Abdelmalek Essaadi

Faculté Polydisciplinaire à Larache
Département Informatique

AJDOUR AYOUB

DCC0020/23

2023/2024

HIBERNATE

Gestion d'école

Pour poursuivre le développement de notre projet de gestion d'école,nous envisageons d'intégrer Hibernate, un framework de persistance en Java, qui offre de nombreux avantages en termes de gestion des données et de simplification des opérations liées à la base de données
L'utilisation de Hibernate permettra également de profiter de fonctionnalités avancées telles que le caching, l'optimisation des requêtes, et la gestion des transactions, ce qui contribuera à améliorer les performances globales de l'application.
La version actualisée du code source, incluant la configuration Hibernate et les nouvelles fonctionnalités destinées à optimiser la gestion des filières et des étudiants, est disponible sur mon compte GitHub. Vous pouvez accéder à ces mises à jour en consultant le référentiel correspondant.

https://github.com/Ayoub-Ajdour/GestionD-Ecole

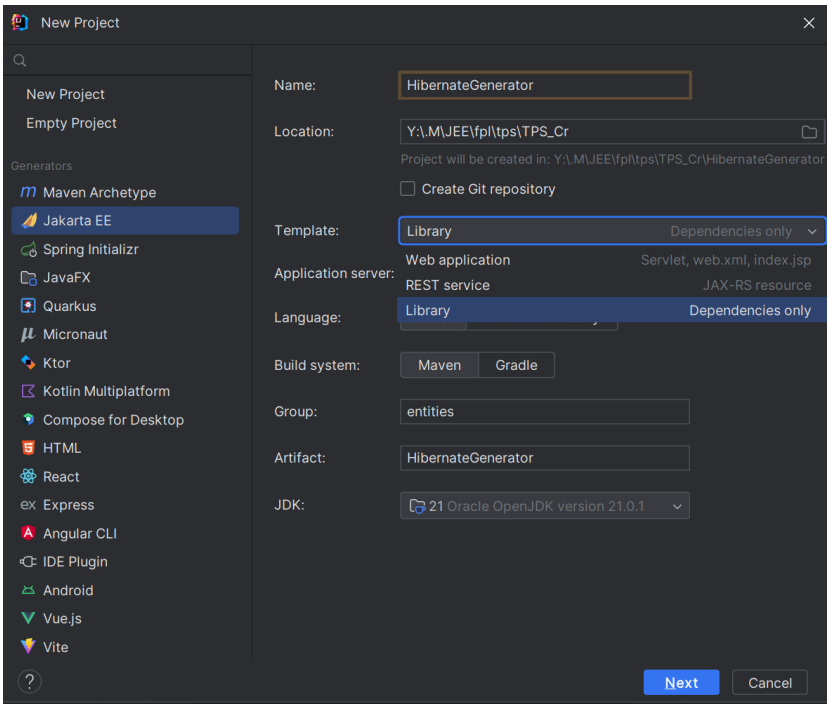


Figure 1: Création de la classe générateur Hibernate

D'abord, on crée une classe pour générer les classes de notre projet.

