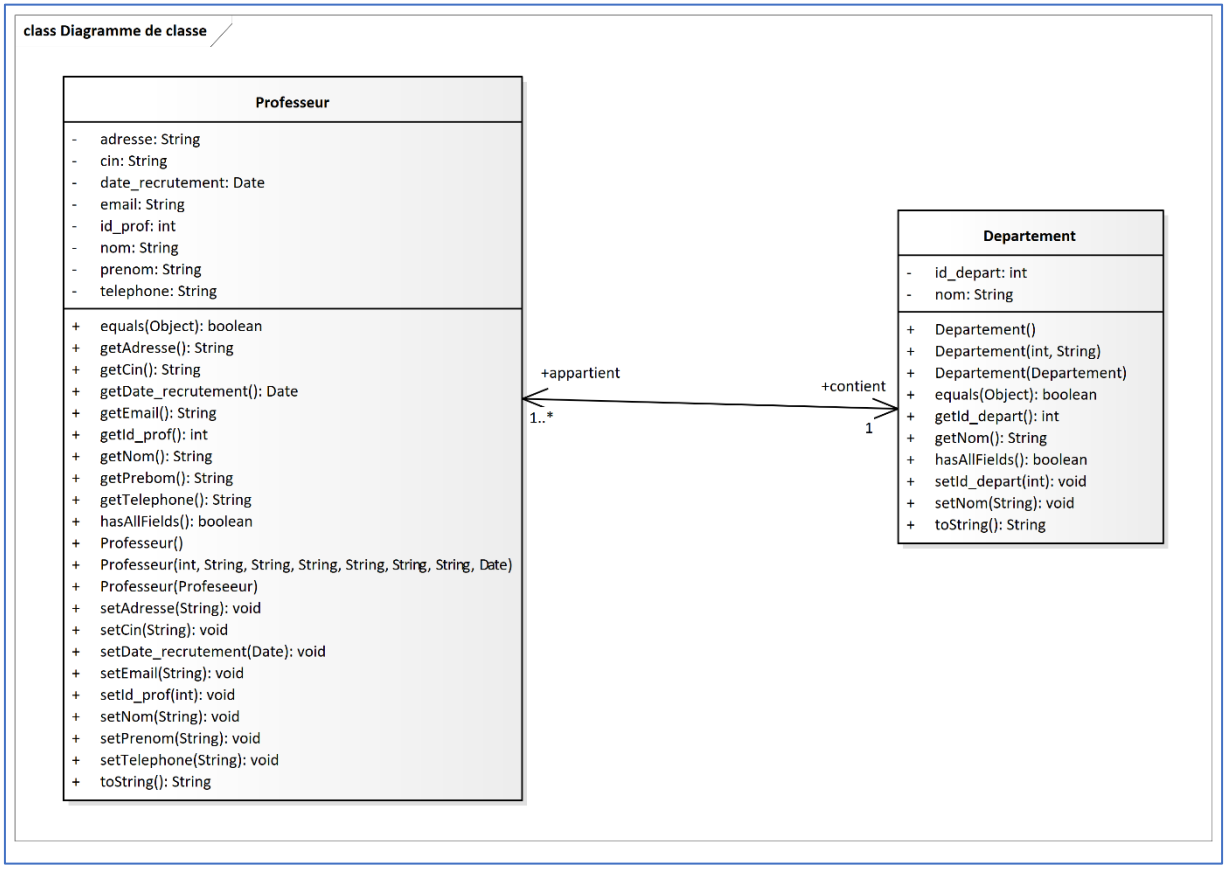
Introduction :

Ce document est un compte rendu d'un travail pratique du module POO en Java, il a comme objectif l'utilisation de la bibliothèque JavaFX pour créer une application de gestion de départements et de professeurs afin d'appliquer et de maitriser certaines notions du cours (la classe persistante, le design pattern Singleton…).

Dans ce compte rendu je vais commencer par la présentation de quelques éléments de la phase de conception, ensuite la réalisation avec le codage tout en suivant l'ordre existant dans la feuille de l'énoncé.

1- La conception :

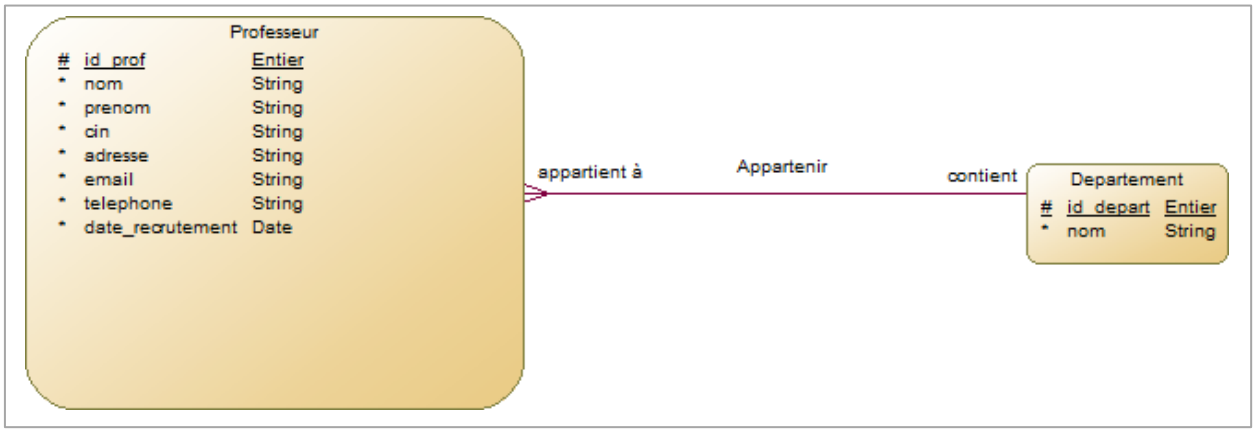
Pour présenter les entités agissantes dans cette application et la relation entre eux, j'ai créé le diagramme de classe suivant, qui présente chaque entité avec ses propriétés et ses méthodes. C'est certes que cette modélisation va recevoir d'autres additions afin de l'implémenter au sein de l'application et de l'adapter avec les besoins techniques, mais elle reste la modélisation de base.



