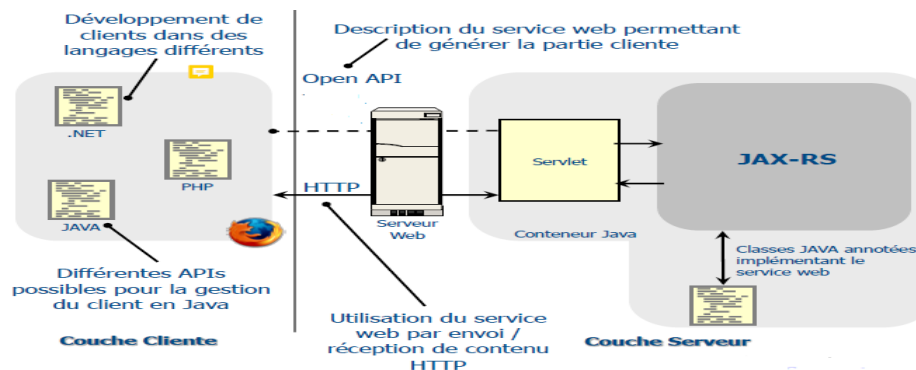


TP n°2 en Systèmes distribués

Objectifs : Mise en œuvre de l'architecture 3-tiers (Technologies Web Services)

Rappel:

- REST n'est pas un protocole ou un format, contrairement à SOAP, HTTP ou RCP, mais un style d'architecture inspiré de l'architecture du web fortement basé sur le protocole HTTP.
- Utiliser dans le développement des applications orientées ressources (ROA) ou orientées données (DOA)
- Les applications respectant l'architecture REST sont dites RESTful.
- Le développement des services web REST repose sur l'utilisation de classes Java et d'annotations.
- **JAX-RS** est l'acronyme Java API for RESTful Web Services.

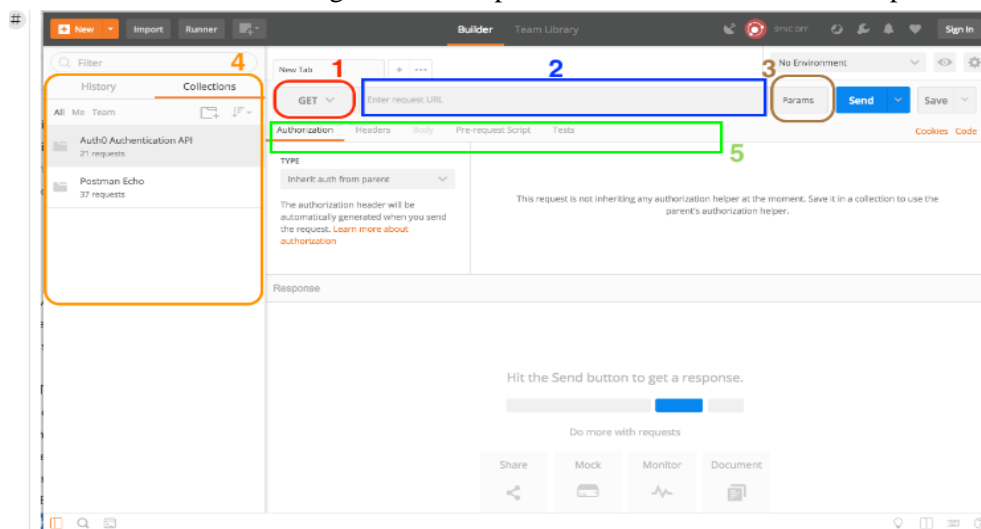


JAX-RS Fonctionnement

- Seule la configuration de la Servlet « JAX-RS » est requise pour réaliser le pont entre les requêtes HTTP et les classes Java annotées
- Un service web REST est déployé dans une application web

Télécharger et installer l'outil PostMan


C'est un logiciel qui se focalise sur les **tests des API**. Il est devenu très populaire pour tester des Microservices, notamment grâce à sa simplicité et ses fonctionnalités très spécialisées.




Interface de Postman (version Chrome)

Télécharger et décompresser la ressource de spécification Jersey pour REST : **jaxrs-ri-2.27**

https://jersey.github.io/download.html

 Jersey - RESTful Web Services in Java.

 **Jersey**

RESTful Web Services in Java.

JAX-RS 2.1 / Jersey 2.26+

Jersey 2.27, that implements JAX-RS 2.1 API is the most recent release of Jersey. To see the details about all changes, bug fixed and updates, please check the [Jersey 2.27 Release Notes](#).