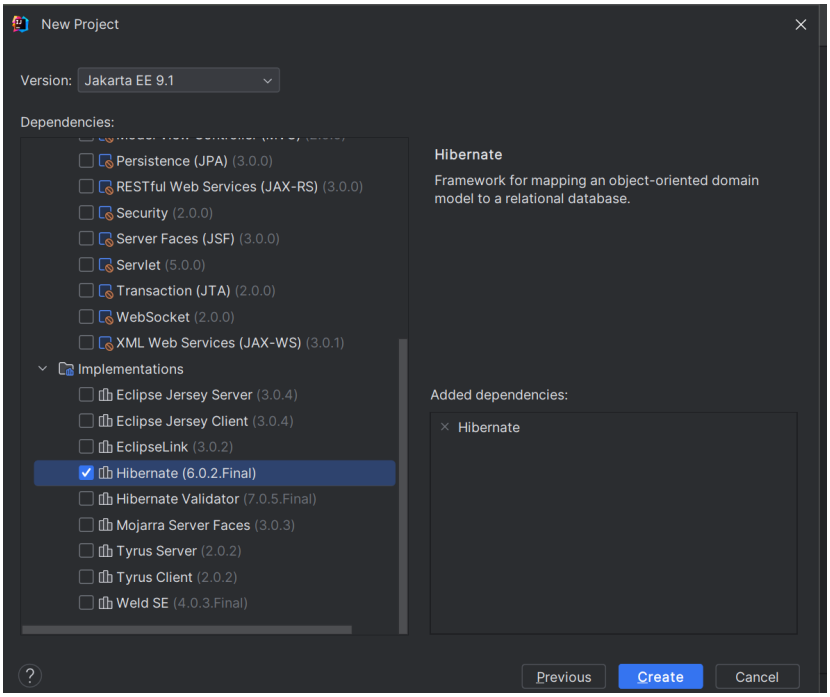


Figure 2: L'ajout d'implémentation

On ajoute l'implémentation Hibernate.

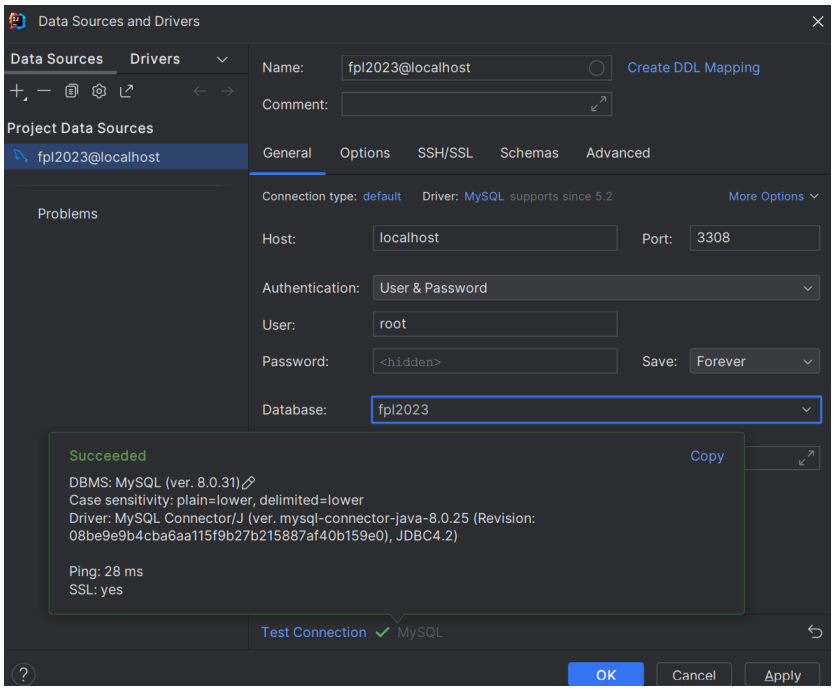


Figure 3: Vérification de la connexion à la base de données

On ajoute la source de données à IntelliJ et on vérifie si la connexion a réussi ou non.