1: Diagramme de classe créé avec Entreprise Architecte

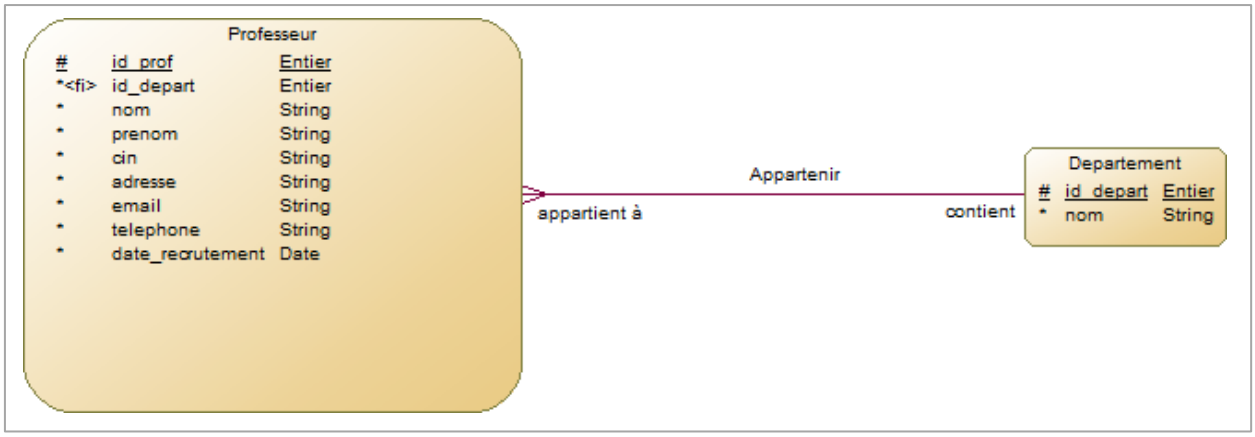
Le **M**odèle **L**ogique de **D**onnées (MLD) est un outil de modélisation merise, consiste à décrire la structure des entités ou données utilisés prenant en compte les relations entre ces entités, il m'a donnée une idée claire sur la structure de la base de données que je vais utiliser pour cette application.

Mais avant de créer le MLD il faut commencer par le MCD (**M**odèle **C**onceptuel de **D**onnées), permet de représenter d'une façon formelle, facile et compréhensible les données qui seront utilisés dans le système d'informations.



2: MCD Créé avec PowerAMC

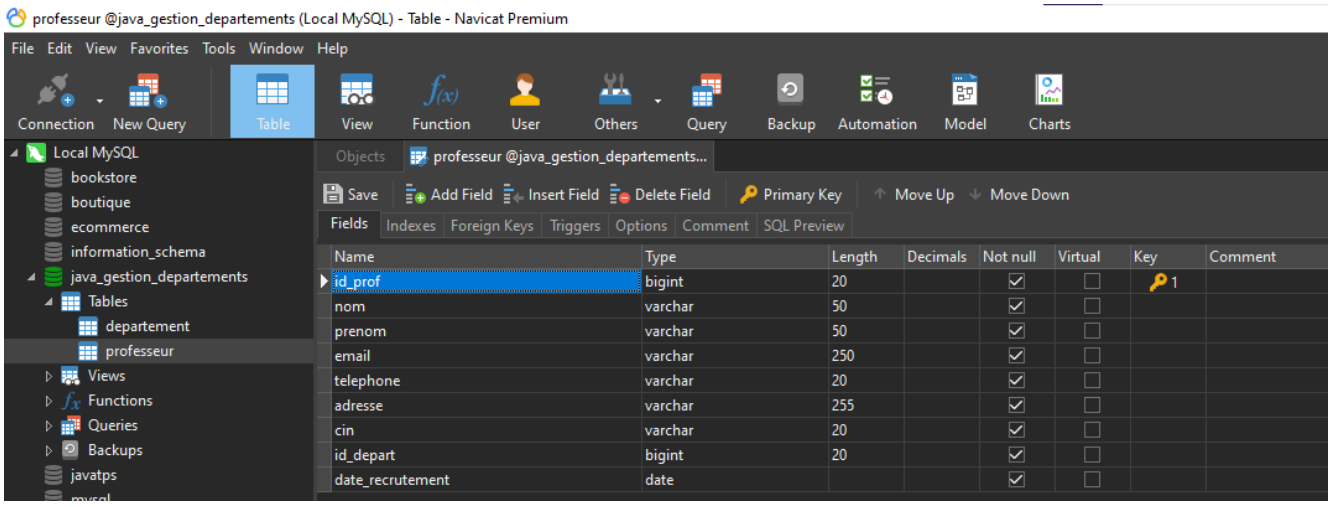
Après la création du MCD, le MLD est automatiquement généré par le logiciel PowerAMC.



