

Programmation EJB 3 JPA Oeuvres

Objectifs de ce TP

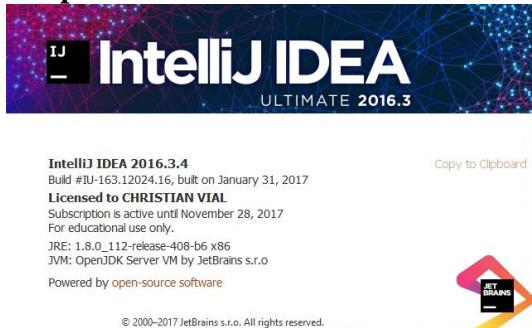
Ce tp vous montre l'utilisation des Entity Beans dans le cadre d'un projet JPA. Vous mettrez en œuvre la fonctionnalité de mapping objet-relationnel avec la base MYSQL. Vous apprêhenderez l'approche POJO (Plain Object Java Object) en construisant une classe Java qui sera mappée dans la base de données.

Présentation de l'application

Environnement

Cette application est développée sous

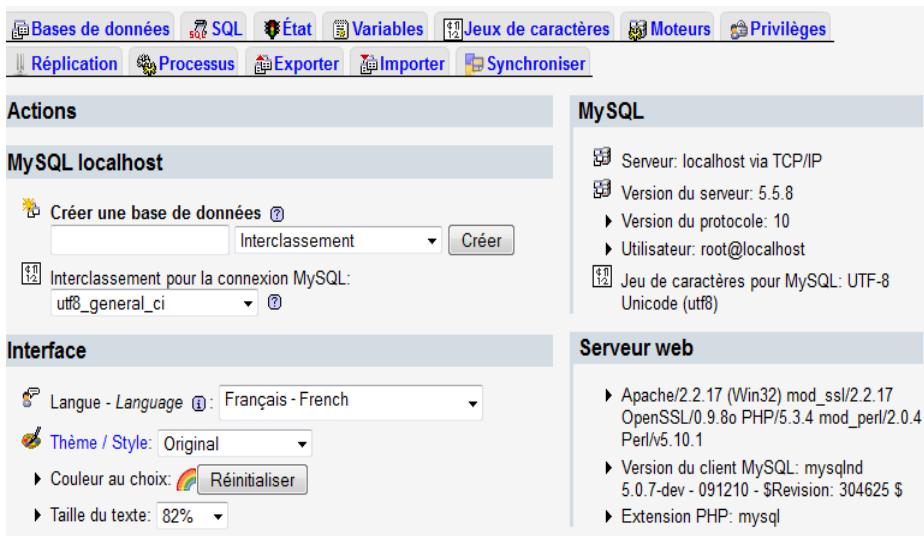
➤ Eclipse Mars



➤ Tomcat 8



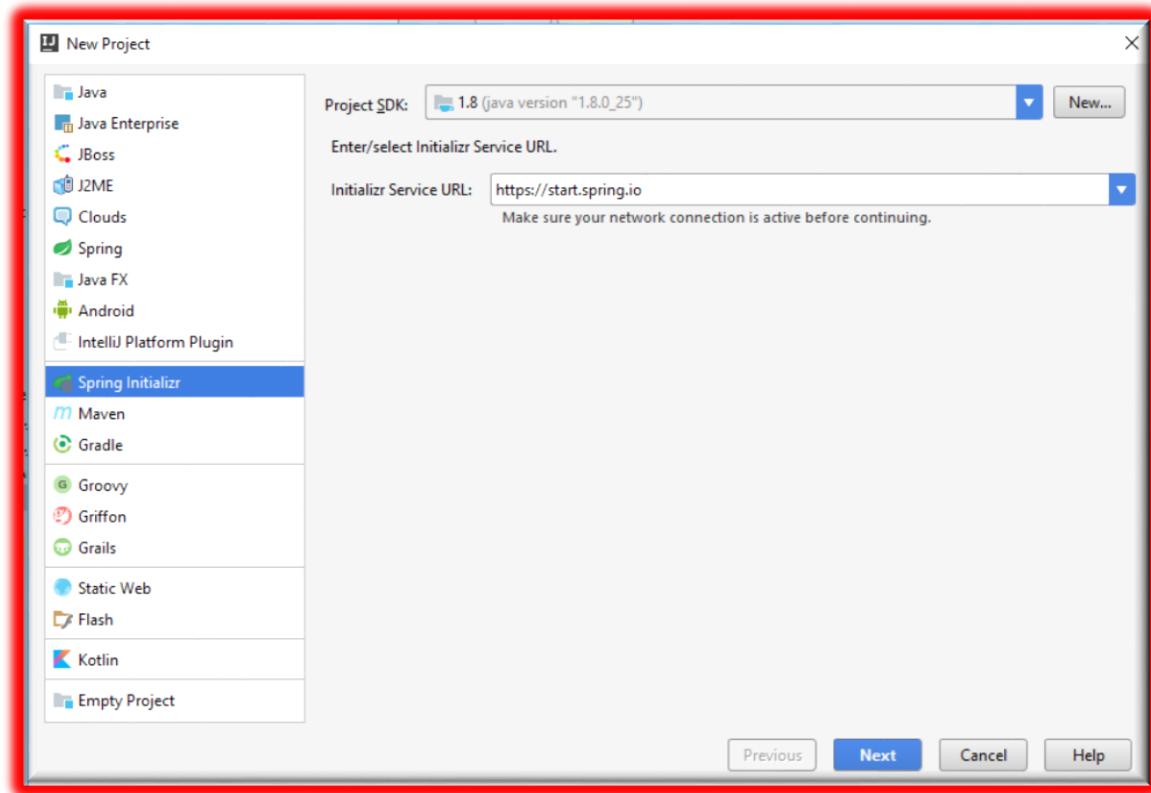
➤ Base de données MYSQL.



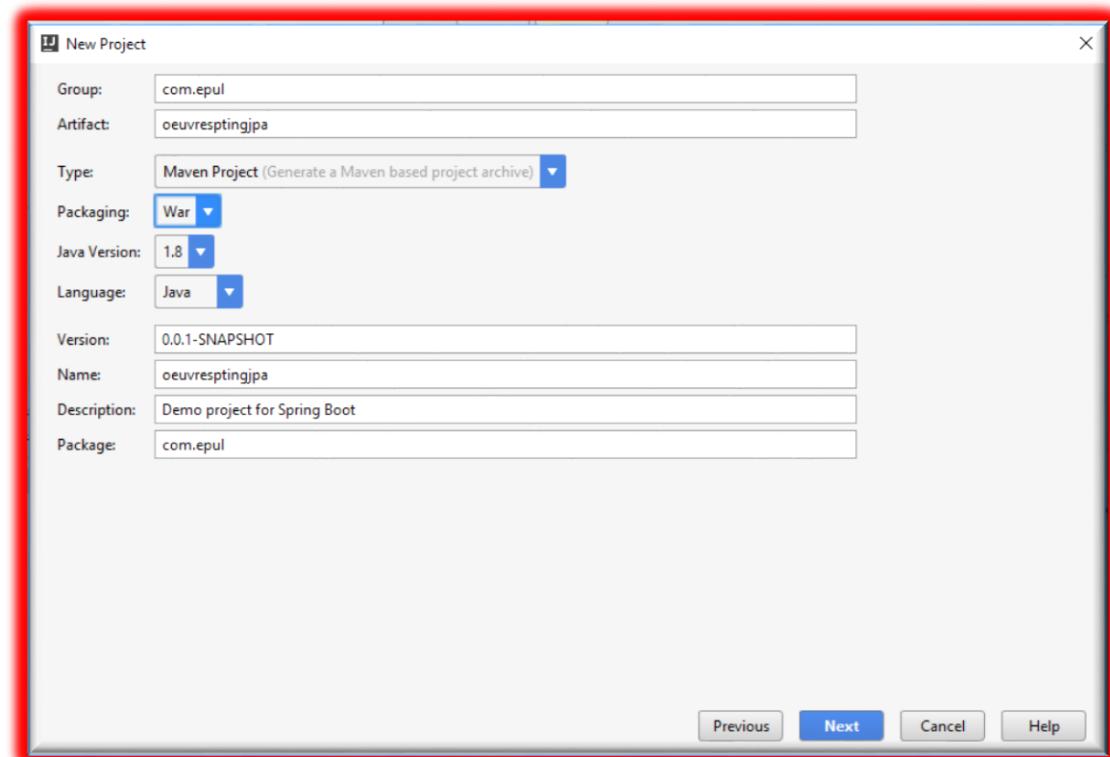
:

Développement de l'application

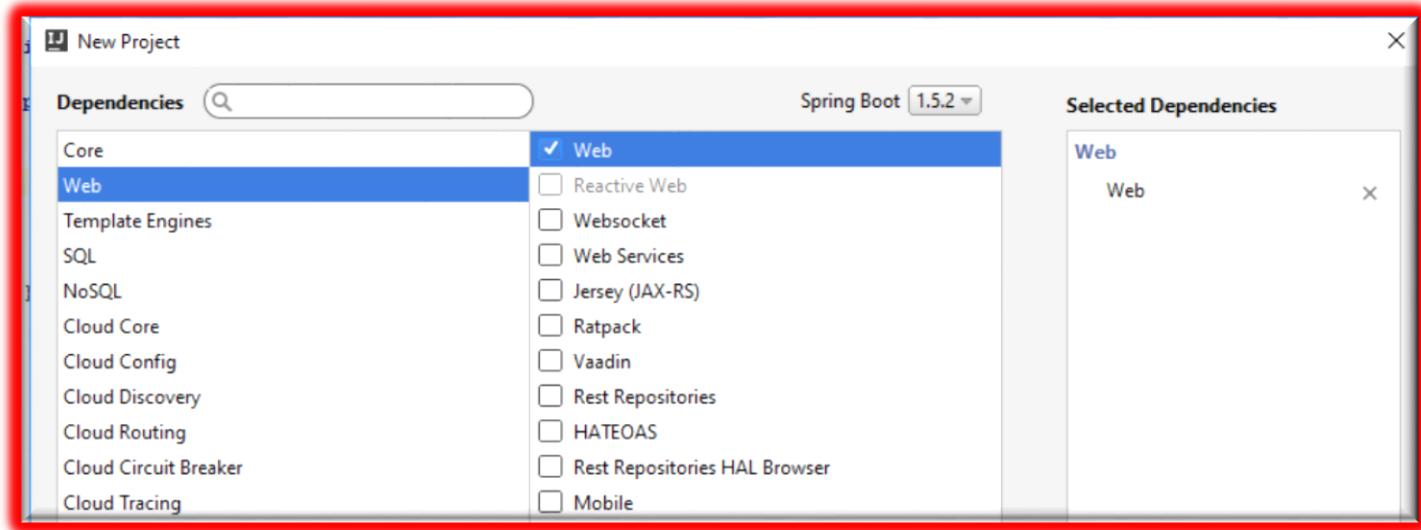
Choisissez le type d'application sous IntelliJ : **Spring JPA**