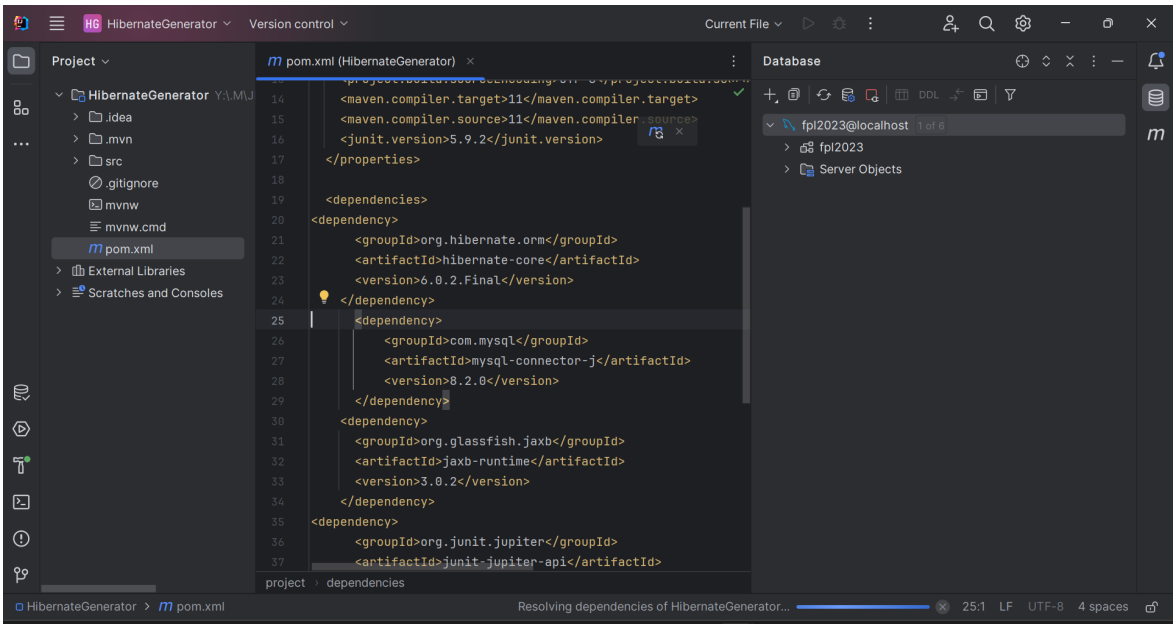


Figure 4: Ajout des dépendances MySQL et Hibernate

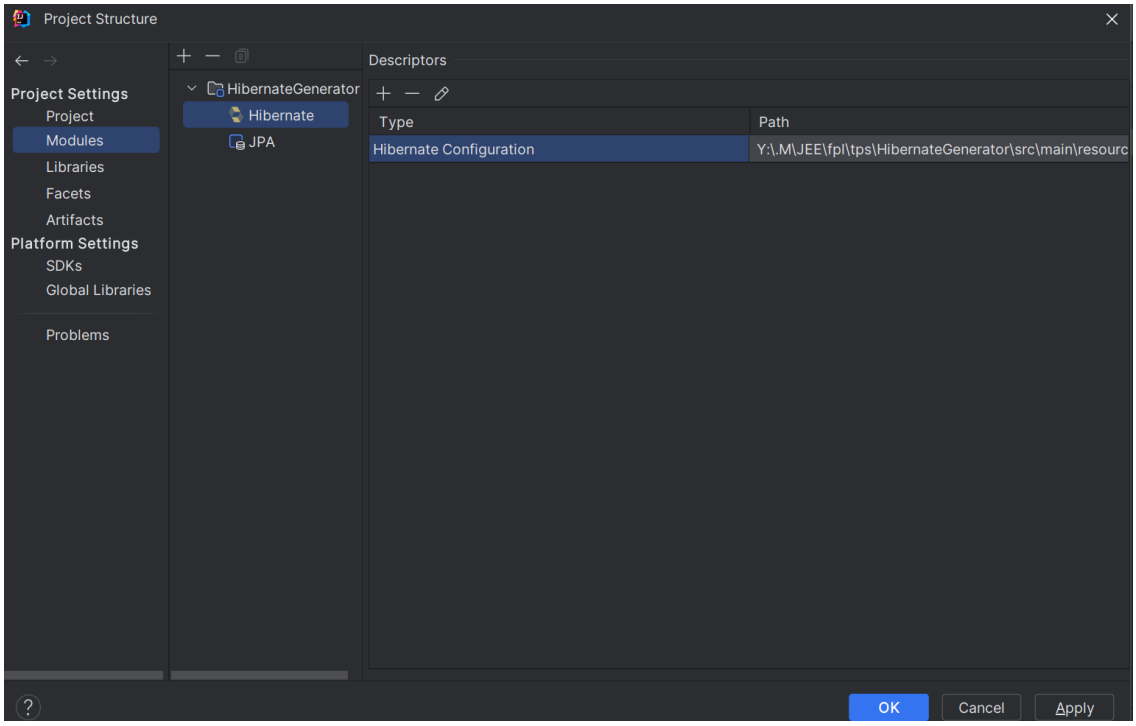


Figure 5: L'ajout du module Hibernate dans la structure du projet.