3: MLD généré par PowerAMC

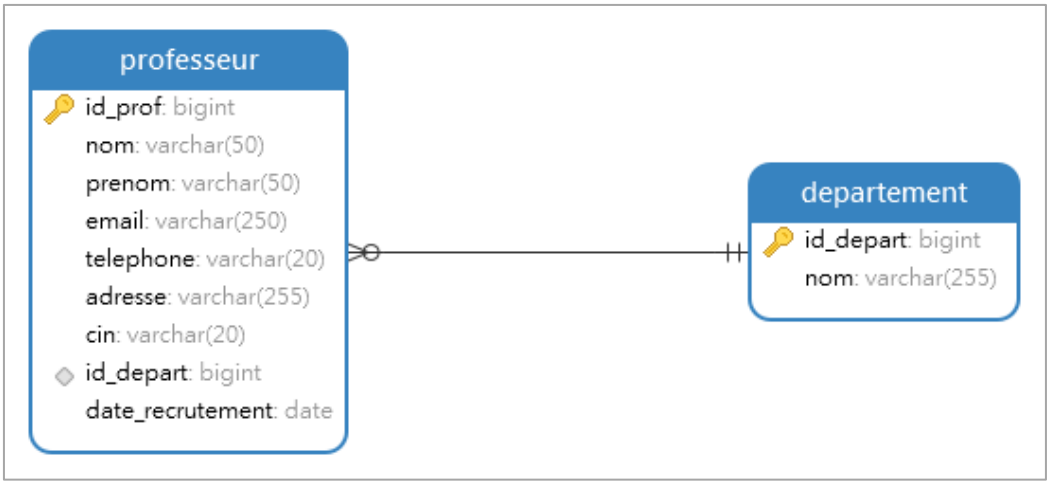
2- La réalisation :

J'ai commencé cette partie par la création de la base de données sur le SGBD MySQL avec ses tables, en utilisant un le logiciel 'Navicat'.



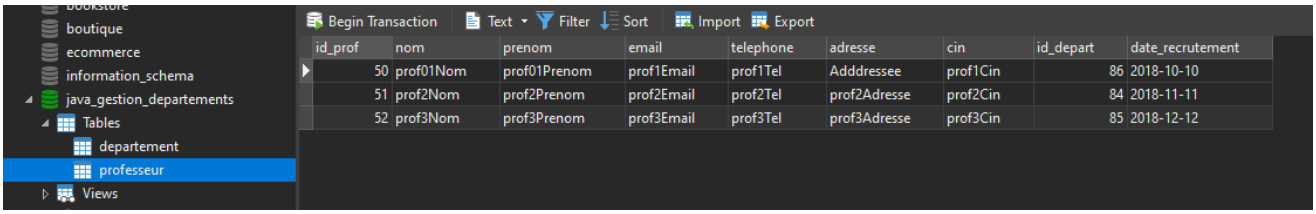
4: Création des tables dans la base de données

Pour illustrer la structure des tableaux et la relation entre eux, je propose le MPD (**M**odèle **P**hysique de **D**onnées) suivant :

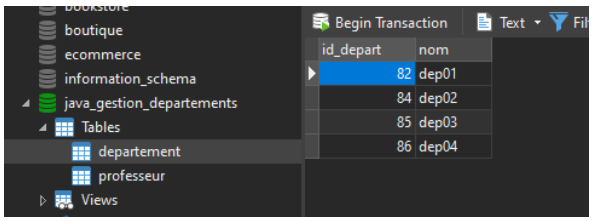


5: MPD de la Base de données généré par Navicat

Voici les deux tableaux créés avec quelques données dedans :



6: Tableau des professeurs



7: Tableau des départements

Après la création des tables à la BD, j'ai créé les classes persistantes des entités Professeur et Département, avec les différents attributs et méthodes. Ici j'ai utilisé le type 'StringProperty' au lieu de type 'String' pour bénéficier de la fonctionnalité de binding bidirectionnel, pour faciliter la connexion entre les champs d'un formulaire et les attributs d'objet d'une entité. C'est-à-dire, au lieu d'attribuer à chaque champ du formulaire un Event qui va prendre la nouvelle valeur saisie et l'affecter à l'attribut adéquat de l'objet, il suffit d'utiliser le type 'StringPropery' et d'établir la liaison entre chaque champ et attribut de l'objet.

Professeur.java | Departement.java

```java
package Entities;

import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;
import java.sql.Date;

public class Professeur {

    private int id_prof;
    private StringProperty nom;
    private StringProperty prenom;
    private StringProperty cin;
    private StringProperty adresse;
    private StringProperty telephone;
    private StringProperty email;
    private Date date_recrutement;
    private Departement departement;

    public Professeur() {
        this( nom: "", prenom: "", cin: "", adresse: "", telephone: "", email: "", date_recrutement: null );
        this.date_recrutement = new Date( year: 2000-1900, month: 1, day: 17);
    }
```

8: Classe persistante Professeur

Professeur.java | Departement.java

```java
package Entities;

import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;
import java.util.ArrayList;
import java.util.List;

public class Departement {

    private int id_depart;
    private StringProperty nom;
    private List<Professeur> professeurs;


    public Departement() { this( nom: ""); }

    public Departement(String nom) {
        this.nom =  new SimpleStringProperty(nom);
        professeurs = new ArrayList<Professeur>();
    }
```