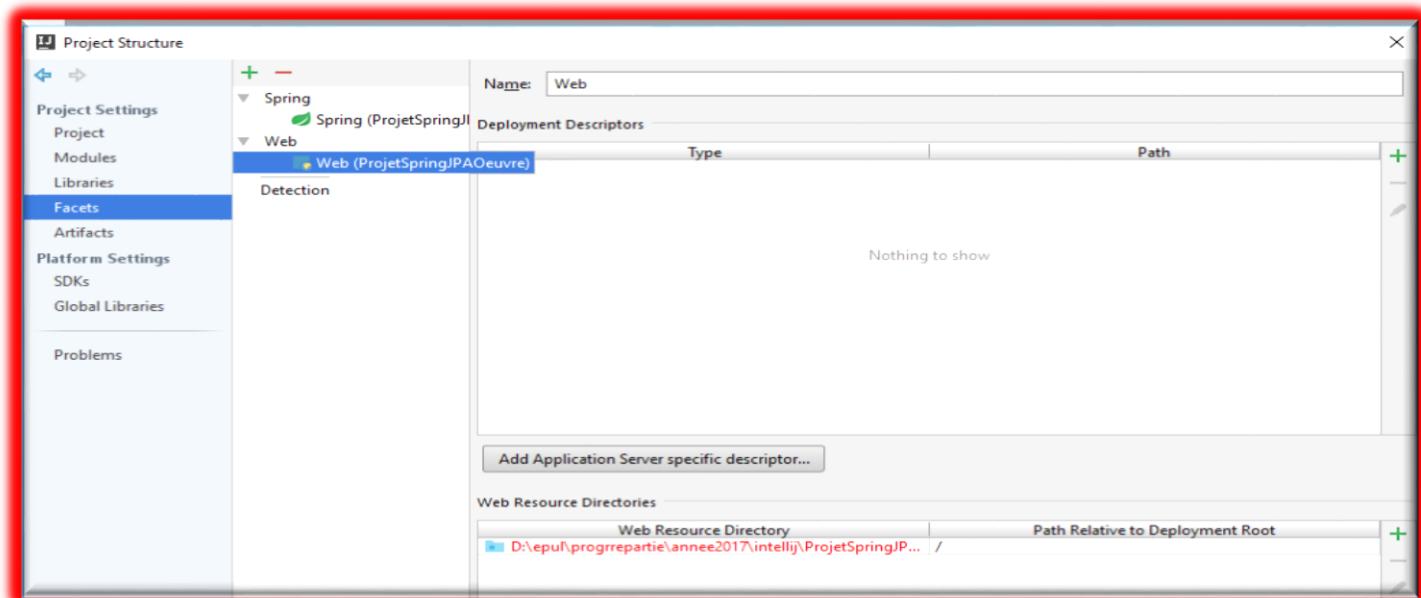
On configure Spring



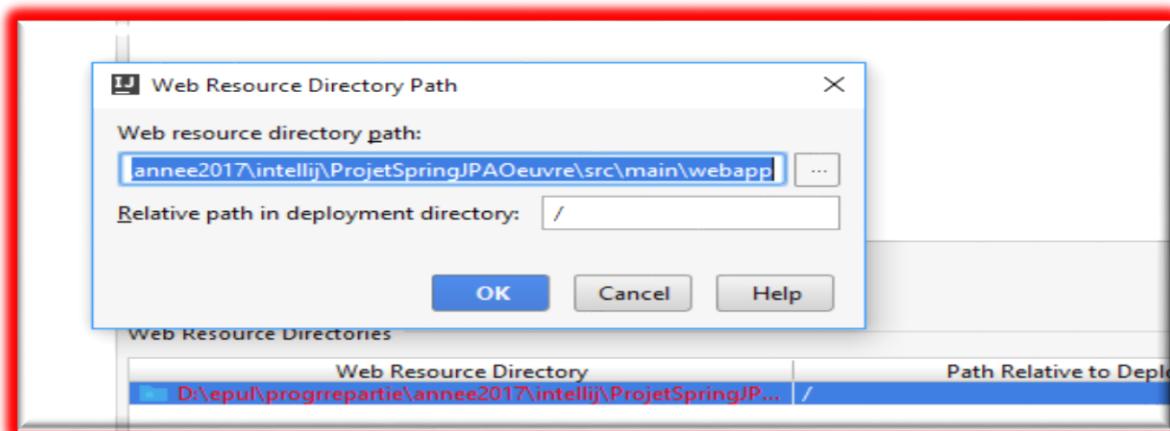
Projet Web MVC

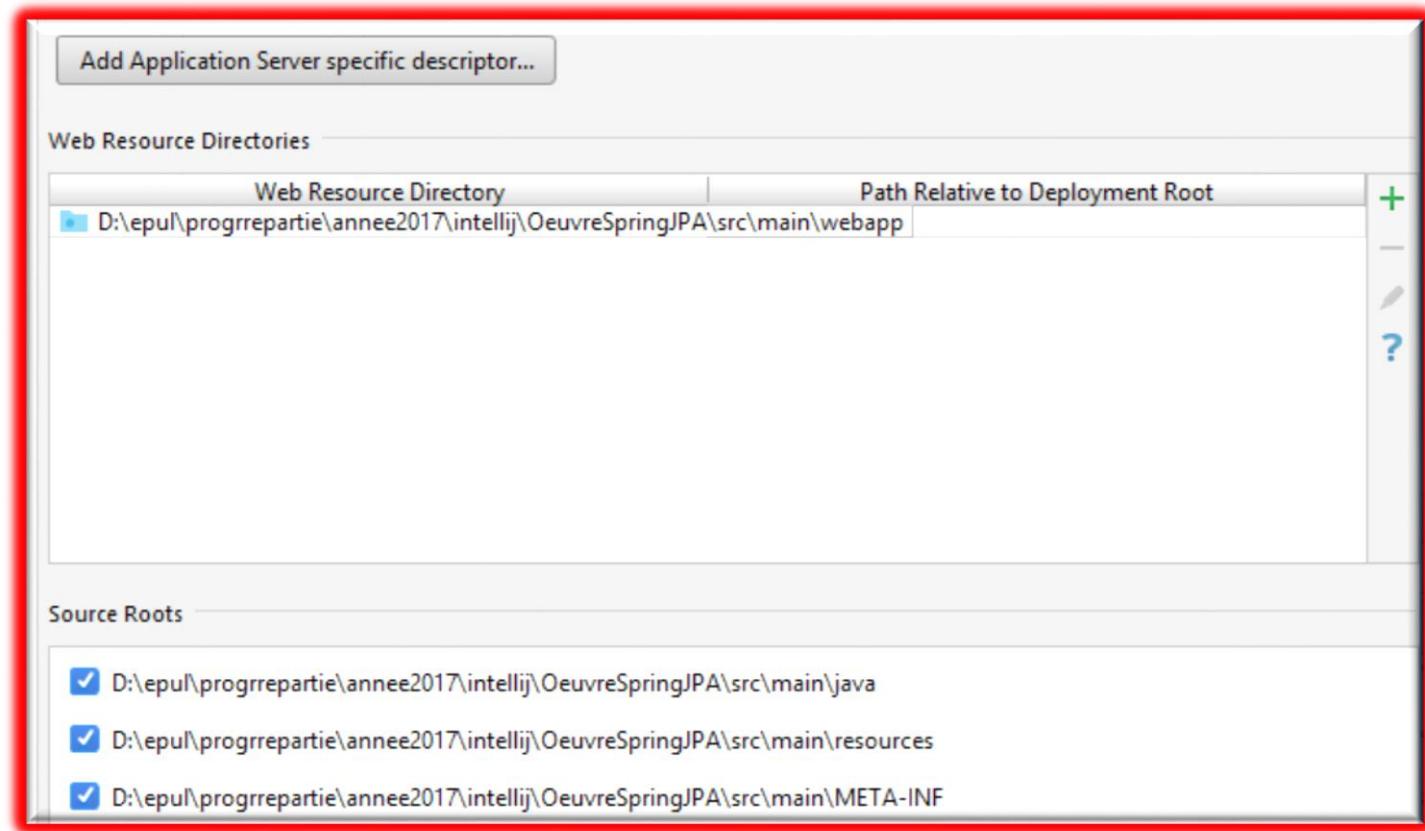


On ajoute le répertoire webapp de façon manuelle

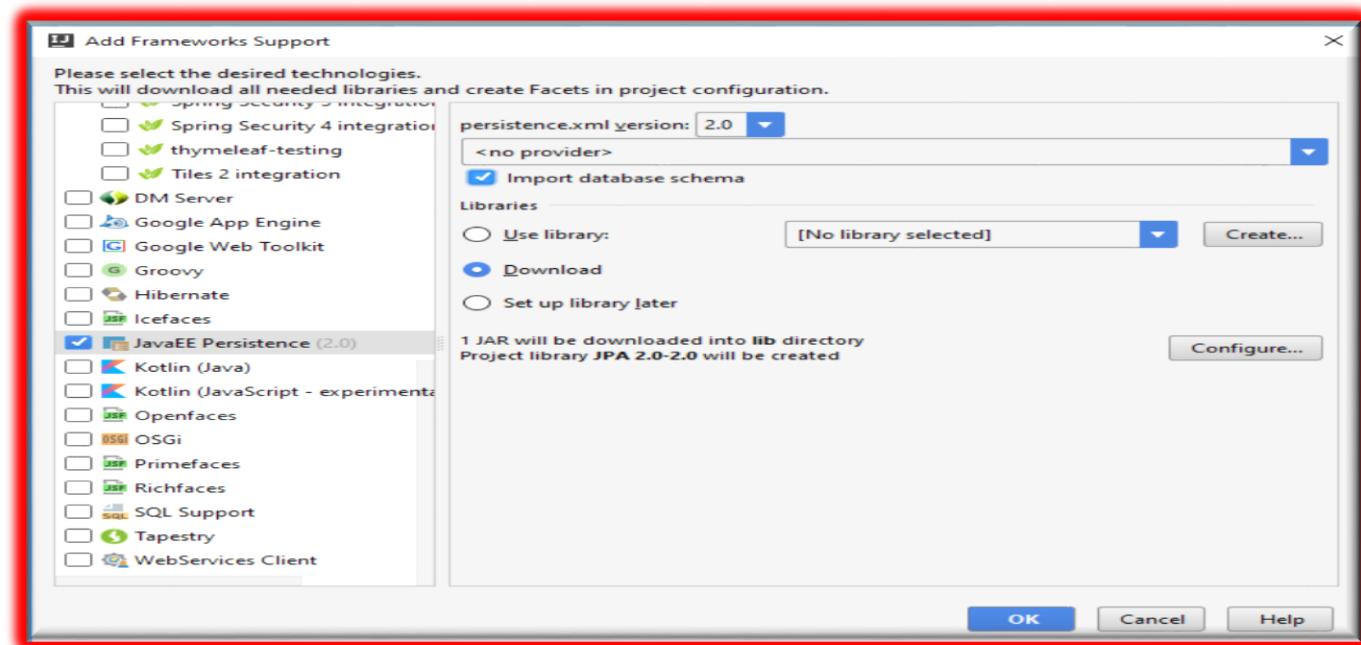


On l'ajoute





On ajoute le frameworks Java Persistence (hibernate)



