Université Abdelmalek Essaadi

Faculté des Sciences et Techniques - Tanger

Département Génie Informatique

Filière :

« Logiciels et systèmes intelligents »

LSI

Lab 3 : Application JEE Distribuée avec EJB

Réalisé par :

Mohamed Amine BAHASSOU

Encadré par :

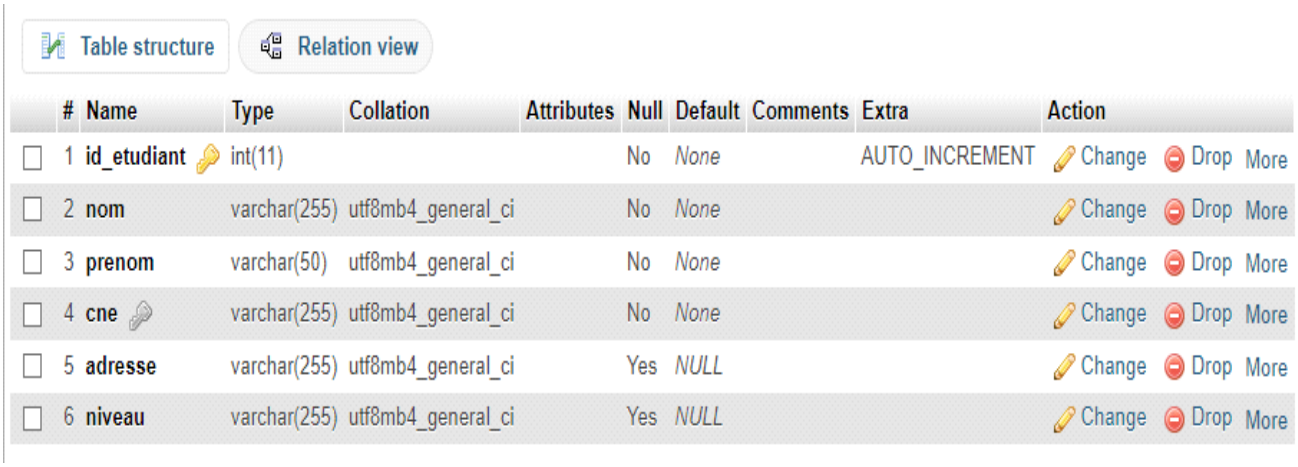
Prof. Lotfi EL AACHAK

Etape 1 : Création du base du Données :

Après on va ajouter la table étudiant avec la commande sql :

```sql
CREATE TABLE etudiant (
    id_etudiant INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prénom VARCHAR(50) NOT NULL,
    cne VARCHAR(15) UNIQUE NOT NULL,
    adresse TEXT,
    niveau VARCHAR(20)
);
```

Pour avoir une table comme ca :



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id_etudiant	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	nom	varchar(255)	utf8mb4_general_ci		No	None			Change Drop More
3	prenom	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More
4	cne	varchar(255)	utf8mb4_general_ci		No	None			Change Drop More
5	adresse	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More
6	niveau	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More

Etape 2 : Création du projet EJB Provider

On a crée une projet EJB dans intellij avec les depedencies suivants :

```xml
<dependencies>
    <dependency>
        <groupId>jakarta.ejb</groupId>
        <artifactId>jakarta.ejb-api</artifactId>
        <version>4.0.1</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>jakarta.servlet</groupId>
        <artifactId>jakarta.servlet-api</artifactId>
        <version>6.1.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.hibernate.orm</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>7.0.0.Alpha3</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jaxb</groupId>
        <artifactId>jaxb-runtime</artifactId>
        <version>4.0.5</version>
    </dependency>
    <dependency>
        <groupId>org.jboss.weld.se</groupId>
        <artifactId>weld-se-core</artifactId>
        <version>6.0.0.Beta1</version>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <version>9.1.0</version>
    </dependency>


    <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.36</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

Dans ce projet il nécessaire d’implementer EJB et servlet et aussi hibernate et mysqlconnector pour la connection au base du données

Et voici la persistence utilisé dans ce projet :

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
https://jakarta.ee/xml/ns/persistence/persistence_3_1.xsd"
             version="3.1">

    <persistence-unit name="Etudiant"
                      transaction-type="JTA">
        <jta-data-source>java:/MySQLDS</jta-data-source>
        <properties>
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL8Dialect" />
            <property name="hibernate.hbm2ddl.auto" value="update" />
            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.format_sql" value="true" />
        </properties>
    </persistence-unit>
</persistence>
```