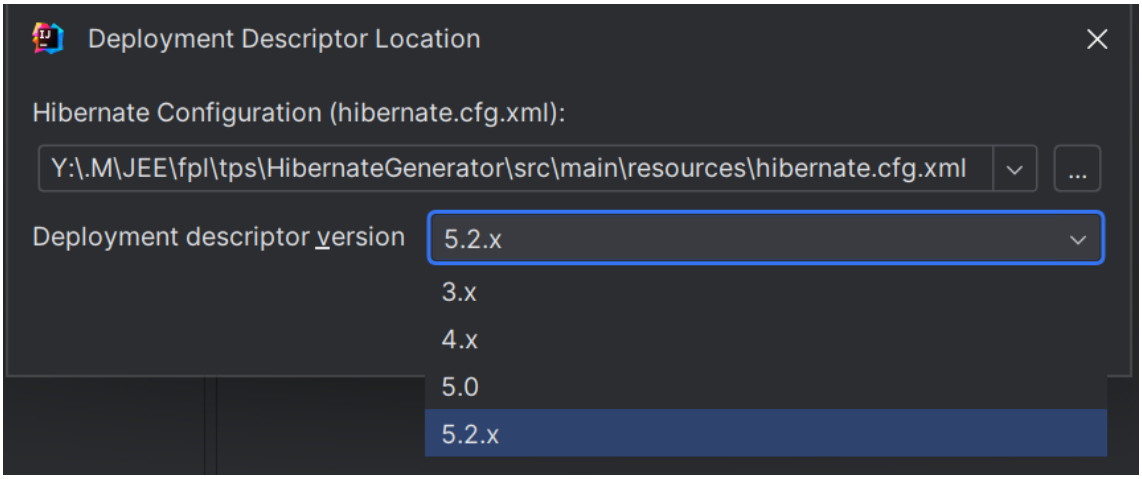


Figure 6:Ajout du fichier de configuration Hibernate sous le nom hibernate.cfg.xml avec une version de déploiement 5.0.