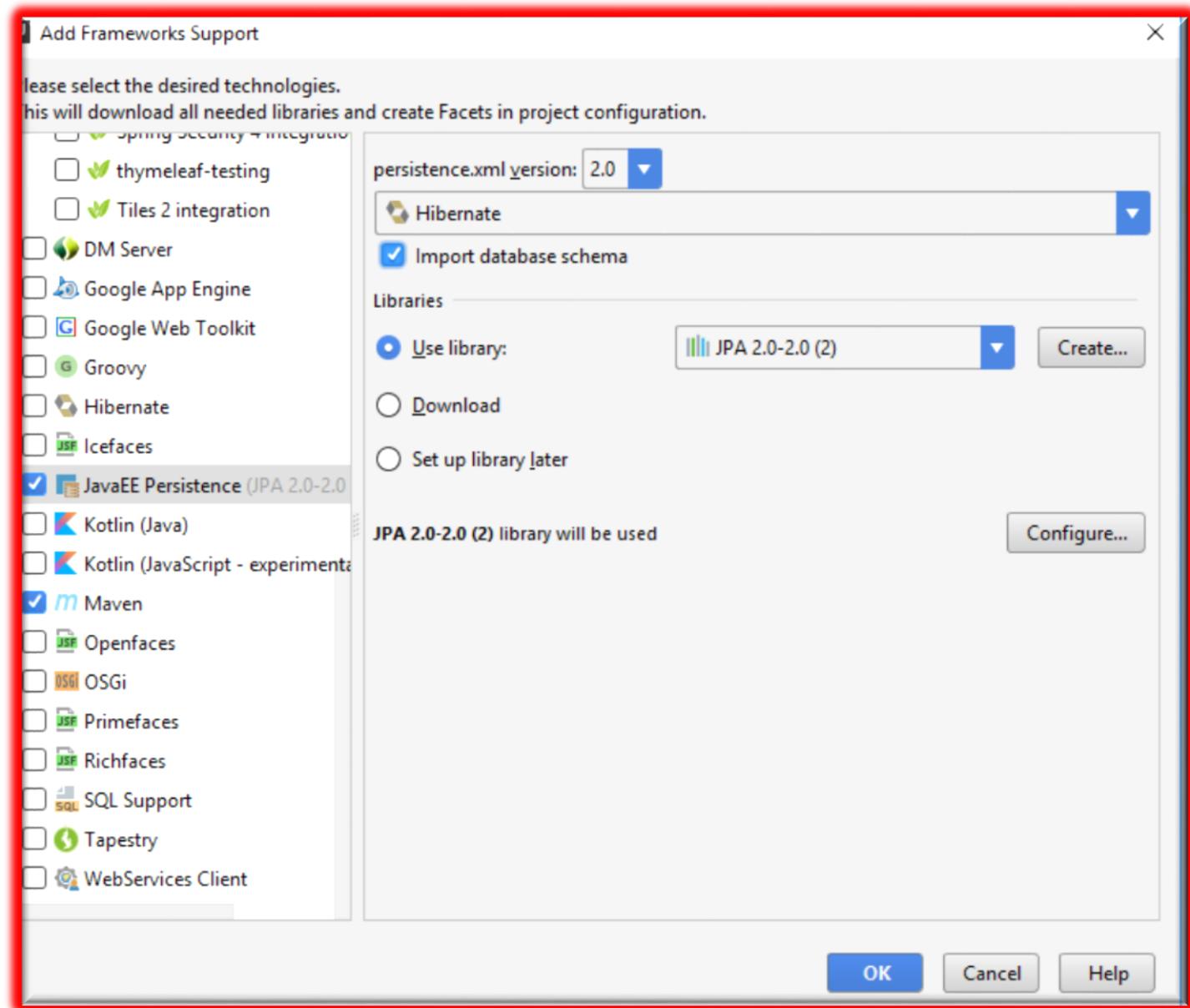
Si vous n'avez pas encore la bibliothèque Hibernate, il faut la télécharger.

Création des classes correspondant aux tables (Mapping)

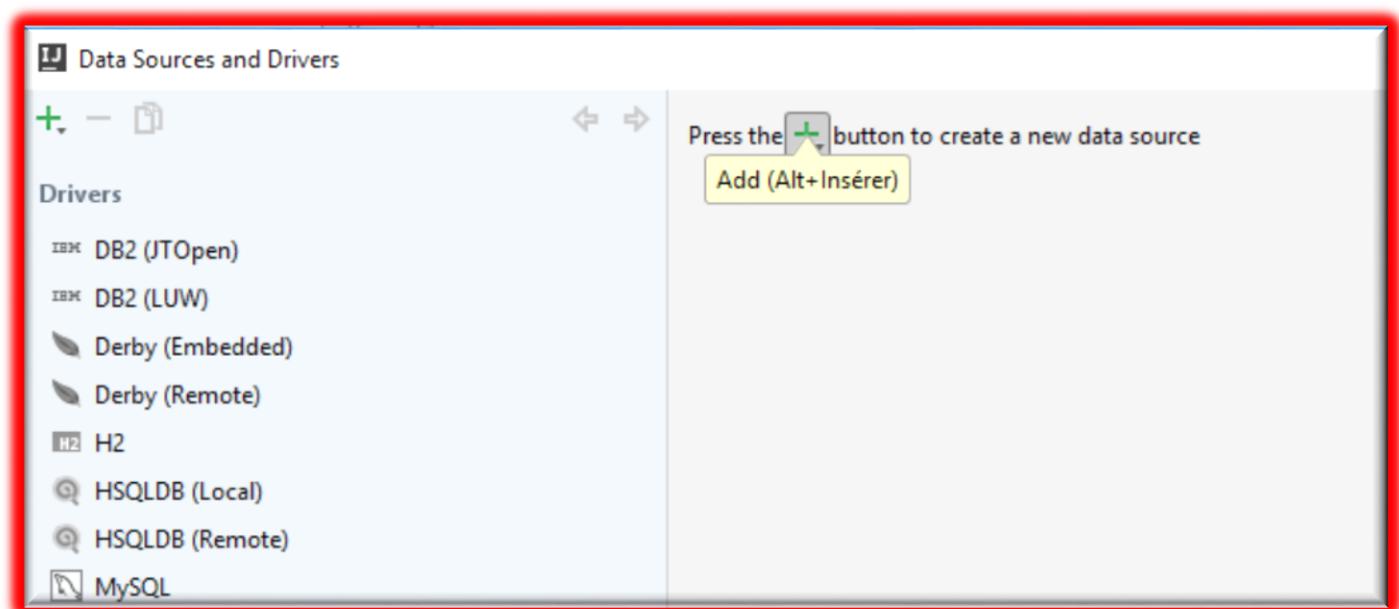
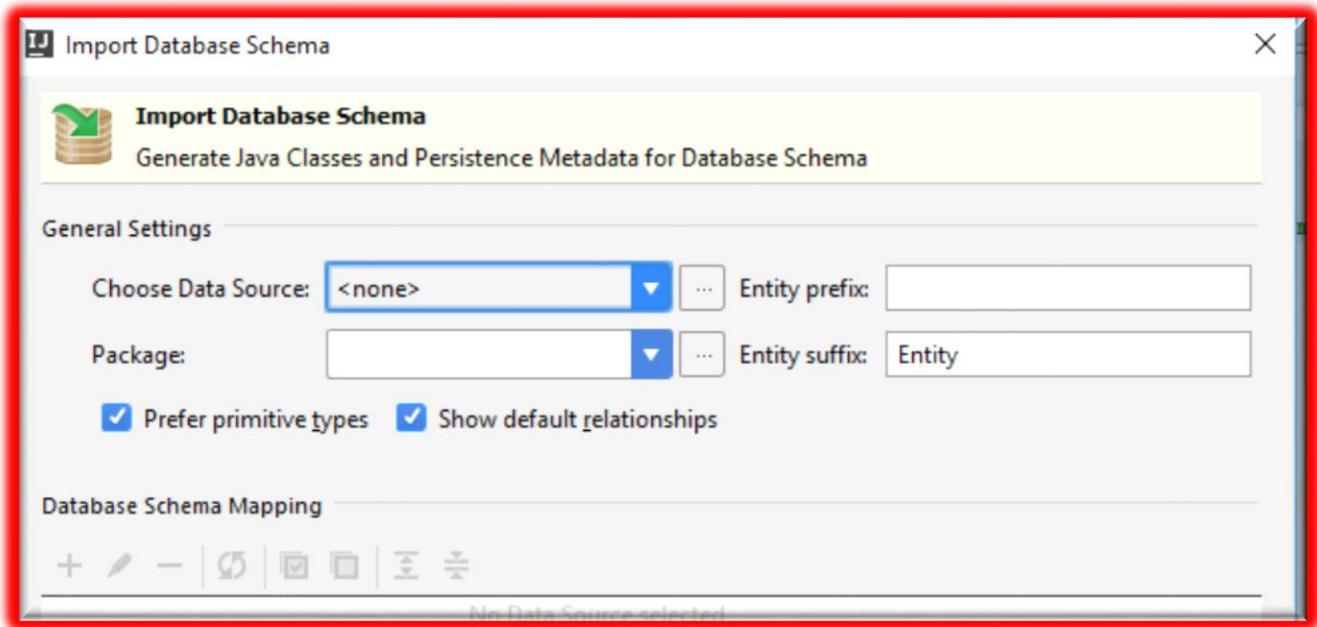
IntelliJ nous offre un outil puissant qui va nous permettre de générer les classes à partir de notre base de données en utilisant notre connexion MySql définie ci-dessus.