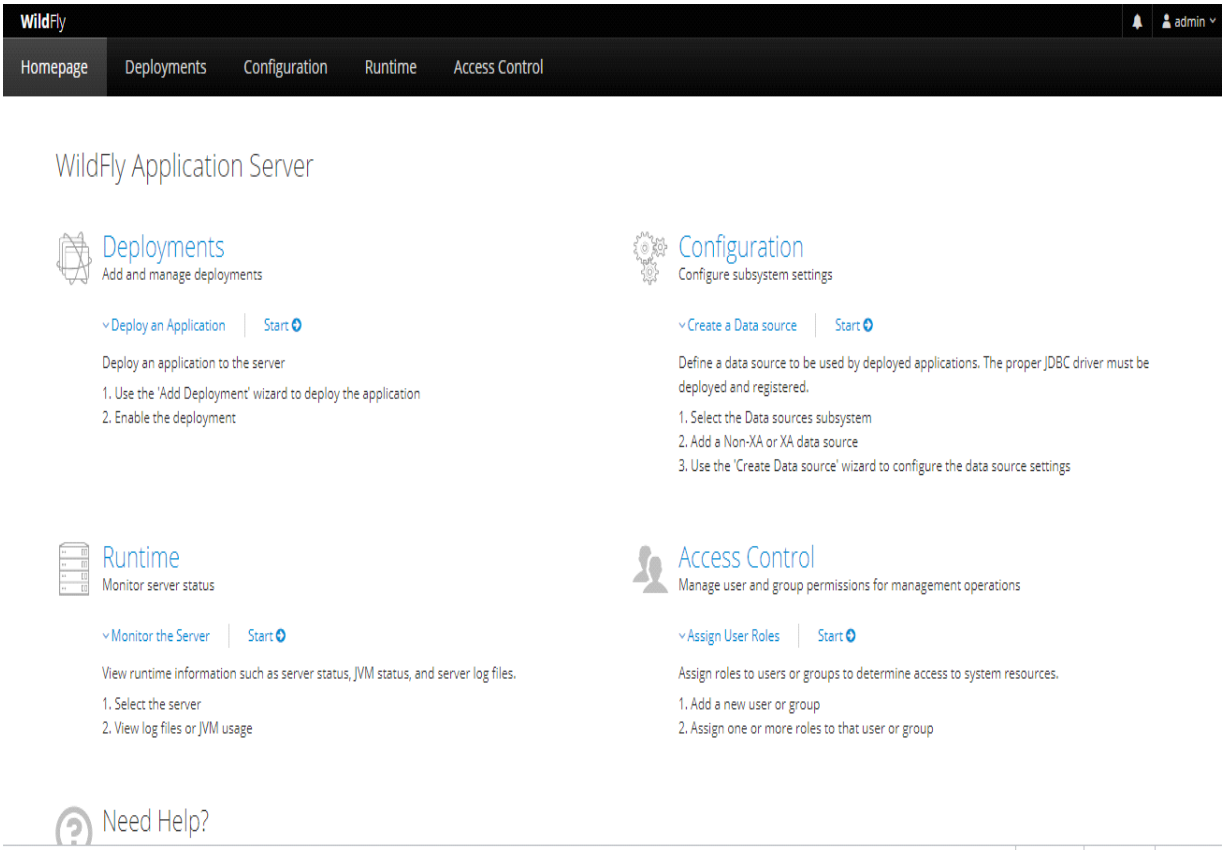
On a MySQLDS comme data source avec le nom du persistence ''Etudiant'' aussi il faut connecter cette datasource dans WildFly pour qu'il connu tout d'abord il faut initilaiser la platforme Admin du wildfly avec cette commande :

```
C:\Servers\wildfly-34.0.1.Final\bin>standalone.bat
```

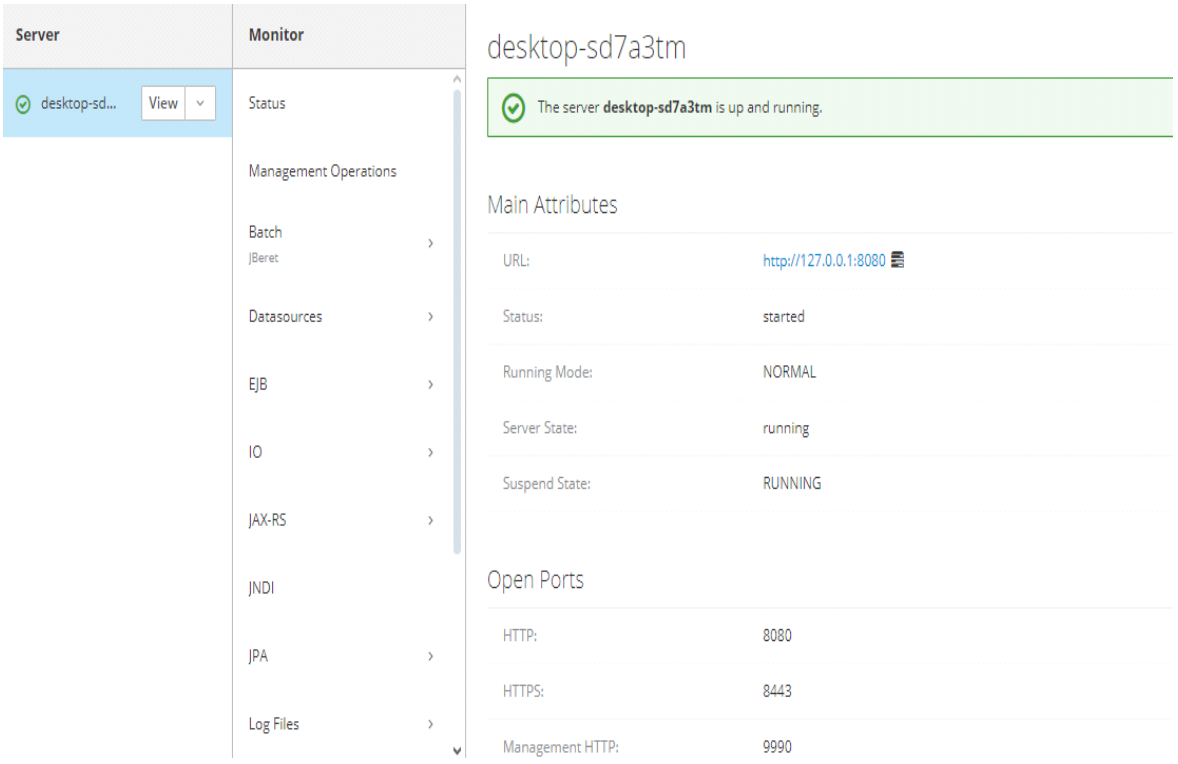
Après on peut accéder la platforme admin en utilisant le lien localhost:9990

```
00:32:50,697 INFO  [org.jboss.as] (Controller Boot Thread) WFLYSRV0051: Admin console listening on http://127.0.0.1:9990
```

Et dans la platforme on saisi l'username et password ici c'est admin pour les deux et on aura l'accès :