For the convenience of non-maven developers the following links are provided:

- [Jersey JAX-RS 2.1 RI bundle](#) contains the JAX-RS 2.1 API jar, all the core Jersey module jars as well as all the required 3rd-party dependencies.
- [Jersey 2.27 Examples bundle](#) provides convenient access to the Jersey 2 examples for off-line browsing.

Exercice1: Créer un simple projet web dynamique Java nommé **restfulLivre**

1. Ajouter dans le dossier lib du projet l'ensemble de librairies du plugging Jersey

2. Ajouter le fichier web.xml au projet créer, avec les annotations suivantes :

```
<servlet>
  <servlet-name>RestServlet</servlet-name>
  <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
  <init-param>
    <param-name>jersey.config.server.provider.packages</param-name>
    <param-value>com.jaxrs.livre.ressource</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>RestServlet</servlet-name>
  <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```

3. Créer les **packages** et ajouter les quatre classes : Livre.Java, DaoClass.java, LivreService.java, et LivreResource.java

```
package com.jaxrs.livre.model;
```

```
import javax.xml.bind.annotation.XmlRootElement;
```

```
@XmlRootElement
```

```
public class Livre {
```

```
    private int code;
```

```
    private String titre;
```

```
    private String auteur;
```

```
    private double prix;
```

```
    // Ajouter le constructeur sans paramètre
```

```
    // Ajouter le constructeur pour initialiser les attributs
```

```
    // Ajouter les méthodes getters et setters
```

```
    // Ajouter la méthode ToString()
```

```
    }
```

```
}
```

```
//*****
```

```
package com.jaxrs.livre.dao;
```

```
import java.util.HashMap;
```

```
import java.util.Map;
```

```
import com.jaxrs.livre.model.Livre;
```

```
public class DaoClass {
```

```
    private static Map<Integer,Livre> livres = new HashMap<Integer,Livre>();
```

```
    public static Map<Integer, Livre> getLivres() {
```

```
        return livres;
```

```
    }
```

```
}
```

```
//*****
```

```
package com.jaxrs.livre.service;
```

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
import java.util.Map;
```

```
import com.jaxrs.livre.dao.DaoClass;
```

```
import com.jaxrs.livre.model.Livre;
```

```
public class LivreService {
```

```
    private Map<Integer,Livre> livres=
```

```
    DaoClass.getLivres();
```

```
    Livre l1=new Livre(1,"Java-RS","Alain",450);
```

```
    Livre l2=new Livre(2,"WS-RS","Alain",350);
```

```
    Livre l3=new Livre(3,"Json","Jon",300);
```

```
    public LivreService(){
```

```
        livres.put(1,l1);
```

```
        livres.put(2,l2);
```

```
        livres.put(3,l3);
```

```
    }
```

```
    public List<Livre> getAllLivres(){
```

```
        return new ArrayList<Livre>(livres.values());
```

```
    }
```

```
    public Livre getLivreById(int id){
```

```
        return livres.get(id);
```

```
    }
```

```
}
```

```
    public Livre getLivreByTitre(String titre){
```

```
        // à définir
```

```
    }
```

```
    public List <Livre> getLivreByAuteur(String
```

```
        auteur)
```

```
    {        // à définir    }
```

```
    public void AjoutLivre(int indice, Livre livre)
```

```
    {
```

```
        livres.put(indice, livre);
```

```
    }
```

```
    public void removeLivreById(int id){
```

```
        livres.remove(id);
```

```
    }
```

```
    public void modifLivre(int id, Livre livre){
```

```
        livres.replace(id, livre);
```

```
    }
```

```

/*****
package com.jaxrs.livre.ressource;

import java.util.List;

import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;

import com.jaxrs.livre.model.Livre;
import com.jaxrs.livre.service.LivreService;

@Path("/livres")
public class LivreResource {
    LivreService livreService = new LivreService();

    @GET
    @Path( "/getalltext" )
    @Produces(MediaType.TEXT_PLAIN) //Text_PLAIN
    public String getLivresText(){ return livreService.getAllLivres }

    @GET @Path("/getalljson") @Produces(MediaType.APPLICATION_JSON)
    public List<Livre> getLivresjson(){ ..... }

    @GET @Path( "/getallxml" ) @Produces(MediaType.APPLICATION_XML)
    public List<Livre> getLivresxml(){ ..... }

    @GET
    @Path("/getbyid/{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public Livre getLivreById(@PathParam("id") int id){ ..... }

    @POST
    @Path("/addlivre")
    @Consumes(MediaType.APPLICATION_JSON) @Produces(MediaType.APPLICATION_JSON)

    public Livre addLivre(Livre livre){
        int id=livreService.getAllLivres().size()+1;
        livreService.AjoutLivre(id, livre);
        return livreService.getLivreById(id);
    }
    // Définir la méthode qui permet de modifier un livre par code.
    @PUT
    @Path("/putlivre/{id}")
    @Consumes(MediaType.APPLICATION_JSON) @Produces(MediaType.APPLICATION_JSON)
    public Livre putLivre(@PathParam("id") int id, Livre livre){
        livre.setId(id);
        livreService.modifLivre(id, livre);
        return livreService.getLivreById(id);
    }

    // Ajouter la méthode qui permet de supprimer un objet livre
    @DELETE @Path("/delbyid/{id}")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    public void deleteLivreById(@PathParam("id") int id) {
        .....
    }
}