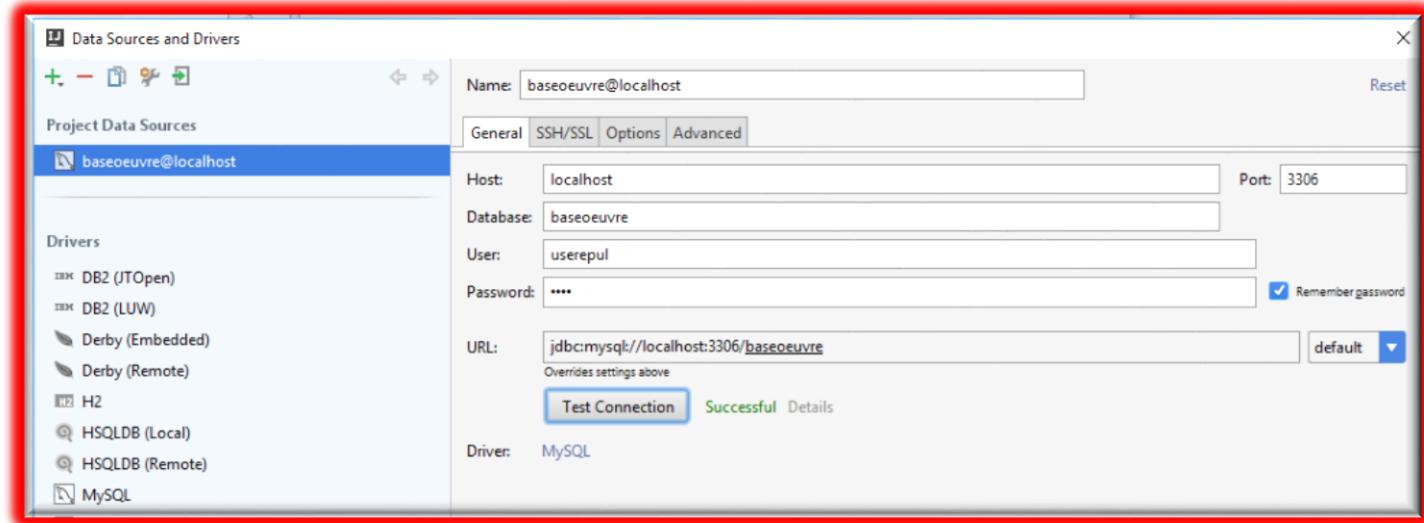
Génération des classes

On sélectionne :

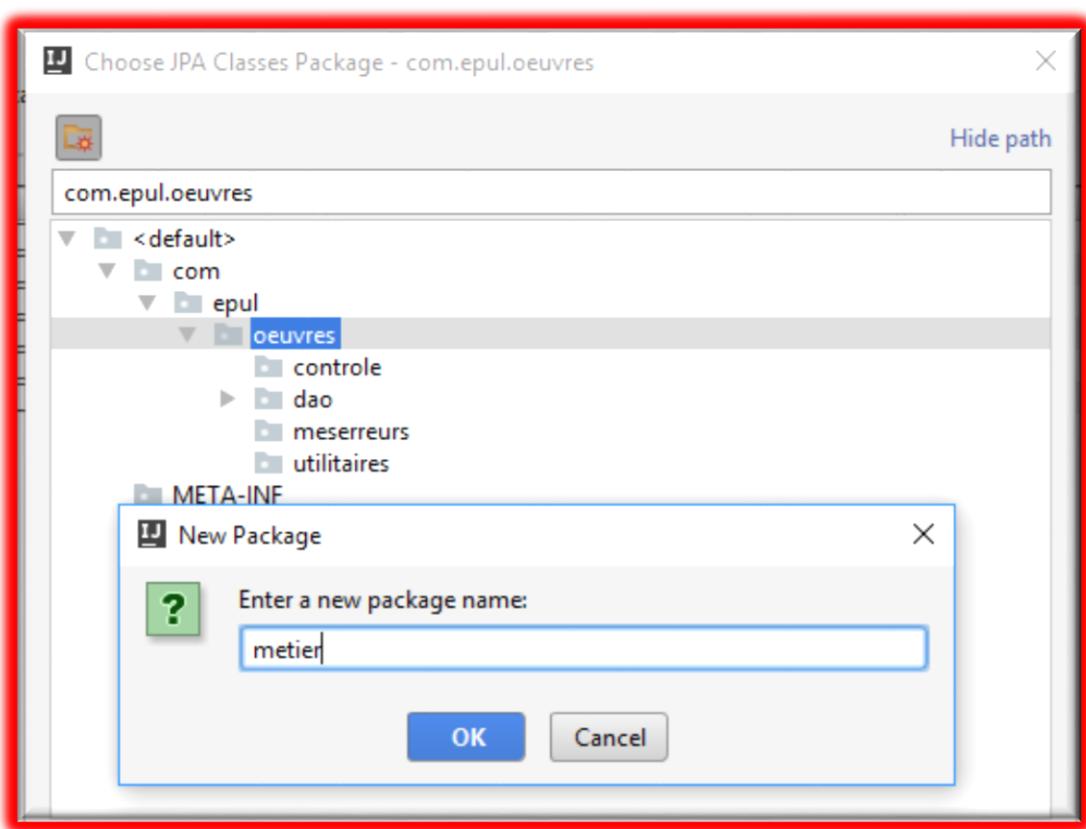


N'oubliez pas de cocher Import database schema pour obtenir l'écran suivant qui permet de générer les classes métier à partir de la base de données.