9: Classe persistante Département

Ensuite, j'ai créé l'interface 'IMetier' qui contient les signatures de toutes les méthodes permettant de lire ou de manipuler les données.

```java
public interface IMetier {

    public void ajouterProfesseur(Professeur professeure);
    public void supprimerProfesseur(Professeur professeure);
    public void modifierProfesseur(Professeur professeure);
    public void affecterProfesseurDepartement( Professeur professeure, Departement departement);
    public List<Professeur> getAllProfesseurs();

    public void ajouterDepartement( Departement departement);
    public List<Departement> getDepartements();
    public void supprimerDepartement( Departement departement);
    public void modifierDepartement( Departement departement);
    public List<Professeur> getProfesseursByDepartement(Departement departement);

}
```

10: Interface IMetier

Par la suite j'ai créé la classe 'SingletonConnexionDb' qui va me permettre de fournir une seule connexion à utiliser dans toute l'application, créée statiquement :

```java
public class SignletonConnexionDB {

    private static String connectionString = "jdbc:mysql://localhost:3306/java_gestion_departements";
    private static String username = "root";
    private static String password = "";

    private static Connection connection;

    static {
        try{
            Class.forName( "com.mysql.cj.jdbc.Driver" );
            connection =  DriverManager.getConnection( connectionString, username, password);
        } catch( Exception exc){
            System.err.print( " [SignletonConnexionDB::Class] Exception> "+exc.getMessage()+"\n==> stack : \n"+exc.getStackTrace() );
        }
    };

    public static Connection getConnection() { return connection; }
}
```

11: classe de connexion à la BD

Dans la classe 'IMetierImplementation' j'ai implémenté l'interface 'IMetier' et j'ai défini les corps des méthodes :

```java
public class IMetierImplementation implements IMetier {

    private Connection connection;

    public IMetierImplementation() {
        connection = SignletonConnexionDB.getConnection();
    }

    @Override
    public void ajouterProfesseur(Professeur professeure) {
        try{
            PreparedStatement ajoutPreStatement = connection.prepareStatement( sql: "INSERT INTO professeur( nom, prenom, e
            ajoutPreStatement.setString( parameterIndex: 1, professeure.getNom());
            ajoutPreStatement.setString( parameterIndex: 2, professeure.getPrenom());
            ajoutPreStatement.setString( parameterIndex: 3, professeure.getEmail());
            ajoutPreStatement.setString( parameterIndex: 4, professeure.getTelephone());
            ajoutPreStatement.setString( parameterIndex: 5, professeure.getAdresse());
```

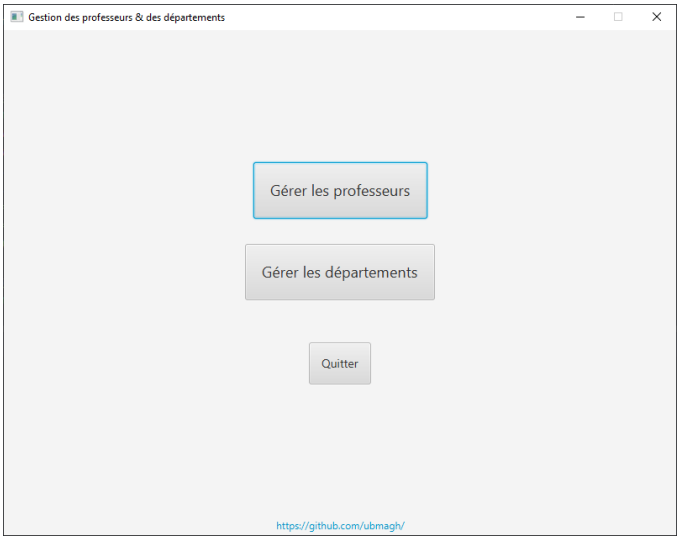
12: Implémentation de l'interface IMetier

Après la création des classes persistante, les tables dans la BD, et les méthodes nécessaires pour manipuler les données, maintenant je vais créer une application console pour tester les méthodes implémentées. Dans ce programme, je crée des professeurs et des départements, et j'affecte des professeurs à certains départements, je fais aussi la modification et la suppression des objets de la BD.

```java
public class ApplicationTestConsole {

    public static void main(String[] args) {

        IMetierImplementation iMetierImplementation = new IMetierImplementation();

        // Ajouter des départements :
        System.out.print("\n\n\t ==> Insérer deux départements : ");
        Departement dep1 = new Departement( nom: "Dep1");
        Departement dep2 = new Departement( nom: "Dep2");
        iMetierImplementation.ajouterDepartement(dep1);
        iMetierImplementation.ajouterDepartement(dep2);
        System.out.print("\n#=>dep1: "+dep1);
        System.out.print("\n#=>dep2: "+dep2);
        System.out.print("\n## Insértion terminée ");

        // modifier les departements Créés
        System.out.print("\n\n\t ==> Modifier les deux départements :   dep1-->dep01  et   dep2-->dep02");
        dep1.setNom("dep01");
```