```

Exercice 2. Web Service Restful avec Maven Project et Hibernate - JPA

- **Maven est un projet Open source**, porté par la fondation Apache (<http://maven.apache.org/>). Il est utilisé pour **automatiser l'intégration continue** lors d'un développement de logiciel. Il utilise un paradigme connu sous le nom de **Project Object Model (POM)** afin de décrire un projet logiciel, ses **dépendances** avec des modules externes et l'ordre à suivre pour sa production. Il est livré avec un grand nombre de tâches prédéfinies, comme la compilation de code Java ou encore sa modularisation.
- Un élément clé et relativement spécifique de **Maven** est son aptitude à fonctionner en réseau. Une des motivations historiques de cet outil est de fournir un moyen de synchroniser des projets indépendants : publication standardisée d'information, distribution automatique de modules jar
- Maven impose une arborescence et un nommage des fichiers du projet selon le **concept de Convention** plutôt que configuration. Ces conventions permettent de réduire la configuration des projets, tant qu'un projet suit les conventions. Si un projet a besoin de s'écarter de la convention, le développeur le précise dans la configuration du projet.

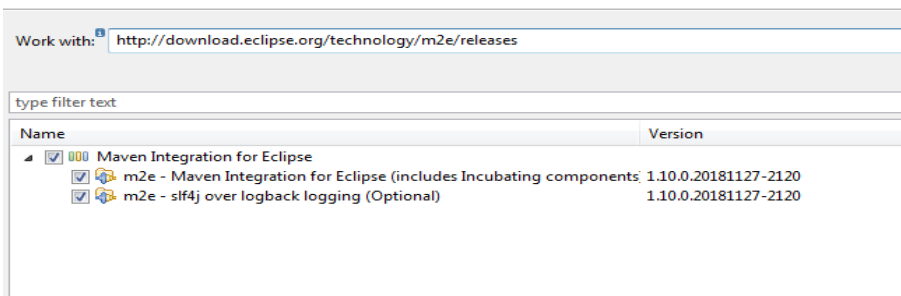
Voici une liste non exhaustive des répertoires d'un projet Maven :

- /src : les sources du projet
 - /src/main : code source et fichiers source principaux
 - /src/main/java : code source
 - /src/main/resources : fichiers de ressources (images, fichiers annexes, etc.)
 - /src/main/webapp : webapp du projet
 - /src/test : fichiers de test
 - /src/test/java : code source de test
 - /src/test/resources : fichiers de ressources de test
 - /src/site : informations sur le projet et/ou les rapports générés suite aux traitements effectués
 - /target : fichiers résultat, les binaires (du code et des tests), les packages générés et les résultats des tests
- **Installation de Maven**
 - **Solution 1 :** Maven est un outil séparé d'Eclipse, qui peut d'ailleurs se télécharger séparément.

