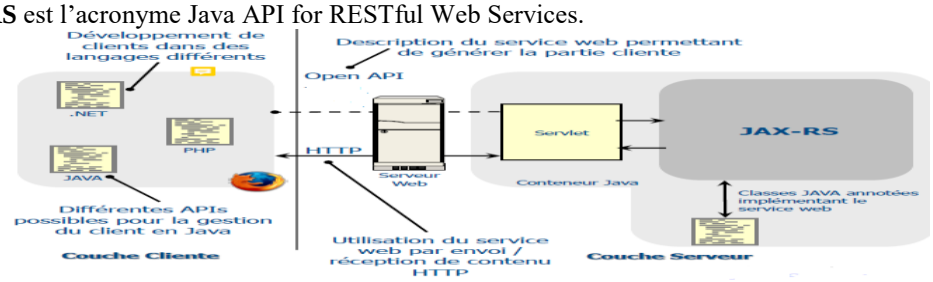
Université Mohammed V de Rabat
**Ecole Nationale Supérieure d'Arts et Métiers**
Filière INDIA (2eme année – S4) A.U. : 2025/2026 Pr. A. El Qadi

# TD-TP n2 – Virtualisation & Architectures Logicielles Distribuées

## Objectifs : Mise en œuvre de l'architecture 3-tiers (Technologie REST 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 orientés 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
- Les API REST écoutent les méthodes HTTP telles que GET, POST et DELETE pour savoir quelles opérations effectuer sur les ressources du service Web.

Une ressource est toute donnée disponible dans le service Web qui peut être consultée et manipulée avec des requêtes HTTP à l'API REST. La méthode HTTP indique à l'API quelle action effectuer sur la ressource. Bien qu'il existe de nombreuses méthodes HTTP, les quatre méthodes répertoriées ci-dessous sont les plus couramment utilisées avec les API REST :

| **Méthode HTTP** | **Description** |
|---|---|
| GET | Récupère une ressource existante |
| POST | Créer une nouvelle ressource. |
| PUT | Mettre à jour une ressource existante |
| DELETE | Supprimer une ressource. |

**Pour les TPs**

**Solution 1 :**

- **Il faut disposer d'un IDE : Eclipse IDE for Enterprise Java and Web Developers:**

**https://www.eclipse.org/downloads/packages/release/2022-06/r/eclipse-ide-enterprise-java-and-web-developers**

- **Un Outils de tests des API CRUD (Postman)**

**Télécharger et installer l'outil PostMan : https://www.postman.com/downloads/**

- **Un serveur d'application (Conteneur Web)**

**Télécharger et installer le serveur d'application Apache Tomcat 9.0/11.0**
**https://tomcat.apache.org/download-90.cgi**

**Solution 2 : IDE Visual Studio Code, avec extensions indispensables :**

- Extension Pack for Java (par Microsoft)
- Extension Tomcat for Java (ou Community Server Connectors)
- Spring Boot Extension Pack (par VMware)
- Language Support for Java(TM) by Red Hat
- (Optionnel mais utile) Maven for Java (par Microsoft)
- MySql
- Posyman

## Solution 2 : Visual studio code

## Ex1. Mise en oeuvre de l'artchitecture REST avec Maven Project

– **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.

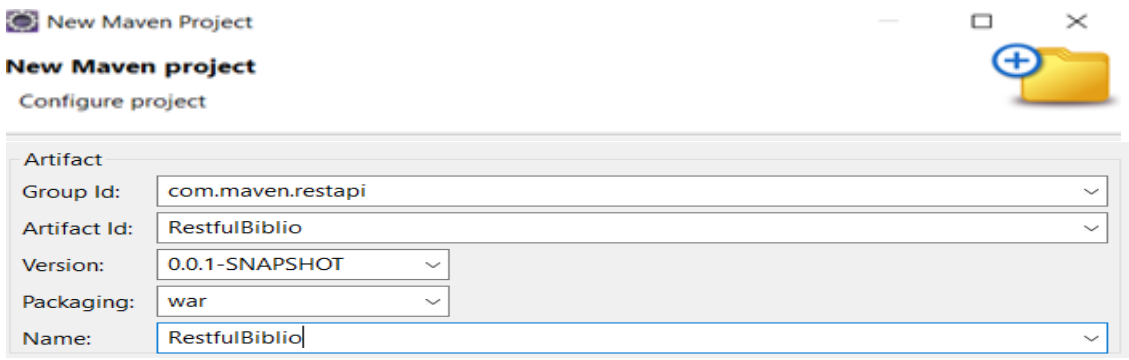
Voici une liste non exhaustive des répertoires d'un projet Maven :