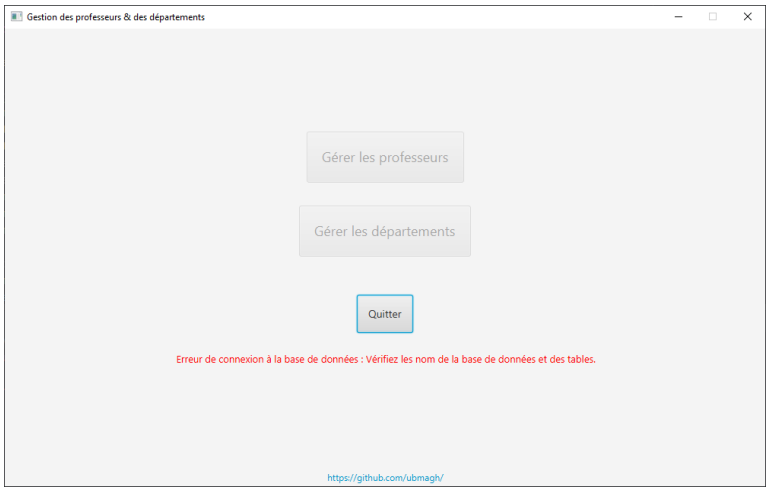
13: Programme pour tester les méthodes de manipulation des données à la BD

Pour réaliser l'interface utilisateur de l'application, j'ai décidé de faire trois interfaces principales ; une interface d'accueil, une autre pour gérer les départements et la dernière pour gérer les professeurs avec la possibilité de naviguer entre les trois interfaces.



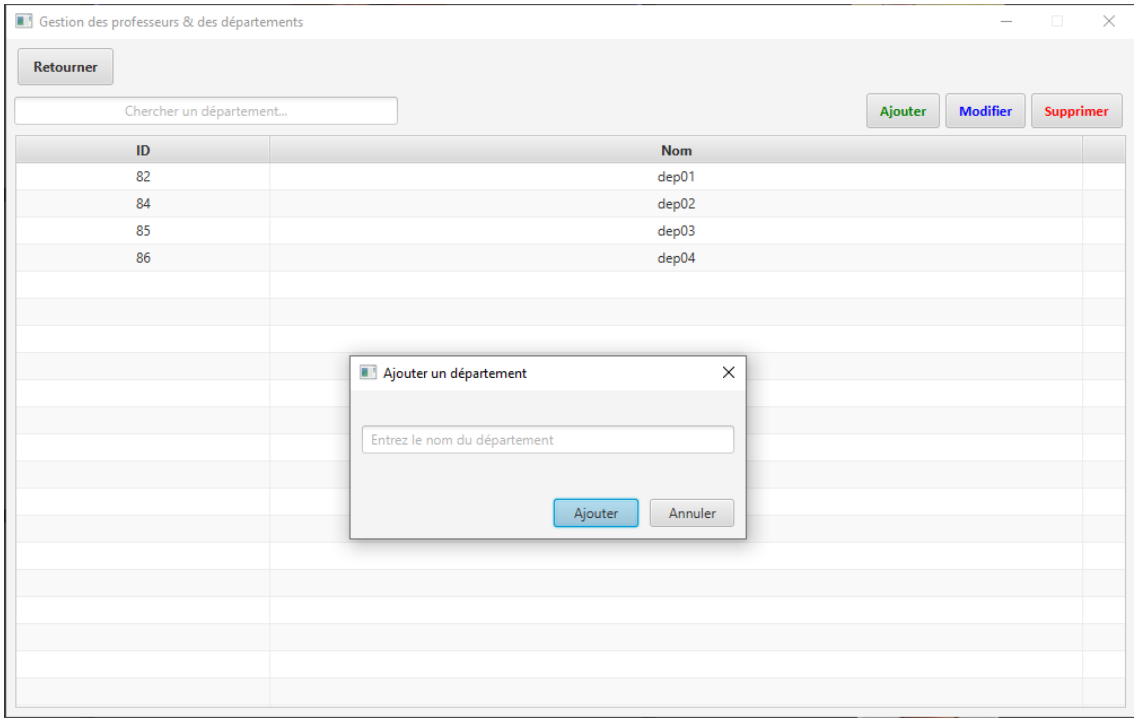
14: Interface d'accueil

S'il y a un problème dans la connexion vers la Base de données vous ne pourriez pas avancer, un message d'erreur est affiché dans l'interface d'accueil avec les deux boutons désactivés.