Dans Eclipse cliquer sur menu *Help* ⇒ *Install new Software* ⇒ *Add* et utiliser la location suivante: <http://download.eclipse.org/technology/m2e/releases>

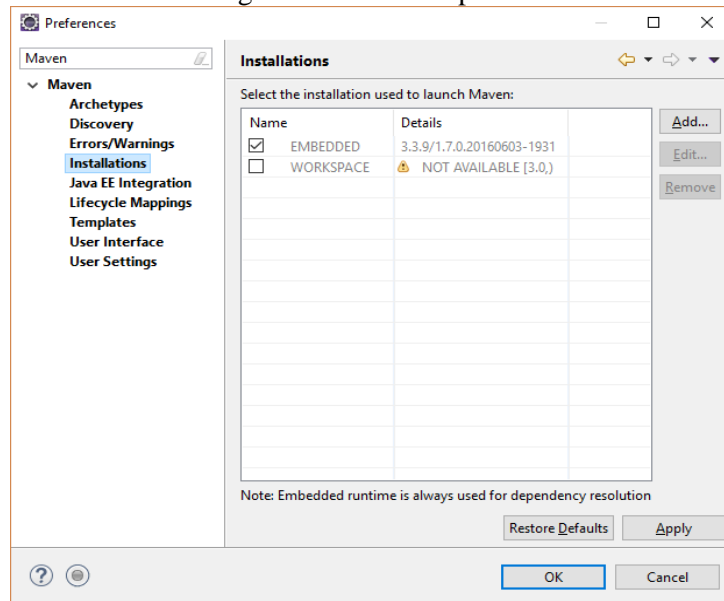
Available Software

Check the items that you wish to install.

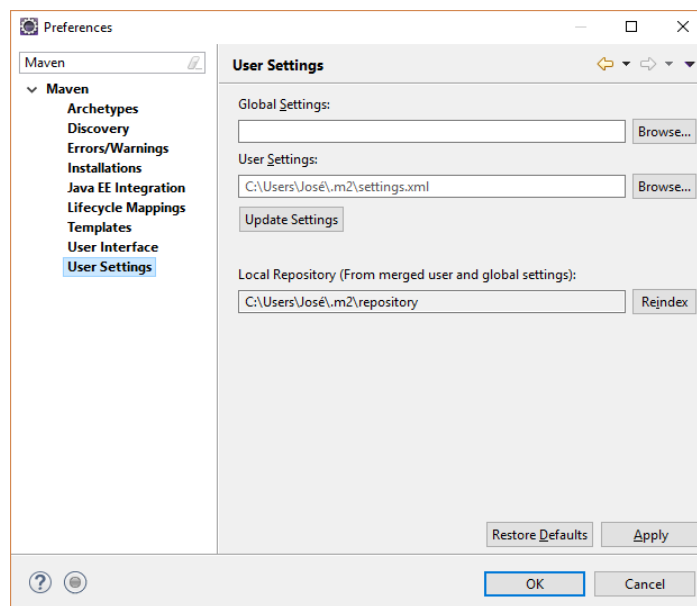


- **Solution 2 :** Lorsque l'on télécharge Eclipse, on télécharge avec lui une version de Maven, qui est embarquée dans Eclipse. On peut aussi choisir d'utiliser une autre version de Maven, installée dans un répertoire de notre disque, avec sa configuration propre. Pour ce faire, il faut aller chercher dans les Préférences d'Eclipse (Menu Window).
- On peut filtrer l'ensemble des préférences avec le mot-clé Maven, et obtenir l'ensemble des préférences Eclipse pour Maven:

- La première que nous allons voir est la préférence Installation, qui nous permet de choisir l'installation de Maven qu'Eclipse doit utiliser. On peut aller chercher une installation locale au travers du bouton Add... et naviguer vers le bon répertoire.



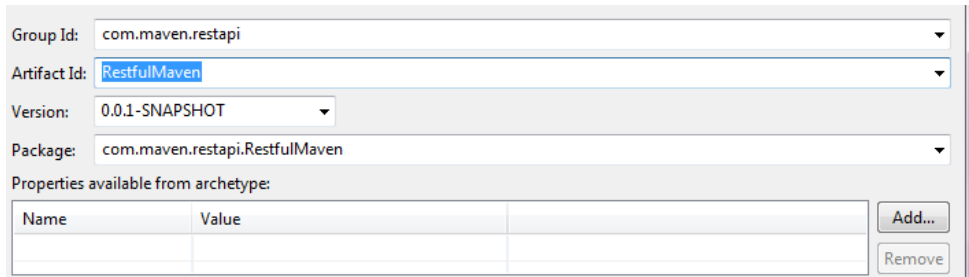
- La deuxième préférence que nous voyons ici, permet d'avoir des informations sur la configuration de Maven. En particulier, on peut lire ici le répertoire que Maven utiliser comme cache. Un cache bien utilisé peut rapidement atteindre plusieurs Go de données.