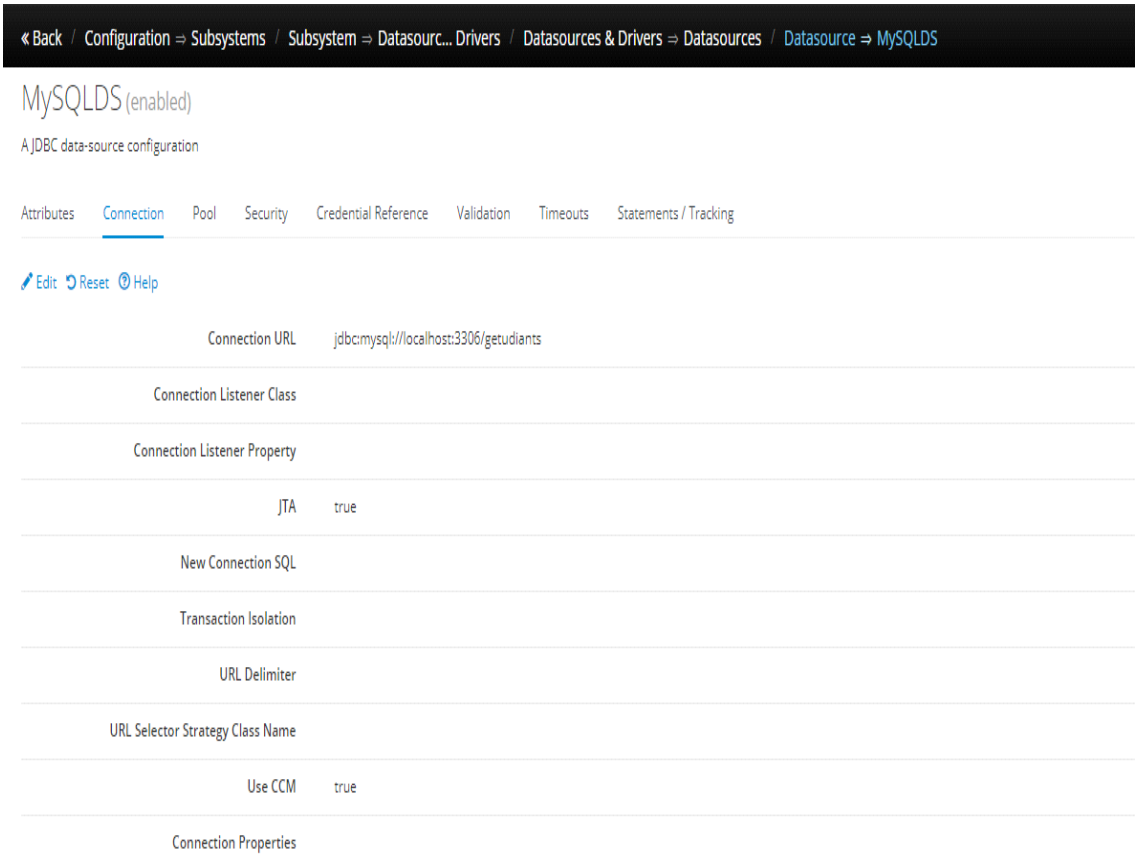


Homepage du wildfly admin control panel:

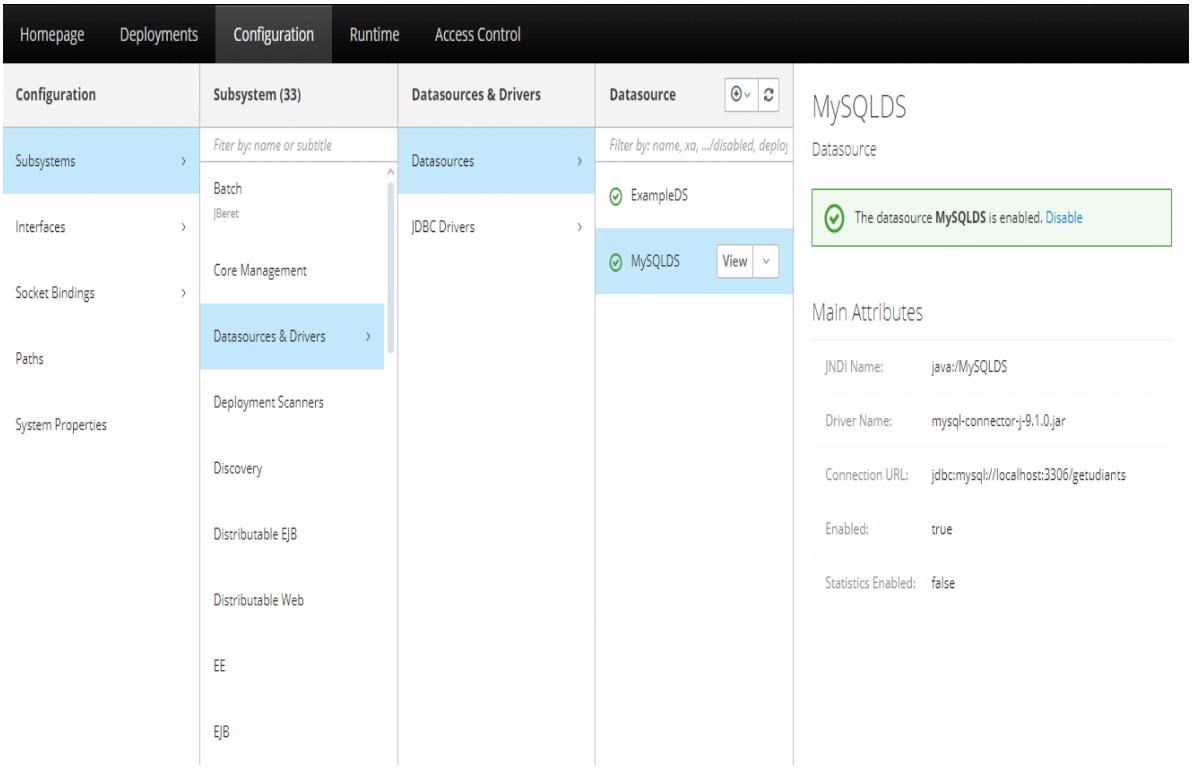


Lorsqu'on a pas des erreurs il doit afficher comme ca

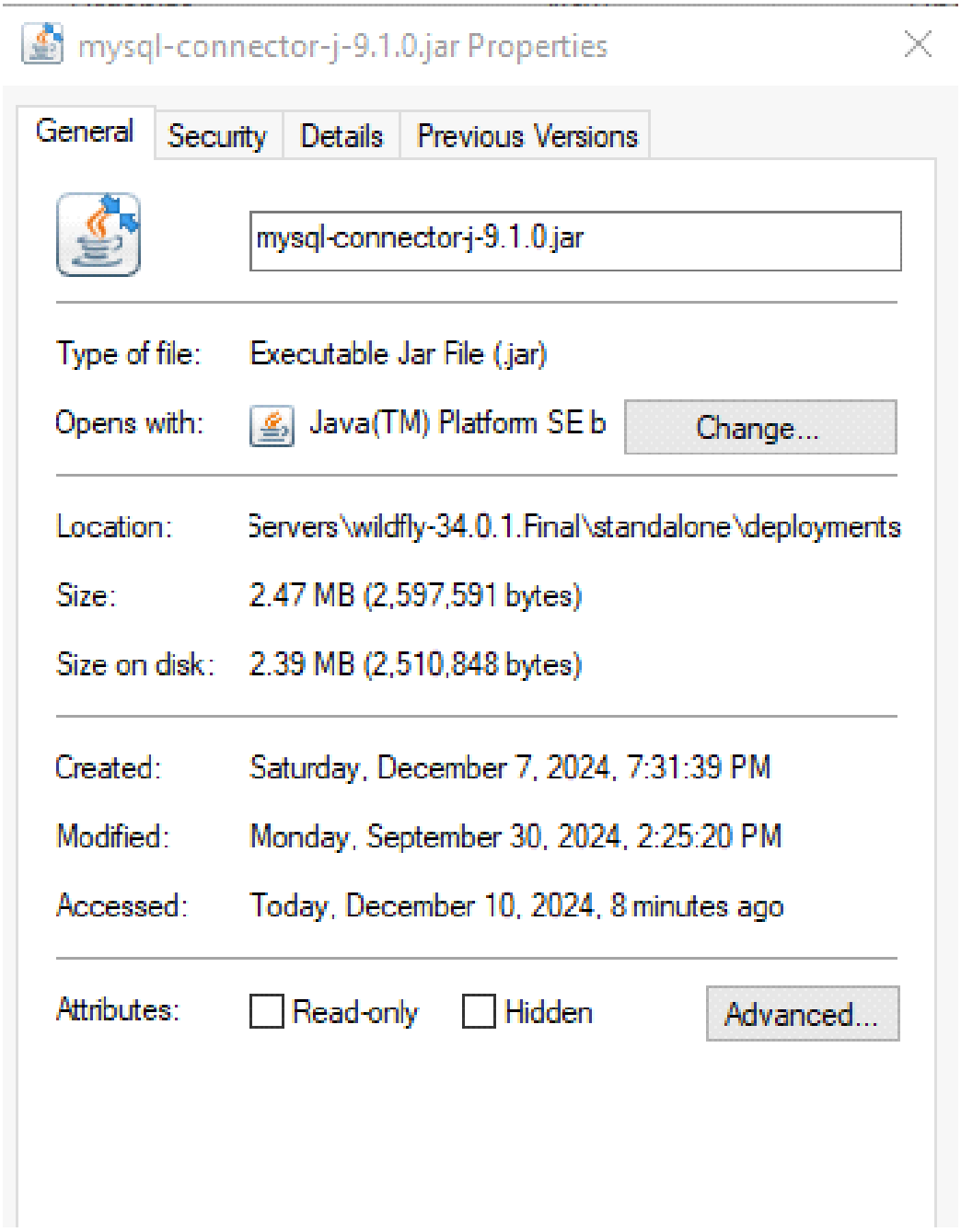
Et on va ajouter le datasource MySQLDS :



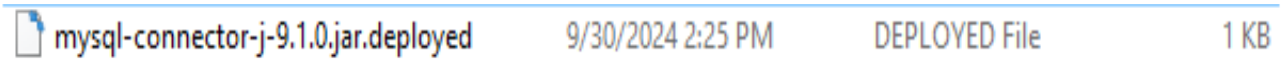
Après l'ajout on peut voir que il existe dans les datasources et on va tester la connection du base du données et il va donner en success :



aussi il faut ajouter une deploiement nécessaire qui est mysqlconnector dans ce dossier :



Lorsqu'il est deployé avec success il doit afficher créer ce fichier :



Etape 3 : creation du classe "Etudiant" :

Voici notre classe etudiant utilisé :

```java
package ma.etudiantejb.ejbcontainer;

import jakarta.persistence.*;
import lombok.*;

import java.io.Serializable;

@Entity
@Table(name = "etudiant")
@Data // Génère getters, setters, toString, equals, hashCode avec Lombok
@NoArgsConstructor // Génère un constructeur sans arguments
@AllArgsConstructor // Génère un constructeur avec tous les arguments
public class Etudiant implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id_etudiant;

    @Column(nullable = false)
    private String nom;

    @Column(nullable = false)
    private String prenom;

    @Column(nullable = false, unique = true)
    private String cne;

    private String adresse;

    private String niveau;
}
```

on a utilisé des annotations JPA et aussi des annotations du lombok pour simplifier l'appel des getters,setters,constructors

Etape 4 : creation du classe du Type Session Bean

Tout d’abord on va créer l’interface remote qui comme ca :

```java
package ma.etudiantejb.ejbcontainer;

import java.util.List;

import jakarta.ejb.Remote;

@Remote
public interface EtudiantServiceRemote {
    void addEtudiant(Etudiant etudiant);
    void updateEtudiant(Etudiant etudiant);
    void deleteEtudiant(int id);
    Etudiant findEtudiantById(int id);
    List<Etudiant> findAllEtudiants();
}
```