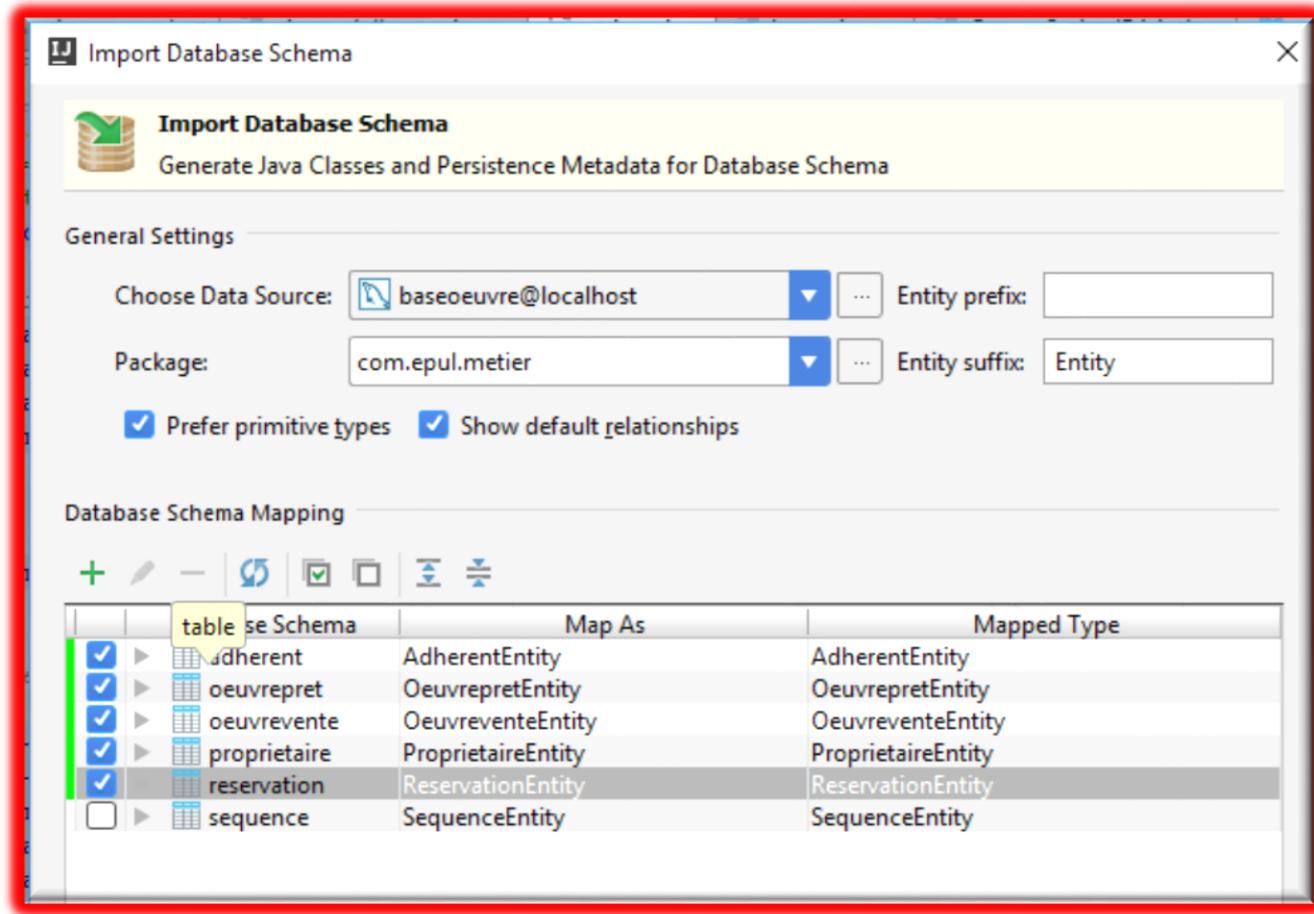




On choisit un répertoire pour stocker les classes métier :



On peut alors commencer la génération

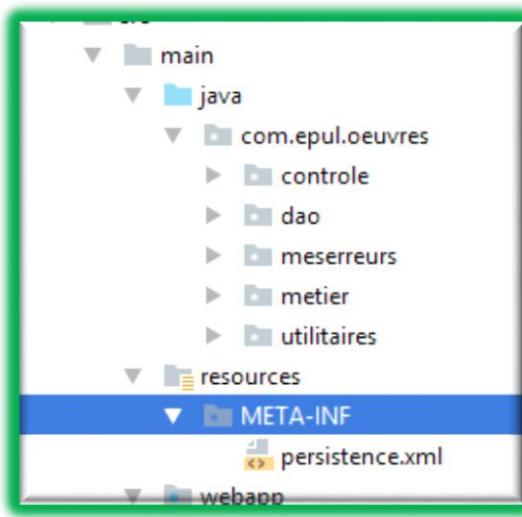


Fichiers générés

Le système vous a créé

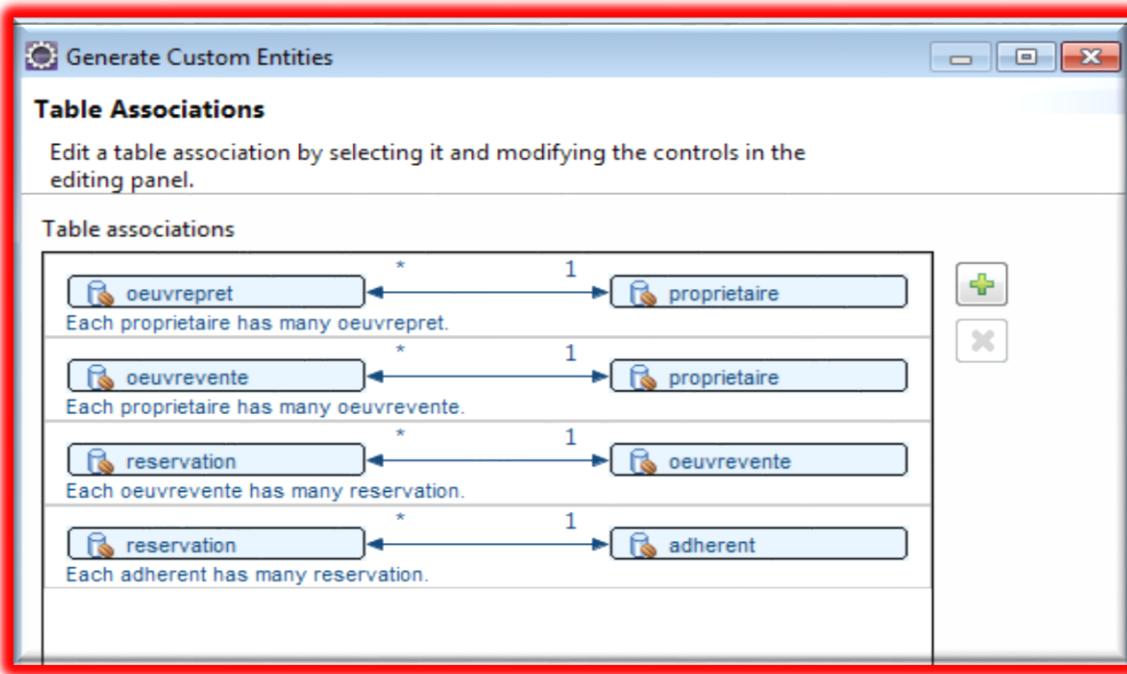
- Un répertoire metier avec les classes issues de la base de données
- un fichier persistence.xml dans un répertoire META-INF

Vous devez déplacer le fichier persistence.xml dans le répertoire main/resources/META-INF



Les associations suivantes ont été ajoutées :

Les clés étrangères sont reconnues et vont donner naissance à des références de navigation.



On sélectionne le modèle lazy et on définit le package métier pour accueillir les classes métiers issues de la base. On n'utilise pas le type Set qui pose quelques problèmes.

Vous devez modifier la classe ReservationEntityPK pour mettre les propriétés insertable et updatable à false.

```
@Column(name = "id_oeuvrerente", insertable = false, updatable = false)
@Id
public int getIdOeuvrerente() { return idOeuvrerente; }

public void setIdOeuvrerente(int idOeuvrerente) { this.idOeuvrerente = idOeuvrerente; }

@Column(name = "id_adherent", insertable = false, updatable = false)
@Id
public int getIdAdherent() { return idAdherent; }
```

Fichier persistance.xml

Le fichier persistance.xml est déployé dans le répertoire resources/META-INF de notre application. Il sera utilisé par notre manager.

Ce fichier contient :

- L'appel d'une datasource
- Le mapping des tables de l'application
- Les propriétés pour accéder à la base de données

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
    xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">

    <persistence-unit name="oeuvresjpa">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
        <class>com.epul.oeuvres.metier.AdherentEntity</class>
        <class>com.epul.oeuvres.metier.OeuvrepreteEntity</class>
        <class>com.epul.oeuvres.metier.OeuvreventeEntity</class>
        <class>com.epul.oeuvres.metier.ProprietaireEntity</class>
        <class>com.epul.oeuvres.metier.ReservationEntity</class>

        <properties>
            <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/baseoeuvre"/>
            <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
            <property name="hibernate.connection.username" value="userepul"/>
            <property name="hibernate.connection.password" value="epul"/>
            <property name="hibernate.archive.autodetection" value="class"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
            <property name="hbm2ddl.auto" value="update" />
            <property name="hibernate.max_fetch_depth" value="3"/>
        </properties>
    </persistence-unit>
</persistence>

```

Classes générées

```

ReservationEntity | idOeuvrevente
package com.epul.oeuvres.metier;

import ...