1. Créer un projet Maven simple : **Cliquer sur menu File/New Project/Maven project**

```
<groupId>com.maven.restapi.JPAHibernate</groupId>
<artifactId>RestfulHibernateMavenLivre</artifactId>
```

- Le **groupId**, constitue la racine du nom de tous les projets gérés par Maven. La façon naturelle pour choisir ce nom racine est de procéder comme pour le choix des noms de package : d'utiliser un nom de domaine que l'on possède.
- l'**artifactId**. Il s'agit du nom du projet proprement dit.
- Le dernier paramètre est le **numéro de version**. Un numéro de version évolue dans la vie d'un projet. Un projet donné, repéré par un groupId et artifactId peut donc exister en plusieurs versions.



Maven crée les répertoires:

- **src/main/java** et **src/test/java**, pour le code source de projet et des tests.
 - **src/main/resources** dans lequel on range des fichiers de ressources.
- Maven a créé un fichier **pom.xml**. Ce fichier, appelé fichier **POM (Project Object Model)** est le fichier central du projet Maven, dans lequel Maven enregistre toutes les informations dont il a besoin.

2. Ajouter les dépendances à ce projet

Maven propose un mécanisme très pratique de déclaration de ces dépendances. Il gère le téléchargement automatique de ces dépendances dans un cache sur notre disque local, et la déclaration de ces dépendances dans Eclipse.

Exemple : Modifier le fichier POM (**Project Object Model**).

<pre><!-- https://mvnrepository.com/artifact/org.glassfish.jersey.bundles/jaxrs-ri --> <dependency> <groupId>org.glassfish.jersey.bundles</groupId> <artifactId>jaxrs-ri</artifactId> <version>2.27</version> </dependency></pre>	Cette déclaration indique à Maven que nous souhaitons mettre jaxrs-ri en version 2.27 en dépendance de notre projet.
<pre><!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core --> <dependency> <groupId>org.hibernate</groupId> <artifactId>hibernate-core</artifactId> <version>5.1.0.Final</version> </dependency></pre>	Ajout de bibliothèques de Hibernate
<pre><!-- <dependency> <groupId>com.oracle</groupId> <artifactId>ojdbc5</artifactId> <version>11.2.0.1.0</version> </dependency> --> <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java --> <dependency> <groupId>mysql</groupId> <artifactId>mysql-connector-java</artifactId> <version>8.0.18</version> </dependency></pre>	Ajout la dépendance à la base de données oracle ou mysql

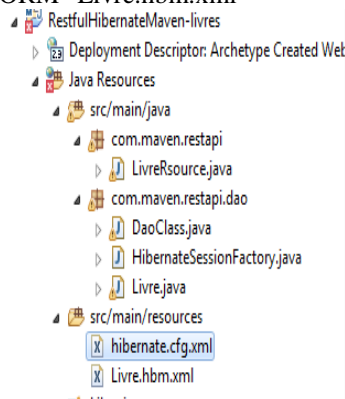
Si vous avez choisi la dépendance avec la base de données oracle alors, il faut télécharger et installer le driver de la base de données Oracle: ojdbc5 / ojdbc6, et voici les étapes à suivre:

3.1 Télécharger et décompresser l'outil : **apache-maven-3.6.0-bin**
3.2 Ajouter le dossier .. \apache-maven-3.6.0-bin\bin dans le Path du variable d'environnement (ou redémarrer votre ordinateur)
3.3 se positionner dans le dossier jdbc\lib d'Oracle et exécuter la commande:
C:\app\...\Product\11.2.0\dbhome_1\jdbc\lib>**mvn** install:install-file -DgroupId=com.oracle -DartifactId=ojdbc6 -Dversion=11.2.0.1.0 -Dpackaging=jar -Dfile=ojdbc6.jar -DgeneratePom=true

3. Ajouter au fichier web.xml la déclaration du conteneur-web de déploiement (ServletContainer) et les packages de spécifications Jersey

```
<servlet>
  <servlet-name>RestfulHibernateMaven-livres</servlet-name>
  <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
  <init-param>
    <param-name>jersey.config.server.provider.packages</param-name>
    <param-value>com.maven.restapi</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>RestfulHibernateMavenLivres</servlet-name>
  <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```

4. Ajouter au projet maven le fichier de configuration "hibernate.cfg.xml", et le fichier de correspondance ORM "Livres.hbm.xml"