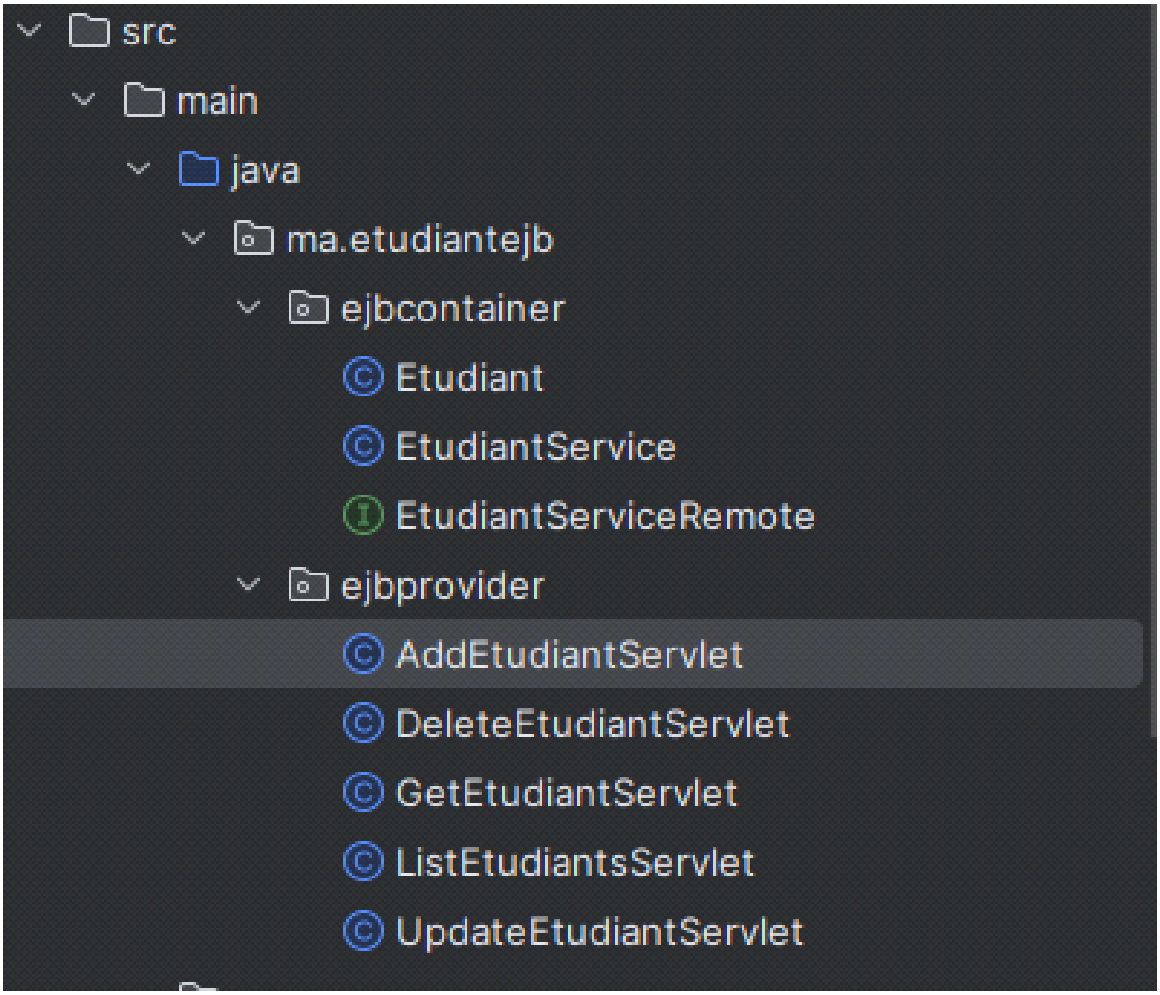
on appelé tous les méthodes cruds qu’on va utiliser et voici le service qui va l’implémenter :

```java
package ma.etudiantejb.ejbcontainer;


import jakarta.ejb.Stateless;
import jakarta.persistence.EntityManager;
import jakarta.persistence.PersistenceContext;

import java.util.List;

@Stateless
public class EtudiantService implements EtudiantServiceRemote {
    @PersistenceContext(unitName = "Etudiant")
    private EntityManager entityManager;

    @Override
    public void addEtudiant(Etudiant etudiant) {
        entityManager.persist(etudiant);
    }

    @Override
    public void updateEtudiant(Etudiant etudiant) {
        entityManager.merge(etudiant);
    }

    @Override
    public void deleteEtudiant(int id) {
        Etudiant etudiant = entityManager.find(Etudiant.class, id);
        if (etudiant != null) {
            entityManager.remove(etudiant);
        }
    }

    @Override
    public Etudiant findEtudiantById(int id) {
        return entityManager.find(Etudiant.class, id);
    }

    @Override
    public List<Etudiant> findAllEtudiants() {
        return entityManager.createQuery("SELECT e FROM Etudiant e", Etudiant.class).getResultList();
    }
}
```

dans cette partie on a definir tous les méthodes CRUDs qui vont manipuler la base du données connecté ‘’getudiants’’ en utilisant EntityManager aussi on a appelé l’annotation @Stateless qui est nécessaire pour le projet EJB

Etape 5 : Création du Projet Web pour consommer l’EJB

Tout d’abord dans le provider il faut créer les servlets nécessaires pour réaliser les taches CRUDs différents :