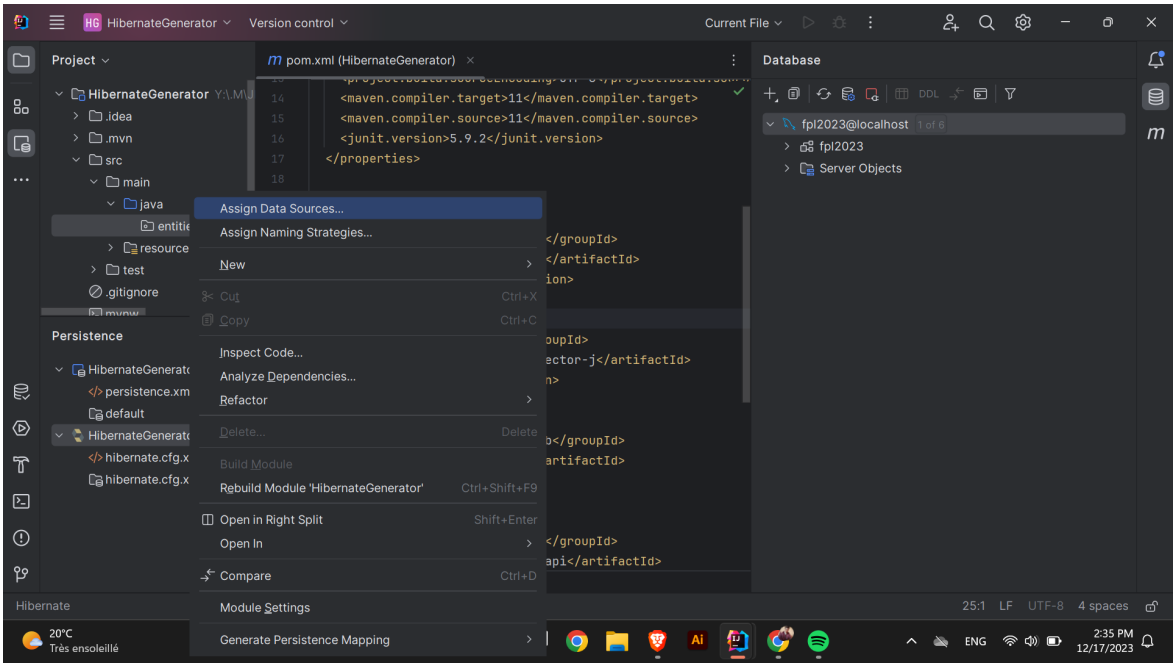


Figure 7: Assignation des données de la ressource (data resource).

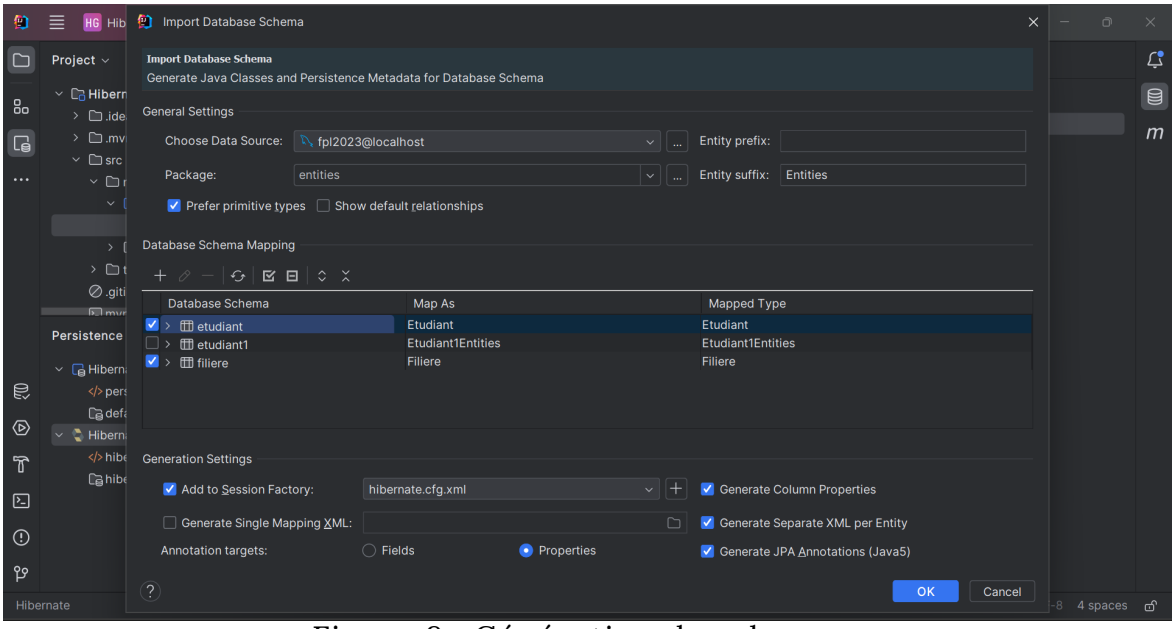
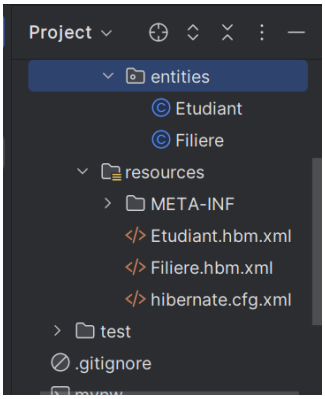