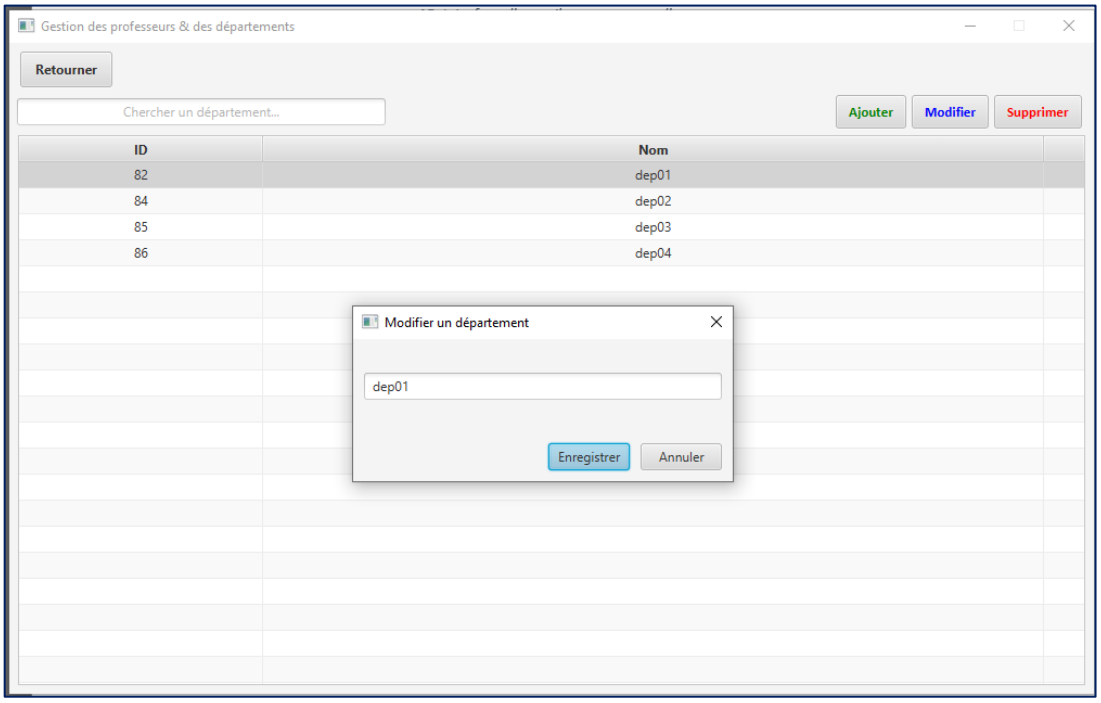
15: Interface d'accueil avec message d'erreur

Voici l'interface de gestion des départements, il contient un champ de recherche sur la volé, et trois boutons pour ajouter modifier et supprimer un département.



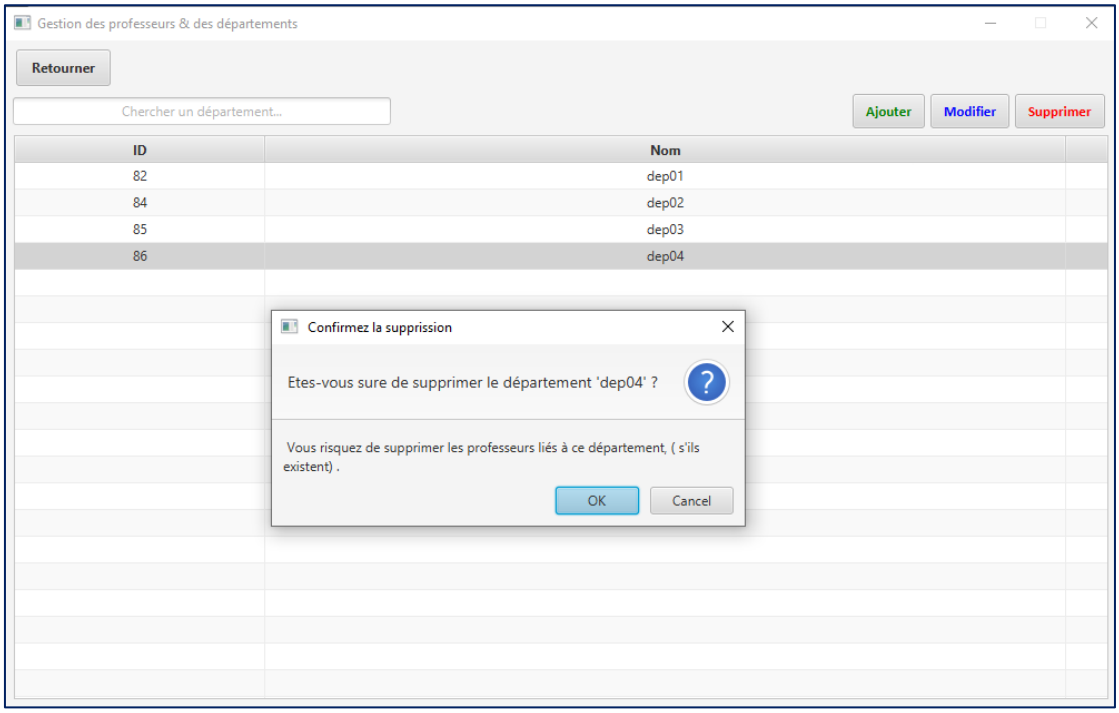
16: Interface de gestion des départements avec dialogue d'ajout

Pour modifier un département, je sélectionne la ligne et j'appuie sur modifier :