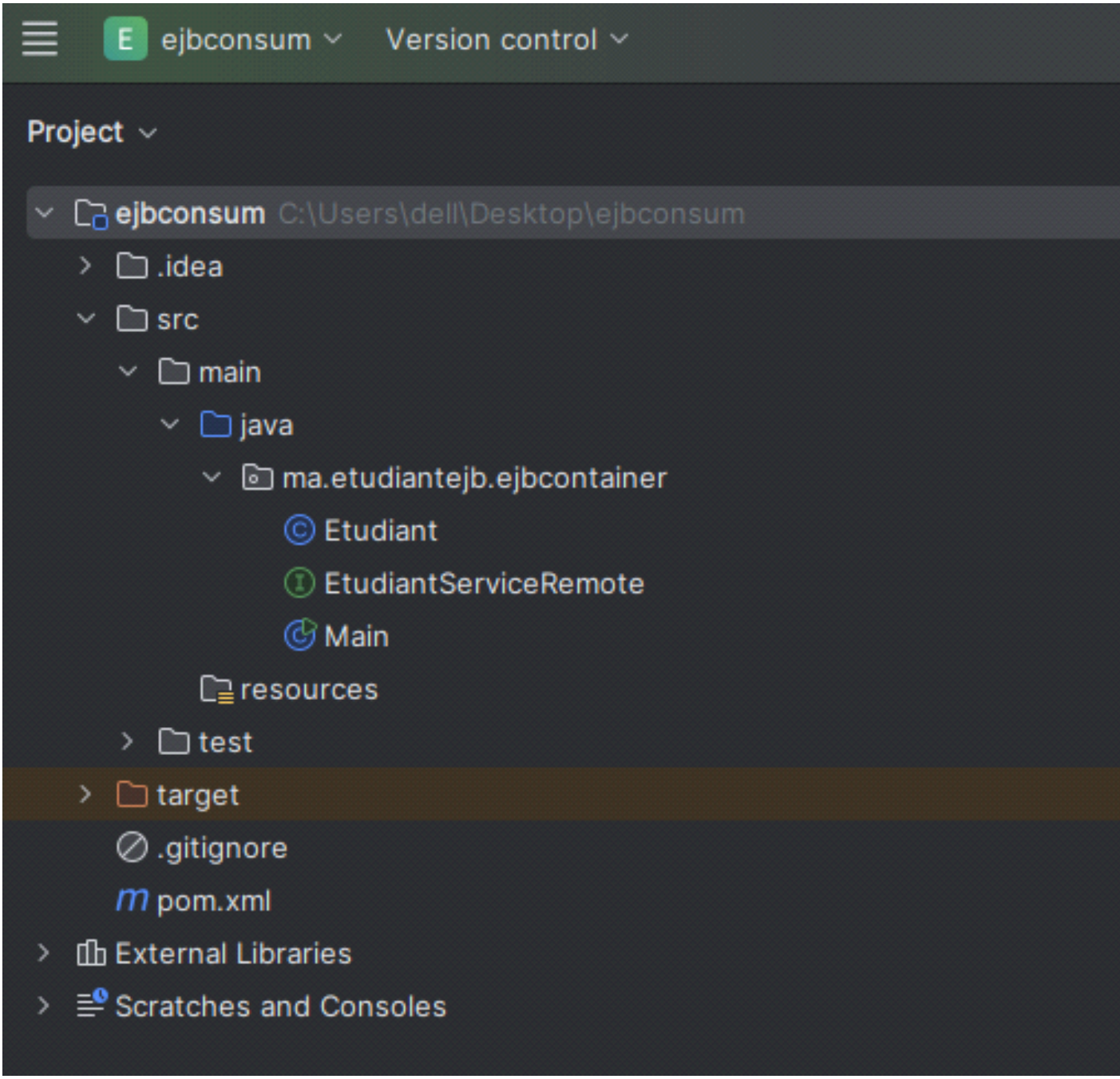


Voici une exemple du servlet comme AddEtudiantServlet :

```java
package ma.etudiantejb.ejbprovider;


import jakarta.ejb.EJB;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import ma.etudiantejb.ejbcontainer.Etudiant;
import ma.etudiantejb.ejbcontainer.EtudiantServiceRemote;


import java.io.IOException;

@WebServlet(name = "AddEtudiantServlet", value = "/add-etudiant")
public class AddEtudiantServlet extends HttpServlet {

    @EJB(lookup = "java:global/etudiantejb-1.0-
SNAPSHOT/EtudiantService!ma.etudiantejb.ejbcontainer.EtudiantServiceRemote")
    private EtudiantServiceRemote etudiantService;

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
        String nom = request.getParameter("nom");
        String prenom = request.getParameter("prenom");
        String cne = request.getParameter("cne");
        String adresse = request.getParameter("adresse");
        String niveau = request.getParameter("niveau");

        Etudiant etudiant = new Etudiant();
        etudiant.setNom("BAHASSOU");
        etudiant.setPrenom("Mohamed Amine");
        etudiant.setCne("J100033932");
        etudiant.setAdresse("Rabat");
        etudiant.setNiveau("CI");
        etudiantService.addEtudiant(etudiant);

        response.getWriter().write("Étudiant ajouté avec succès !");
    }
}
```

dans cette servlet on a connecter avec service EJB avec une méthode post pour l'ajoute si on utilise Postman ou bien une autre maniére pour tester, et aussi il va créer une utilisateur par défaut qui est Dupont Jean pour remplir la BDD initialement

pour la structure du projet web voici la structure que j'ai fait :

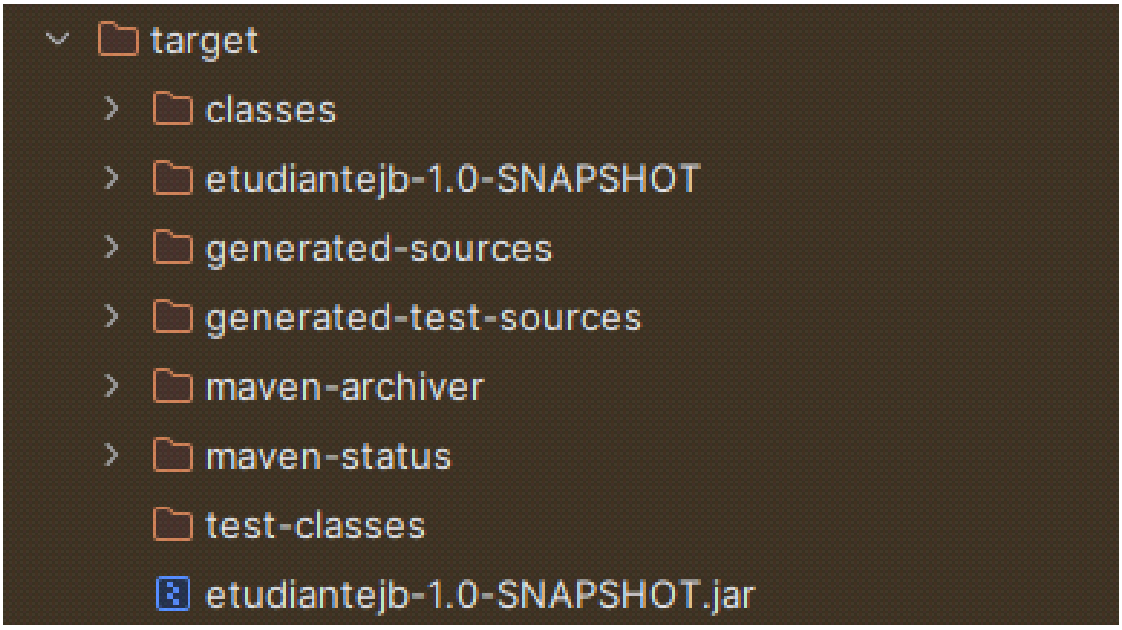


J’appelle juste la classe Remote et Entity d’Etudiant et voici le Main utilisé pour tester les fonctionnalités :