| | |
|---|---|
| • /src : les sources du projet<br>• /src/main : code source et fichiers source principaux<br>• /src/main/java : code source<br>• /src/main/resources : fichiers de ressources (images, fichiers annexes, etc.)<br>• /src/main/webapp : webapp du projet<br>• /src/test : fichiers de test | • /src/test/java : code source de test<br>• /src/test/resources : fichiers de ressources de test<br>• /src/site : informations sur le projet et/ou les rapports générés suite aux traitements effectués<br>• /target : fichiers résultat, les binaires (du code et des tests), les packages générés et les résultats des tests |

### Créer un projet Maven (RestfulBiblio) depuis VS Code

1. Ouvrir un terminal intégré dans VS Code (Ctrl + ù), et **exécuter la commande suivante** :

```
mvn archetype:generate "-DgroupId=com.maven.restapi" "-DartifactId=RestfulBiblio" "-DarchetypeArtifactId=maven-archetype-webapp" "-DinteractiveMode=false"
```




```
<groupId>com.maven.restapi</groupId>
<artifactId>RestfulBiblio</artifactId>
```

- Le **groupId**, constitue la racine du nom de tous les projets gérés par Maven. La façon naturelle pour choisir le 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.

**Structure attendue** :

```
RestfulBiblio/
├── src/
│   └── main/
│       ├── java/          ← à créer manuellement
│       ├── resources/     ← à créer manuellement
│       └── webapp/
│           └── WEB-INF/
│               └── web.xml
└── pom.xml
```

## 2. Ajouter les dépendances à ce projet

**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.

**Maven** propose un mécanisme très pratique de déclaration de dépendances. Il gère le téléchargement automatique de ces dépendances **dans un cache sur le disque local.**

**Modifier le fichier POM,** remplacer le contenu du pom.xml par celui-ci **(**corrigé et adapté **à** Jakarta EE 9+**,** car **Jersey 3+** utilise Jakarta au lieu de javax):

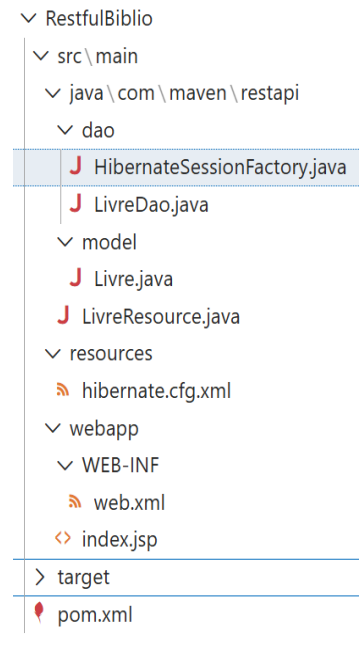
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.maven.restapi</groupId>
  <artifactId>RestfulBiblio</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>RestfulBiblio</name>

   <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
    <!-- JUnit -->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.13.2</version>
      <scope>test</scope>
    </dependency>
```

```xml
<!-- Jersey JAX-RS RI (Jakarta EE 9+) -->
   <dependency>
     <groupId>org.glassfish.jersey.bundles</groupId>
     <artifactId>jaxrs-ri</artifactId>
     <version>3.1.0</version>
   </dependency>

   <!-- Hibernate ORM 6 (Jakarta Persistence) -->
   <dependency>
     <groupId>org.hibernate.orm</groupId>
     <artifactId>hibernate-core</artifactId>
     <version>6.5.0.Final</version>
   </dependency>

   <!-- MySQL Connector -->
   <dependency>
     <groupId>mysql</groupId>
     <artifactId>mysql-connector-java</artifactId>
     <version>8.0.33</version> <!-- version plus récente et stable -->
   </dependency>

   <!-- Servlet API (fourni par Tomcat) -->
   <dependency>
     <groupId>jakarta.servlet</groupId>
     <artifactId>jakarta.servlet-api</artifactId>
     <version>5.0.0</version>
     <scope>provided</scope>
   </dependency>
 </dependencies>
 <build>
   <finalName>RestfulBiblio</finalName>
   <plugins>
     <plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-compiler-plugin</artifactId>
       <version>3.11.0</version>
       <configuration>
         <source>17</source>
         <target>17</target>
       </configuration>
     </plugin>
     <plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-war-plugin</artifactId>
       <version>3.3.2</version>
     </plugin>
   </plugins>
 </build>
</project>
```

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

Modifier **src/main/webapp/WEB-INF/web.xml** :

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
                        https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd"
        version="5.0">
  <display-name>RestfulBiblio</display-name>
  <servlet>
    <servlet-name>RestfulBiblio</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>RestfulBiblio</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
</web-app>
```