/**
 * Created by christian on 19/02/2017.
 */
@Entity
@Table(name = "reservation", schema = "baseoeuvre", catalog = "")
@IdClass(ReservationEntityPK.class)
public class ReservationEntity {
    private int idOeuvrevente;
    private int idAdherent;
    private Date dateReservation;
    private String statut;
    private OeuvreventeEntity oeuvreventeByIdOeuvrevente;
    private AdherentEntity adherentByIdAdherent;
}

```

On peut noter les ManyToOne dans cette classe

```
@ManyToOne  
@JoinColumn(name = "id_oeuvrevente", referencedColumnName = "id_oeuvrevente", nullable = false)  
public OeuvreventeEntity getOeuvreventeByIdOeuvrevente() { return oeuvreventeByIdOeuvrevente; }  
  
public void setOeuvreventeByIdOeuvrevente(OeuvreventeEntity oeuvreventeByIdOeuvrevente) {  
    this.oeuvreventeByIdOeuvrevente = oeuvreventeByIdOeuvrevente;  
}  
  
@ManyToOne  
@JoinColumn(name = "id_adherent", referencedColumnName = "id_adherent", nullable = false)  
public AdherentEntity getAdherentByIdAdherent() { return adherentByIdAdherent; }  
  
public void setAdherentByIdAdherent(AdherentEntity adherentByIdAdherent) {  
    this.adherentByIdAdherent = adherentByIdAdherent;  
}
```

Maven : Bibliothèques du projet

Pour ce projet, nous devons ajouter les dépendances suivantes :

- driver Mysql JDBC Driver
- couche hibernate en version 5.2
- provider hibernate
- couche JSTL

Voici le fichier pom.xml

La version d'hibernate sera définie dans les dépendances.

```
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
<java.version>1.8</java.version>
<hibernate.version>5.2.2.Final</hibernate.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>

    <!-- Test -->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.7</version>
        <scope>test</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.36</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>${hibernate.version}</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>${hibernate.version}</version>
    </dependency>
```

Développement de la couche DAO

Cette couche va implémenter les traitements de notre application. Les appels sous forme de requêtes écrites dans le langage HQL, sont pilotés par un EntityManager. C'est un objet de type `javax.persistence.EntityManager`. Il assure les interactions entre la base de données et les beans entités (classes issues du mapping). Il permet tout type de requête (recherche mais aussi ajout, modification, suppression). L'EntityManager est donc au cœur de toutes les actions de persistance.

Les beans entités étant de simple POJO ((Plain Old Java Object), leur instanciation se fait comme pour tout autre objet Java. Les données de cette instance ne sont rendues persistantes que par une action explicite demandée à l'EntityManager sur le bean entité.

Un EntityManager gère un ensemble défini de beans entités nommé persistence unit. La définition d'un persistence unit est assurée dans un fichier de description nommé `persistence.xml`.

Classe abstraite EntityManager

Nous allons écrire une classe abstraite nommée `EntityService` qui va déclarer un objet de type EntityManager, nommé emf et le connecter à notre fichier `persistence.xml`

```
package com.epul.oeuvres.dao;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

/**
 * Created by Valentin on 06/04/2016.
 */
public abstract class EntityService {

    protected EntityManager entitymanager;
    protected EntityManagerFactory emf;

    public EntityTransaction startTransaction() throws Exception
    {
        emf = Persistence.createEntityManagerFactory( persistenceUnitName: "oeuvresjpa");
        entitymanager = emf.createEntityManager();

        return entitymanager.getTransaction();
    }

}
```

Ajout des traitements

Les traitements sont définis dans une classe nommée Service qui va hériter de la classe EntityService.

```
1 package dao;
2
3 import meserreurs.MonException;
4 import java.util.*;
5
6 import javax.persistence.EntityTransaction;
7
8 import metier.*;
9
10 public class Service extends EntityService {
11
12     // ajout 'un adhérent
13 }
```

Ajout d'un adhérent

Voici le code de l'ajout d'un adhérent

```
public void insertAdherent(AdherentEntity unAdherent) throws MonException {
    try
    {
        EntityTransaction transac = startTransaction();
        transac.begin();
        entitymanager.persist(unAdherent);
        transac.commit();
        entitymanager.close();
    }
    catch (RuntimeException e)
    {
        new MonException("Erreur de lecture", e.getMessage());
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

On lance une transaction et on ajoute l'adhérent avec la méthode persist.

Recherche des adhérents

Ce traitement va utiliser une requête écrite en HQL.

```

/* Lister les adhérents
 */
public List<AdherentEntity> consulterListeAdherents() throws MonException {
    List<AdherentEntity> mesAdherents = null;
    try {
        EntityTransaction transac = startTransaction();
        transac.begin();
        mesAdherents = (List<AdherentEntity>)entitymanager.createQuery( s: "SELECT a FROM AdherentEntity a ORDER BY a.nomAdherent" ).getResultList();
        entitymanager.close();
    }
    catch (RuntimeException e)
    {
        new MonException("Erreur de lecture", e.getMessage());
    } catch (Exception e) {
        e.printStackTrace();
    }
    return mesAdherents;
}