Figure 8: Génération des classes

Suite à la configuration initiale d'Hibernate et à l'assignation des données de la ressource, nous procédons à la génération des classes "Etudiant" et "Filiere" à partir du schéma de la base de données. Ces classes générées sont automatiquement placées dans le package "entities" du projet, assurant ainsi une représentation fidèle des entités dans le code source en accord avec la structure de la base de données.

```java
package entities;

import jakarta.persistence.*;

@Entity
public class Filiere {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "idFiliere", nullable = false)
    private int idFiliere;
    2 usages
    @Basic
    @Column(name = "Code", nullable = false, length = 50)
    private String code;
    2 usages
    @Basic
    @Column(name = "libelle", nullable = false, length = 50)
    private String libelle;

    2 usages
    public int getIdFiliere() { return idFiliere; }

    no usages
    public void setIdFiliere(int idFiliere) { this.idFiliere = idFiliere; }
```

Figure 10: Classe "Filiere" après génération




```java
    @Column(name = "idEtudiant", nullable = false)
    private int idEtudiant;

    2 usages
    @Basic
    @Column(name = "nom", nullable = false, length = 20)
    private String nom;

    2 usages
    @Basic
    @Column(name = "prenom", nullable = false, length = 20)
    private String prenom;

    2 usages
    @Basic
    @Column(name = "cne", nullable = false, length = 20)
    private String cne;

    2 usages
    @ManyToOne
    @JoinColumn(name = "filiere", nullable = false)
    private Filiere filiere;

    // Getters and setters
```

Figure 11: Classe "Etudiant" après génération

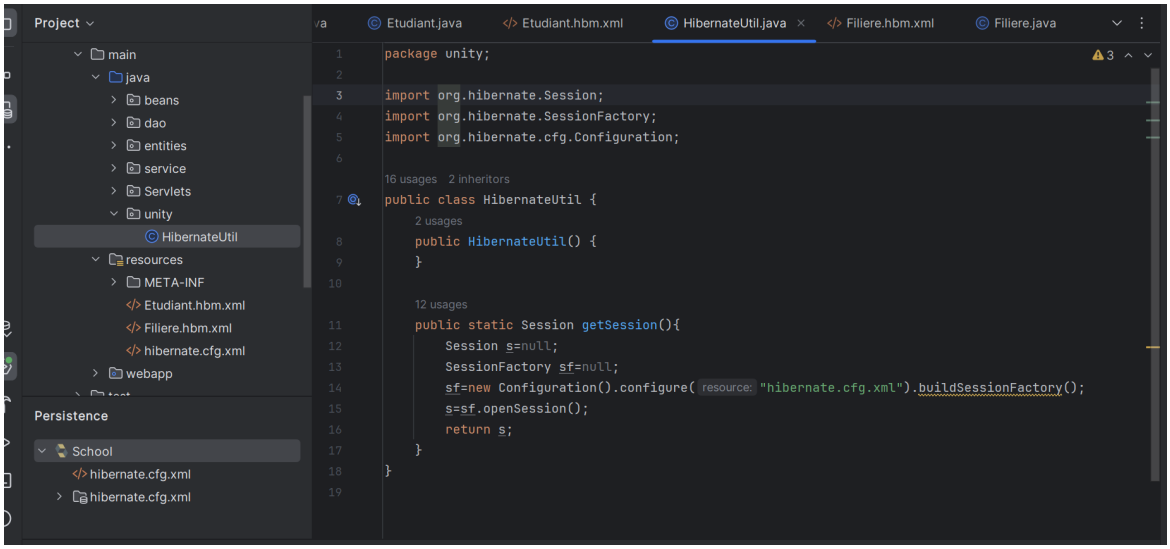


Figure 12: Classe "HibernateUtil"

Cette classe, "HibernateUtil", joue un rôle crucial dans notre projet. Elle contient une méthode spécifique chargée d'ouvrir une session Hibernate et de la configurer correctement. Cette étape est essentielle pour établir une connexion efficace avec la base de données, assurant ainsi la cohérence et la fiabilité des opérations effectuées par notre application. La configuration appropriée de la session garantit également une gestion optimale des transactions, contribuant ainsi à la robustesse et à la stabilité globale de notre système.