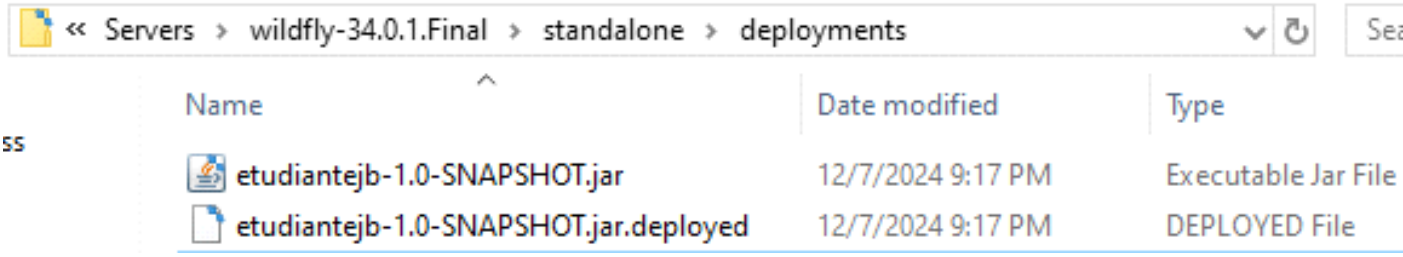
```java
package ma.etudiantejb.ejbcontainer;

import jakarta.ejb.EJB;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import java.util.List;
import java.util.Properties;

public class Main {
    public static void main(String[] args) {
        Context context = null;
        EtudiantServiceRemote service = null;

        try {
            // Configuration JNDI
            Properties jndiProperties = new Properties();
            jndiProperties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.wildfly.naming.client.WildFlyInitialContextFactory");
            jndiProperties.put(Context.PROVIDER_URL, "http-remoting://localhost:8080");
            jndiProperties.put("jboss.naming.client.ejb.context", true);

            context = new InitialContext(jndiProperties);

            // Lookup du service EJB
            service = (EtudiantServiceRemote) context.lookup(
                    "ejb:/etudiantejb-1.0-
SNAPSHOT/EtudiantService!ma.etudiantejb.ejbcontainer.EtudiantServiceRemote"
            );

            // Ajouter un étudiant
            Etudiant etudiant = new Etudiant();
            etudiant.setNom("BAHASSOU");
            etudiant.setPrenom("Mohamed Amine");
            etudiant.setCne("J100033932");
            etudiant.setAdresse("Rabat");
            etudiant.setNiveau("CI");
            service.addEtudiant(etudiant);

            // Récupérer un étudiant
            Etudiant fetched = service.findEtudiantById(1);
            if (fetched ≠ null) {
                System.out.println("Étudiant récupéré : " + fetched);

                // Mettre à jour un étudiant
                fetched.setNom("Updated Name");
                service.updateEtudiant(fetched);
            } else {
                System.out.println("Étudiant avec l'ID 1 non trouvé.");
            }

            // Lister tous les étudiants
            List<Etudiant> etudiants = service.findAllEtudiants();
            System.out.println("Liste des étudiants : " + etudiants);

            // Supprimer un étudiant
//            service.deleteEtudiant(1);
        } catch (NamingException e) {
            System.err.println("Erreur de JNDI: " + e.getMessage());
        } catch (Exception e) {
            System.err.println("Erreur: " + e.getMessage());
        } finally {
            // Cleanup context
            try {
                if (context ≠ null) {
                    context.close();
                }
            } catch (NamingException e) {
                System.err.println("Erreur lors de la fermeture du context: " + e.getMessage());
            }
        }
    }
}
```

dans cette parite il va appeler l'EJB du projet provider qui est cité dans le context lookup donc il va utilisé le datasource qui est MySQLDS, mais tout d'abord il faut deployer le projet provider dans WildFly admin control panel pour qu'il doit utiliser :

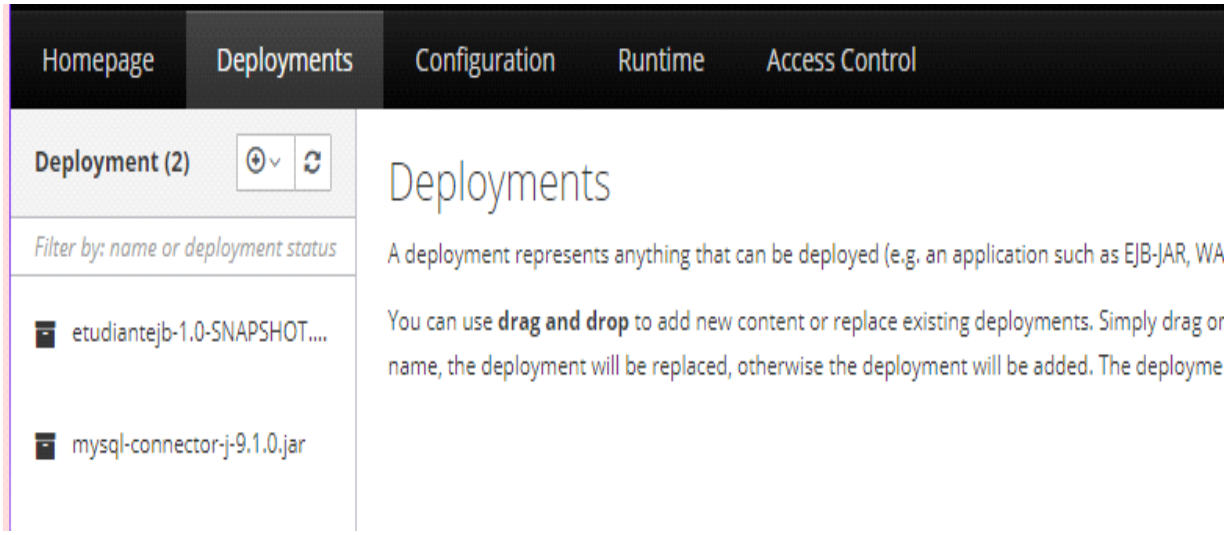
après le build du maven on va voir le fichier target pour avoir le jar :