### 4. Ajouter au projet Maven le fichier de configuration "hibernate.cfg.xml"

Créer le fichier **src/main/resources/hibernate.cfg.xml** :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
    <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernatedb?useSSL=false&amp;serverTimezone
=UTC</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">root</property>

    <property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>
    <property name="hibernate.current_session_context_class">thread</property>
    <property name="hibernate.show_sql">false</property>
    <property name="hibernate.connection.pool_size">1</property>

    <mapping class="com.maven.restapi.model.Livre"/>
  </session-factory>
</hibernate-configuration>
```

5. **Créer la classe de mapping ORM :**
com/maven/restapi/dao/**HibernateSessionFactory.java**

```
package com.maven.restapi.dao;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateSessionFactory {
   private static final SessionFactory sessionFactory = buildSessionFactory();
   private static SessionFactory buildSessionFactory() {
      try {      return new Configuration().configure().buildSessionFactory();   }
     catch (Throwable ex) {
         System.err.println("Error in buildSessionFactory(): " + ex);
         throw new ExceptionInInitializerError(ex);
      }
   }
   public static SessionFactory getSessionFactory() {     return sessionFactory;   }
   public static Session currentSession() {   return sessionFactory.openSession();   }
   public static void closeSession() {     sessionFactory.close();    }
}
```

**6. Créer les packages et classes Java**

Dans **src/main/java**, crée les dossiers :

- com/maven/restapi
- com/maven/restapi/model
- com/maven/restapi/dao

Créer et compléter les codes sources des trois classes : Livre, LivreDao, et LivreResource.

**@entity**: Déclare que cette classe ne s'agit pas d'une classe ordinaire mais d'une table à la base de donne qui sera persiste.
**@id**: le champ id est un identifiant de la table etudiant.
**@GeneratedValue**: génération d'une clé auto incrémente.

```
// Livre.java
package com.maven.restapi.model;
import jakarta.persistence.*;
import jakarta.xml.bind.annotation.XmlRootElement;

@Entity
@Table(name = "Livre")
@XmlRootElement
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 double prix;

    // constructeur sans parametres
   // constructeur avec parametres
   // Getters & Setters
   @Override
   public String toString() {
      return "Livre{id=" + id + ", titre='" + titre + "', auteur='" +
auteur + "', prix=" + prix + "}";
   }
}
```

```java
//LivreDao.java
package com.maven.restapi.dao;

import java.util.List;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.query.Query;
import com.maven.restapi.model.Livre;

public class LivreDao {

   public String getMessage() {  return "hibernate jpa ok";  }

   public List<Livre> getLivres() {
      Session session = HibernateSessionFactory.currentSession();
      List<Livre> livres = session.createQuery("FROM Livre",
Livre.class).list();
      session.close();
      return livres;
   }

   public Livre getLivreById(int id) {
      Session session = HibernateSessionFactory.currentSession();
      Livre livre = session.get(Livre.class, id);
      session.close();
      return livre;
   }

   public Livre getLivreByTitre(String titre) {
      Session session = HibernateSessionFactory.currentSession();
      Query<Livre> query = session.createQuery("FROM Livre
WHERE titre = :titre", Livre.class);
      query.setParameter("titre", titre);
      Livre result = query.uniqueResult();
      session.close();
      return result;
   }

   public Livre saveLivre(Livre livre) {
      Session session = HibernateSessionFactory.currentSession();
      Transaction tx = session.beginTransaction();
      session.save(livre);
      tx.commit();
      session.close();
      return livre;
   }

   public int updateLivre(int id, Livre livre) {
      Session session =
HibernateSessionFactory.currentSession();
      Transaction tx = session.beginTransaction();
    Query query = session.createQuery(
         "UPDATE Livre SET titre = :titre, auteur = :auteur,
prix = :prix WHERE id = :id");
      query.setParameter("id", id);
      query.setParameter("titre", livre.getTitre());
      query.setParameter("auteur", livre.getAuteur());
      query.setParameter("prix", livre.getPrix());
      int rows = query.executeUpdate();
      tx.commit();
      session.close();
      return rows;
   }