Les methodes de DAO :

```java
@Override
public boolean create(Etudiant c) {
    try{
        s= HibernateUtil.getSession();
        tx=s.beginTransaction();
        s.save(c);
        tx.commit();
    }catch (HibernateException e){
        e.printStackTrace();
        System.out.println("pb insert etud "+e.getMessage());
        return false;
    }finally {
        s.close();
    }

    return true;
}
```

Figure 13: La methode create()

```java
2 usages
@Override
public boolean update(Etudiant c) {
    try{
        s= HibernateUtil.getSession();
        tx=s.beginTransaction();
        s.update(c);
        tx.commit();
    }catch (HibernateException e){
        e.printStackTrace();
        System.out.println("pb update etud "+e.getMessage());
        return false;
    }finally {
        s.close();
    }
    return true;
}
```

Figure 14: La methode update()

```java
2 usages
@Override
public boolean delete(Etudiant c) {
    try{
        s= HibernateUtil.getSession();
        tx=s.beginTransaction();
        s.delete(c);
        tx.commit();
    }catch (HibernateException e){
        e.printStackTrace();
        System.out.println("pb delete etudiant "+e.getMessage());
        return false;
    }finally {
        s.close();
    }

    return true;
}
```

Figure 15: La methode delete()

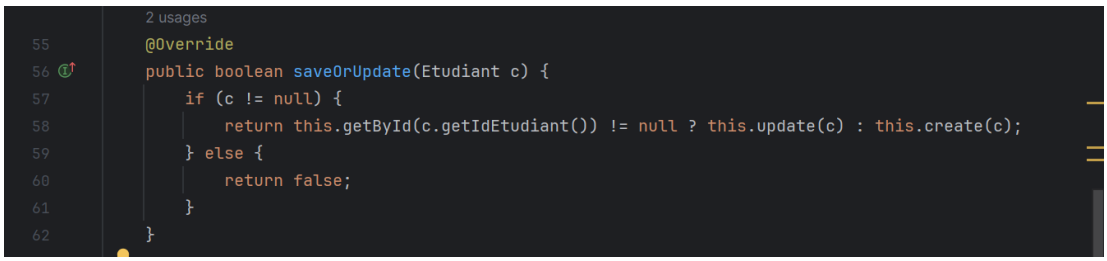
```java
@Override
public Etudiant getByCne(String cne) {
    try (Session session = HibernateUtil.getSession()) {
        tx = session.beginTransaction();

        String hql = "FROM Etudiant WHERE cne = :cne";
        Query<Etudiant> query = session.createQuery(hql, Etudiant.class);
        query.setParameter( s: "cne", cne);
        Etudiant etudiant = query.uniqueResult();
        tx.commit();

        return etudiant;
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

Figure 16: La methode getByCne()

```java
7 usages
@Override
public Etudiant getById(Integer c) {
    try (Session session = HibernateUtil.getSession()) {
        Transaction transaction = session.beginTransaction();
        Etudiant etudiant = session.get(Etudiant.class, c);
        transaction.commit();

        return etudiant;
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

Figure 15: La methode getByCne()

Figure 17: La methode getById()



```java
2 usages
@Override
public boolean saveOrUpdate(Etudiant c) {
    if (c != null) {
        return this.getById(c.getIdEtudiant()) != null ? this.update(c) : this.create(c);
    } else {
        return false;
    }
}
```

Figure 18: La methode saveOrUpdate()

```
    2 usages
    @Override
    public List<Etudiant> getAll() {
        try  {
            System.out.println("hnaaa  2");
            s = HibernateUtil.getSession();

            Query<Etudiant> query = s.createQuery( s: "from Etudiant ", Etudiant.class);
            List<Etudiant> etudiants = query.list();

            System.out.println(etudiants.toString());

            return etudiants;
        } catch (HibernateException e) Close
            e.printStackTrace();
            System.out.println("eroorr f affichage  etudiant"+e.getMessage());
            return Collections.emptyList();
        }
    }
}
```

Figure 19: La methode getAll()