Ce jar on va le copier coller dans le dossier du deployments du wildfly



Et on peut voir que il est déployer avec succes :



Maintenant pour tester notre web project on va juste lancer le server Wildfly dans project provider, aussi il est important du wildfly d'etre aussi commencé dans le cmd avec ''standalone.bat''

```
"C:\Program Files\Java\jdk-21\bin\java.exe" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr
Dec 10, 2024 12:55:35 AM org.jboss.logmanager.JBossLoggerFinder getLogger
ERROR: The LogManager accessed before the "java.util.logging.manager" system property was set to "org.jbos
Connected to server
```

Et on va tester quelque méthodes dans notre web projet pour vérifier la fonctionnement

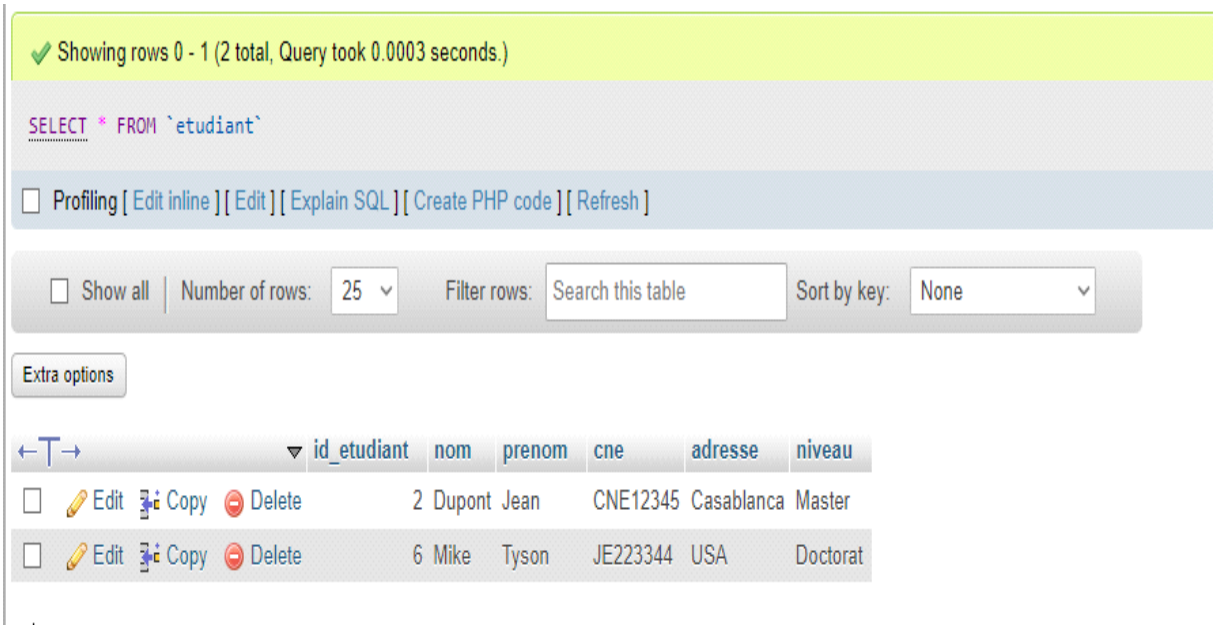
Pour l'ajout :

```java
// Ajouter un étudiant
Etudiant etudiant = new Etudiant();
etudiant.setNom("Mike");
etudiant.setPrenom("Tyson");
etudiant.setCne("JE223344");
etudiant.setAdresse("USA");
etudiant.setNiveau("Doctorat");
service.addEtudiant(etudiant);
```

On voir dans le serveur wildfly dans cmd :

```
01:02:24,027 INFO  [stdout] (default task-2) Hibernate:
01:02:24,028 INFO  [stdout] (default task-2)     insert
01:02:24,029 INFO  [stdout] (default task-2)     into
01:02:24,029 INFO  [stdout] (default task-2)         etudiant
01:02:24,030 INFO  [stdout] (default task-2)         (adresse, cne, niveau, nom, prenom)
01:02:24,030 INFO  [stdout] (default task-2)     values
01:02:24,031 INFO  [stdout] (default task-2)         (?, ?, ?, ?, ?)
```

Et dans la BDD :



Pour récuperer et modifier :

```java
//            Récupérer un étudiant
        Etudiant fetched = service.findEtudiantById(3);
        if (fetched != null) {
            System.out.println("Étudiant récupéré : " + fe

            // Mettre à jour un étudiant
            fetched.setNom("Updated Name");
            service.updateEtudiant(fetched);
        } else {
            System.out.println("Étudiant avec l'ID 1 non f
        }
```

Voici la récuperation :

```
Étudiant récupéré : Etudiant(id_etudiant=2, nom=Dupont, prenom=Jean, cne=CNE12345, adresse=Casa
```

Et voici la modificaiton fait :

←T→			id_etudiant	nom	prenom	cne	adr
☐	Edit	Copy Delete	2	Updated Name	Jean	CNE12345	Cas
☐	Edit	Copy Delete	6	Mike	Tyson	JE223344	USA

Pour lister tous :

```java
    // Lister tous les étudiants
    List<Etudiant> etudiants = service.findAllEtudiants();
    System.out.println("Liste des étudiants : " + etudiants)
```

Voici la résultat obtenu :

```
Liste des étudiants : [Etudiant(id_etudiant=2, nom=Updated Name, prenom=Jean, cne=CNE12345, adresse=Casablanca, niveau=M
```

Il affiche encore les infos mais il est très longue pour capturer tous dans le terminal

Pour supprimer :

```
// Supprimer un étudiant
service.deleteEtudiant( id: 2);
```

Base du données après suppression :

Donc tous les méthodes CRUD sont disponibles dans notre web projet et il fonctionne bien on peut implémenter ces méthodes CRUDs plus encore dans une interface web qui est simple