   public int deleteLivre(int id) {
      Session session =
HibernateSessionFactory.currentSession();
      Transaction tx = session.beginTransaction();
      Query query = session.createQuery("DELETE FROM
Livre WHERE id = :id");
      query.setParameter("id", id);
      int rows = query.executeUpdate();
      tx.commit();
      session.close();
      return rows;
   }
}
```

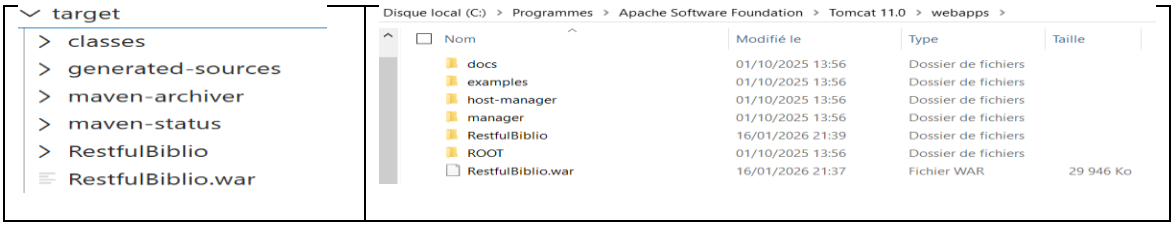
```java
// LivreResource.java
package com.maven.restapi;

import java.util.List;
import jakarta.ws.rs.*;
import jakarta.ws.rs.core.MediaType;
import jakarta.ws.rs.core.Response;
import com.maven.restapi.dao.LivreDao;
import com.maven.restapi.model.Livre;

@Path("/livres")
public class LivreResource {
    LivreDao dao = new LivreDao();

    @GET
    @Path("/message")
    @Produces(MediaType.TEXT_PLAIN)
    public String getMessage() {
        return "OK";
    }

    @GET
    @Path("/getallLivres")
    @Produces(MediaType.APPLICATION_JSON)
    public List<Livre> getLivres() {
        return dao.getLivres();
    }

    @GET
    @Path("/getlivreByid/{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public Livre getLivreById(@PathParam("id") int id) {
        return dao.getLivreById(id);
    }

    @GET
    @Path("/getlivreBytitre/{titre}")
    @Produces(MediaType.APPLICATION_JSON)
    public Livre getLivreByTitre(@PathParam("titre") String titre)  {
        return dao.getLivreByTitre(titre);
    }

    @POST
    @Path("/postLivre")
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Response addLivre(Livre livre) {
        dao.saveLivre(livre);
        return Response.ok().build();
    }

    @PUT
    @Path("/updLivre/{id}")
    @Consumes(MediaType.APPLICATION_JSON)
    public Response updateLivre(@PathParam("id") int id,
Livre livre) {
        int count = dao.updateLivre(id, livre);
        if (count == 0) {
            return
Response.status(Response.Status.BAD_REQUEST).build();
        }
        return Response.ok().build();
    }

    @DELETE
    @Path("/delete/{id}")
    public Response deleteLivre(@PathParam("id") int id) {
        int count = dao.deleteLivre(id);
        if (count == 0) {
            return
Response.status(Response.Status.BAD_REQUEST).build();
        }
        return Response.ok().build();
    }
}
```

### 7. Compiler et générer le .war

Dans le terminal de VS Code : **mvn clean package**

Maven génére le fichier **.war** dans le dossier target;

- Copier ce fichier dans le **dossier webapps** du repertoire Apache Tomcat




- Arreter et redemarrer le sercveur Tomcat

### 8. Créer dans la base de données hibernatedb.mysql la table Livre

Create table Livre (Id int primary key, titre varchar (50), auteur varchar(50), prix number(7,2));

9. **Tester les differentes ressources avec avec l'outil Postman:**
   http://localhost:8081/RestfulBiblio/rest/livres/message
   ok

   http://localhost:8081/RestfulBiblio/rest/livres/postLivre

   **Methode : Post**
   Cliquer sur l'onglet "Body", puis :

| Cas 1 : Envoyer du **JSON** (le plus courant pour les API REST)<br>1. Sélectionner raw,<br>2. Dans le menu déroulant à droite, choisir JSON<br>3. Entrer le jeu de donnees format json :<br>{<br>"id":1,<br>"prix":2500.0,<br>"titre":"article1"<br>} | Cas 2 : Envoyer un **formulaire HTML** (x-www-form-urlencoded)<br>1. Sélectionner x-www-form-urlencoded<br>2. Ajouter les paires clé/valeur :<br>Key : titre → Value : 1984<br>Key : auteur → Value : George Orwell<br><br>Cas 3 : Envoyer un **formulaire avec fichiers** (form-data)<br>1. Sélectionner form-data<br>2. Ajouter des champs texte ou fichiers (utile pour les uploads) |
|---|---|


   http://localhost:8081/RestfulBiblio/rest/livres/getallLivres
   [{"id":1,"prix":2500.0,"titre":"article1"}]

**Ex2.** Ajouter au premier projet les deux classes Etudiant.java et Emprunt.java