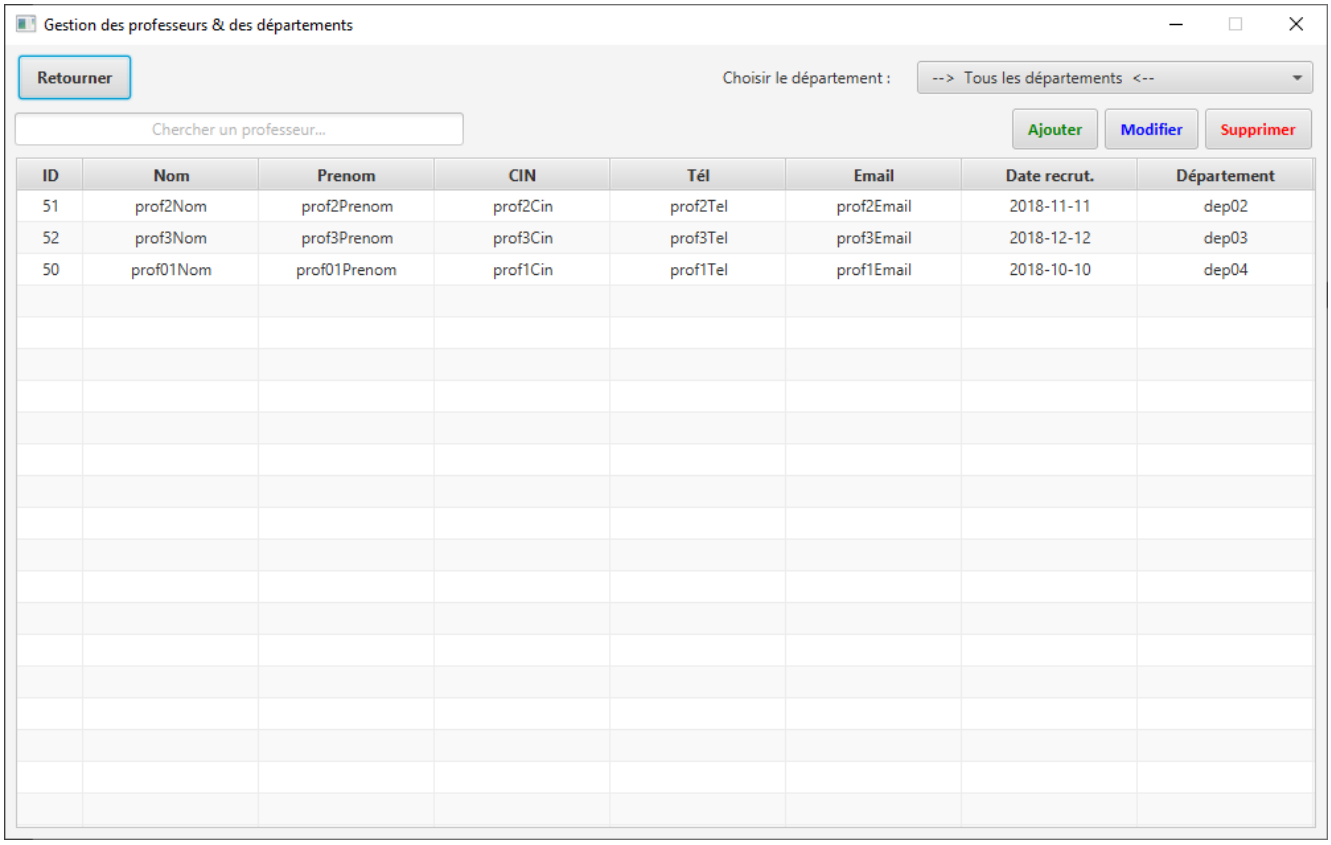
17: interface de gestion des départements ; modifier un département

De la même façon pour supprimer un département, sauf qu'il y a une boite de dialogue qui s'affiche pour confirmer l'action.



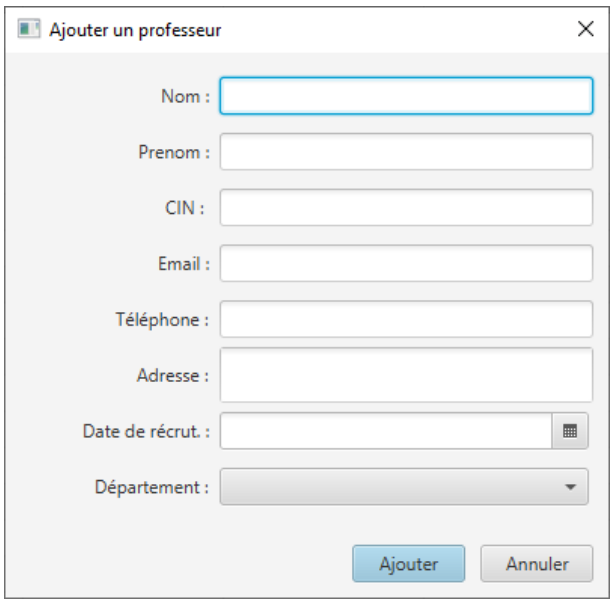
18: interface de gestion des départements ; supprimer un département

Maintenant je retourne vers l'interface d'accueil et je clique sur la gestion des professeurs voici l'interface qui va s'afficher, il y a aussi le champ de recherche avec les trois boutons, et aussi un menu déroulant pour choisir le département duquel l'application va charger les professeurs.