```
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.
<hibernate-mapping package="com.maven.restapi.dao">
  <class name="Livre" table="livre">
    <id name="id" type="integer">
      <column name="id" sql-type="int(4)" />
      <generator class="increment" />
    </id>
    <property name="titre" type="string"> <column name="titre" sql-type="varchar(50)" not-null="true" />
  </property>
    <property name="auteur" type="string"> <column name="auteur" sql-type="varchar(50)" not-null="true" />
  </property>
    <property name="prix" type="double"> <column name="prix" sql-type="number" not-null="true" /> </property>
  </class>
</hibernate-mapping>
```

Livres.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
```

hibernate.cfg.xml en cas la dépendance avec la base de données mysql


```

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd ">
<hibernate-configuration>
  <session-factory>
    <!-- Database connection settings -->
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <!-- <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernatedb</property>-->
    <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernatedb?useTimezone=true&serverTimezone=UTC
</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">admin</property>

    <!-- JDBC connection pool (use the built-in) -->
    <property name="connection.pool_size">1</property>
    <property name="hibernate.current_session_context_class">thread</property>
    <!-- SQL dialect -->
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLInnoDBDialect</property>
    <!-- Echo all executed SQL to stdout -->
    <property name="show_sql">>false</property>

    <!-- Drop and re-create the database schema on startup -->
    <property name="hibernate.hbm2ddl.auto">update</property>

    <property
name="hibernate.transaction.factory_class">org.hibernate.transaction.JDBCTransactionFactory</property>
    <!-- Mapping file -->
    <mapping resource="Livre.hbm.xml"/>
  </session-factory>
</hibernate-configuration>

```

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd ">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:bd1</property>
    <property name="hibernate.connection.username">system</property>
    <property name="hibernate.connection.password">bd1</property>
    <property name="hibernate.connection.pool_size">1</property>

    <property name="hibernate.current_session_context_class">thread</property>
    <property name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property>

    <property name="hibernate.show_sql">>false</property>
    <property name="hibernate.transaction.factory_class">org.hibernate.transaction.JDBCTransactionFactory</property>
    <!-- Mapping file -->
    <mapping resource="Livre.hbm.xml"/>
  </session-factory>
</hibernate-configuration>

```

hibernate.cfg.xml en cas la dépendance avec la base de données oracle

5. Créer la classe de mapping ORM : HibernateSessionFactory.java
6. Créer les classes des web services : Livre.java, LivreService.java et la classe de ressource LivreResource.java

@entity: Déclare que cette classe ne s'agit pas d'une classe ordinaire mais d'une table à la base de donnée qui sera persiste.

@id: le champ id est un identifiant de la table etudiant.

@GeneratedValue: génération d'une clé auto incrémente.

```

package com.maven.restapi.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;

```

```

package com.maven.restapi;

import java.util.List;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;

```

<pre> import javax.persistence.Id; import javax.persistence.Table; import javax.xml.bind.annotation.XmlRootElement; @Entity @Table(name = "Livre") @XmlRootElement(name = "Livre") public class Livre { @Id @Column(name="ID") private int id; @Column(name = "titre") private String titre; @Column(name = "auteur") private String auteur; @Column(name = "prix") private float reffil; // getters et setters .. } </pre>	<pre> import javax.ws.rs.POST; import javax.ws.rs.PUT; import javax.ws.rs.Path; import javax.ws.rs.PathParam; import javax.ws.rs.Produces; import javax.ws.rs.core.MediaType; import javax.ws.rs.core.Response; import com.maven.restapi.model.Etudiant; import com.maven.restapi.service.EtudiantService; @Path("/livres") public class LivreResource { LivreService dao = new LivreService(); } </pre>
<pre> package com.maven.restapi.service; import java.util.List; import org.hibernate.Query; import org.hibernate.Session; import org.hibernate.Transaction; import com.maven.restapi.model.Etudiant; public class LivreService { Session session=null; } </pre>	

1. Compléter les codes sources des trois classes précédentes
2. Créer dans la base de données mysql la table Livre
Create table Livre (Id int primary key, titre varchar(50), auteur varchar(50), prix number(7,2));
3. Appliquer les mise à jour et Exécuter le projet Maven: Maven génère le fichier .war dans le dossier target;
Test les web services avec l'outil PostMan: Copier le fichier .war dans le dossier webapps du Tomcat, et redémarrer le Tomcat.