```

Recherche d'un adhérent

Ce traitement utilise une méthode de recherche de l'objet EntityManager.

```

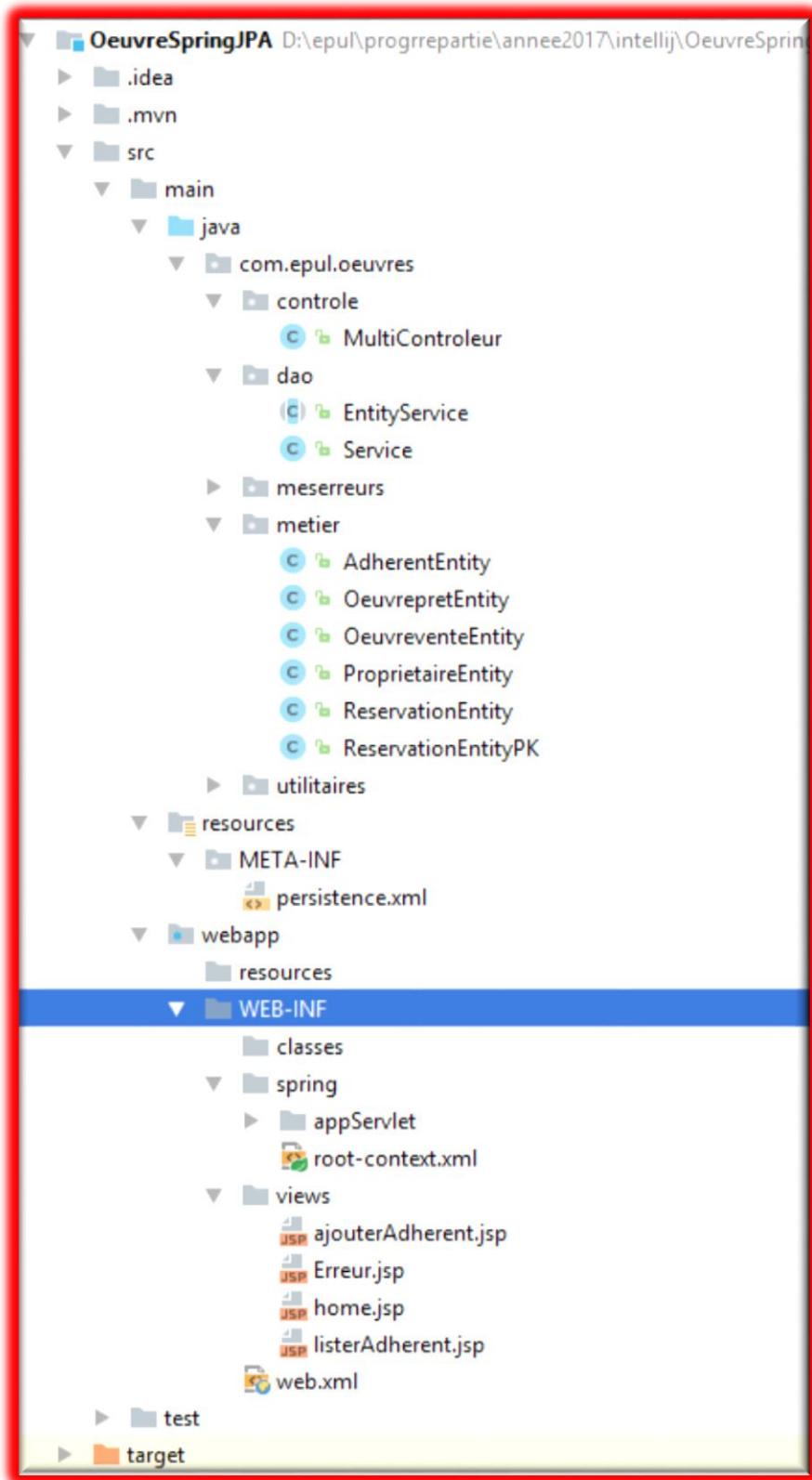
/* Consultation d'un adhérent par son numéro
 */
public AdherentEntity adherentById(int numero) throws MonException {
    List<AdherentEntity> adherents = null;
    AdherentEntity adherent = new AdherentEntity();
    try {
        EntityTransaction transac = startTransaction();
        transac.begin();

        adherents = (List<AdherentEntity>)entitymanager.createQuery( s: "SELECT a FROM AdherentEntity a WHERE a.idAdherent='"+numero+"'").getResultList();
        adherent = adherents.get(0);
        entitymanager.close();
    } catch (RuntimeException e)
    {
        new MonException("Erreur de lecture", e.getMessage());
    } catch (Exception e) {
        e.printStackTrace();
    }
    return adherent;
}

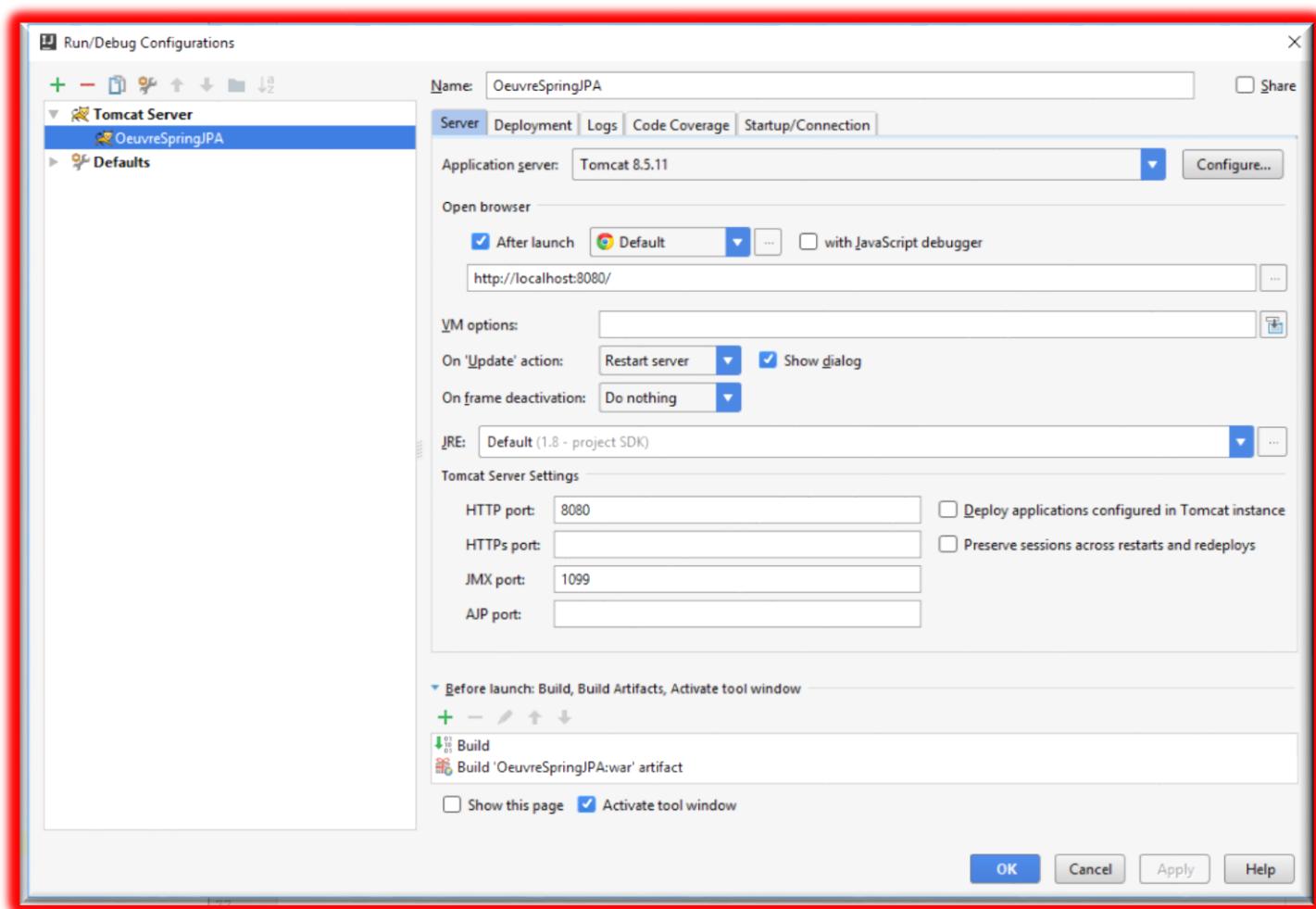
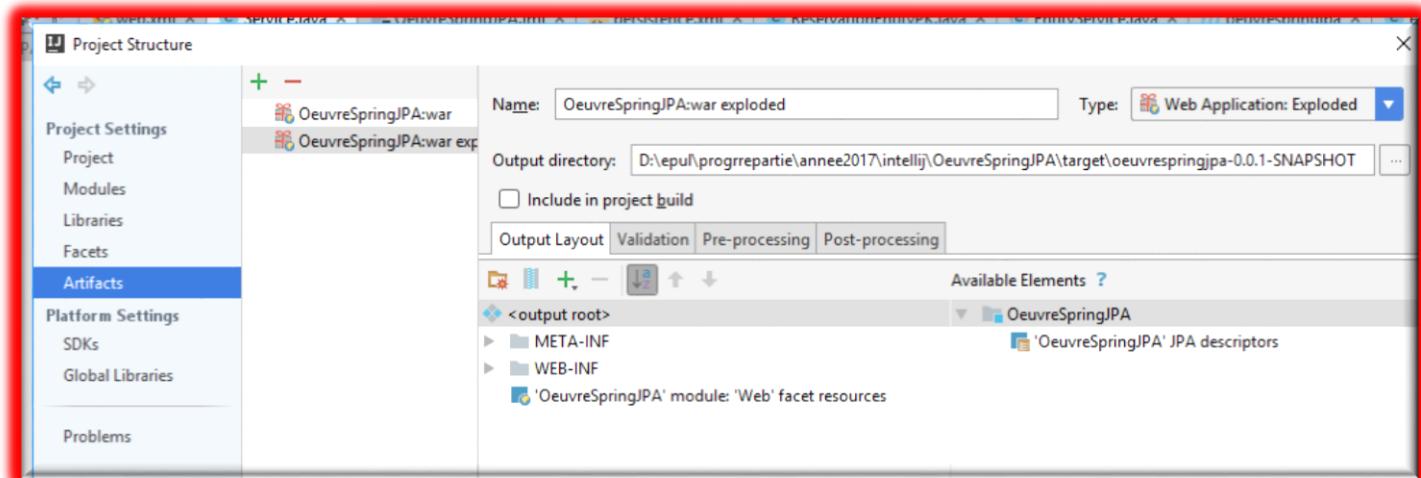
```

Couche de présentation

Voici l'architecture de l'application



Artifacts de l'application



Vous pouvez ajouter votre propre couche de présentation.

Travail à faire	
1.1	Construisez l'application à partir des sources données sous Spiral 
1.2	Ajoutez votre couche de présentation et les traitements que vous avez définis dans l'application Œuvres.