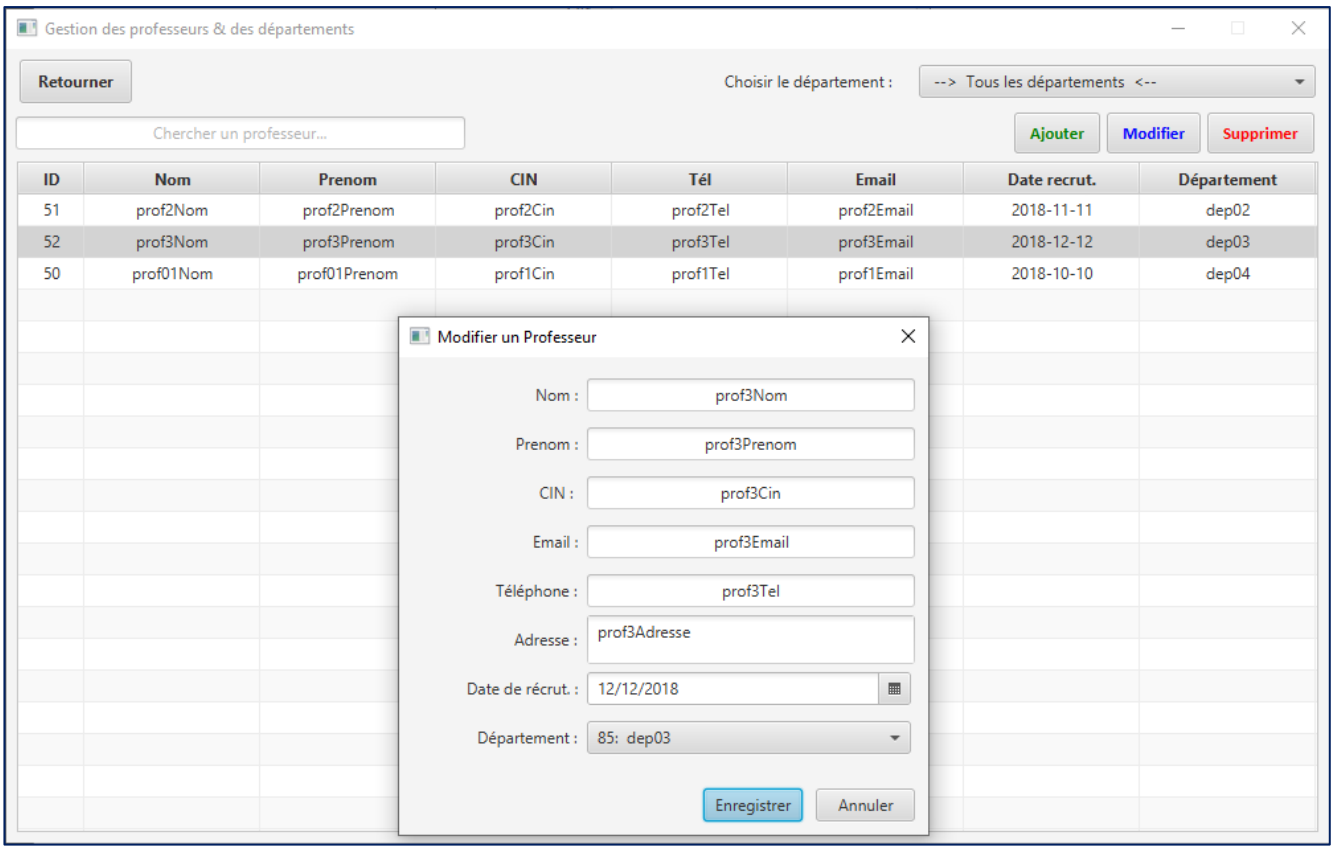
19: interface de gestion des professeurs

Pour ajouter un professeur je clique sur le bouton 'ajouter', la boite de dialogue suivante s'affiche pour saisir les informations du professeur :

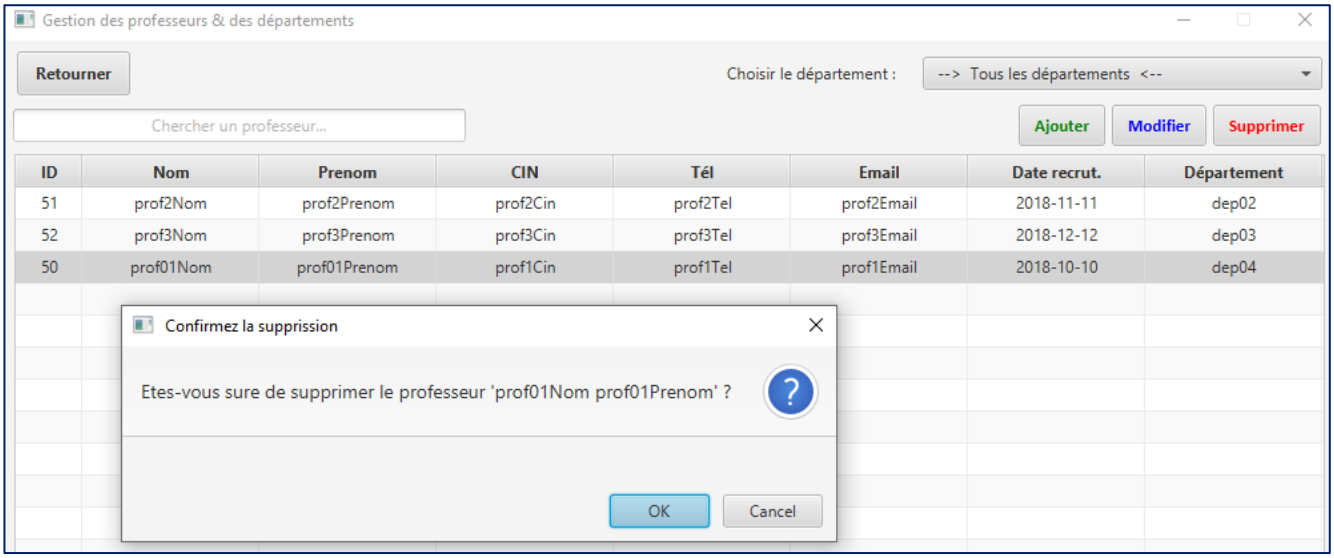


20: Boite de dialogue pour saisir les informations du nouveau professeur

Pour modifier un professeur :



21: Gestion des professeurs ; modifier un professeur



22: Gestion des professeurs ; supprimer un professeur