```
classe Etudiant {
        private int id;
        private String nom;
        private String prenom;
        private String reffil;
}
classe Emprunt {
        private int idEtud;
        private int idLivre
        private date datesortie
        private date dateretour
}
```

En suivant les étapes du 1er exerrcie, créer les classes permettant de gérer les informations des étudiants et des prêts de livres dans une bibliothèque en utilisant une architecture REST, ajouter des fonctionnalités supplémentaires comme la persistance des données avec une base de données (par exemple, en utilisant JPA/Hibernate).

Utiliser un outil comme Postman pour tester les endpoints REST :
GET /etudiants pour obtenir tous les étudiants.
POST /etudiants avec un corps JSON pour ajouter un étudiant.
GET /emprunts pour obtenir tous les emprunts.
POST /emprunts avec un corps JSON pour ajouter un emprunt.
GET /emprunts/{idEtud} pour obtenir tous les emprunts d'un étudiant spécifique.

**Ex3. Développement d'une API REST pour la gestion des inscriptions dans un centre de formation**

1. Créer un projet Maven standard de type web application (archétype maven-archetype-webapp) avec les coordonnées suivantes :
   - groupId : com.formation.api
   - artifactId : GestionStagiaires
   - version : 1.0.0
2. Configurer du fichier pom.xml

Ajouter dans le fichier pom.xml les dépendances nécessaires :

- jakarta.ws.rs-api (ou org.glassfish.jersey.bundles:jaxrs-ri pour Jersey 3+)
- jakarta.servlet-api
- junit (pour les tests)

Les plugins :

- maven-compiler-plugin (Java 11 ou 17)
- maven-war-plugin

3. Modifier le fichier src/main/webapp/WEB-INF/web.xml pour configurer le conteneur JAX-RS (ServletContainer) et indiquer le package contenant les ressources REST.
4. Création des classes métier et du service

   a) Créer le package com.formation.api.model

   Définir la classe Stagiaire.java avec les attributs :
   - id (identifiant unique du stagiaire)
   - nom (nom complet)
   - email (adresse e-mail)
   - formation (intitulé de la formation suivie)

   Ajouter les constructeurs, getters, setters et toString().

   b) Créer le package com.formation.api.service

   Implémenter la classe StagiaireService.java avec les méthodes suivantes :

| Méthode | Description |
|---|---|
| Stagiaire getStagiaire(int id) | Retourne le stagiaire dont l'identifiant est id |
| List<Stagiaire> getStagiaires() | Retourne la liste de tous les stagiaires |
| void ajouterStagiaire(Stagiaire s) | Ajoute un nouveau stagiaire à la collection |
| boolean modifierEmail(int id, String nouvelEmail) | Met à jour l'e-mail du stagiaire id |
| boolean supprimerStagiaire(int id) | Supprime le stagiaire d'identifiant id |

   Utiliser une ArrayList ou une HashMap<Integer, Stagiaire> comme stockage en mémoire (pas de base de données requise pour cet exercice).

5. Créer la ressource REST : StagiaireResource.java

   Dans le package com.formation.api, créer la classe StagiaireResource.java avec les points de terminaison suivants :

| Méthode HTTP | Chemin | Format | Action |
|---|---|---|---|
| GET | /stagiaires | JSON | Liste tous les stagiaires |
| GET | /stagiaires/{id} | XML | Récupère un stagiaire par son identifiant |
| POST | /stagiaires/ajout | JSON | Ajoute un nouveau stagiaire (corps de la requête = objet JSON) |
| PUT | /stagiaires/email/{id} | JSON | Met à jour l'e-mail du stagiaire id (envoyer {"email": "nouveau@mail.com"}) |
| DELETE | /stagiaires/supprimer/{id} | – | Supprime le stagiaire d'identifiant id |

6. Compiler et générer le .war (dans le terminal de VS Code : mvn clean package )
7. Tester les differentes ressources avec avec l'outil Postman:

   GET http://localhost:8081/GestionStagiaires/api/stagiaires

   POST http://localhost:8081/GestionStagiaires/api/stagiaires/ajout

   DELETE http://localhost:8081/GestionStagiaires/api/stagiaires/supprimer/101