



# Chapitre 3

## Outil de gestion de projet et d'automatisation de la construction (build)



Enseignante: Dr-Ing. Amina JARRAYA  
Email : [amina.jarraya@ensi-uma.tn](mailto:amina.jarraya@ensi-uma.tn)  
Niveau: I13- GL

# Plan

1. Introduction
2. Présentation du Maven
  - Les éléments clés d'une architecture maven
  - Le fichier POM.xml
  - Les dépôts de Maven
  - Les commandes Maven
3. Installation Maven

# 1. Introduction

# C'est quoi l'automatisation de la construction ?

- Un outil de gestion de projet et d'automatisation de la construction (build) est une solution logicielle conçue pour faciliter la planification, l'organisation, l'exécution et le suivi de projets de construction, tout en intégrant des fonctionnalités d'automatisation pour simplifier les tâches répétitives et accélérer les processus



Compiling  
Code



Packaging  
Binaries



Running  
Tests



Application  
Deployment

# Outils d'automatisation de construction des builds

- Il existe plusieurs outils qui permettent de gérer la fabrication de librairies/applications.

- Les 3 principaux sont :

- **Ant** (correspondant à Make en C) Plus vraiment utilisé



- **Maven** (spécification déclarative) Utilisé pour les projets simples

**Maven**

- **Gradle** (spécification en code) Utilisé pour les projets plus complexes (ex Android)



## 2. Présentation du Maven

# C'est quoi maven ?

- Développé par la fondation Apache, c'est un générateur de squelette pour des projets Java ou JEE (il simplifie le processus de construction)
- Aussi, un gestionnaire de dépendance



- **Automatisation de tâches récurrentes**
- **Construction, Compilation des projets**
- **Gestion des dépendances**



- **Génération des livrables**
- **Génération de la documentation et de rapports**
- **Déploiement d'applications**

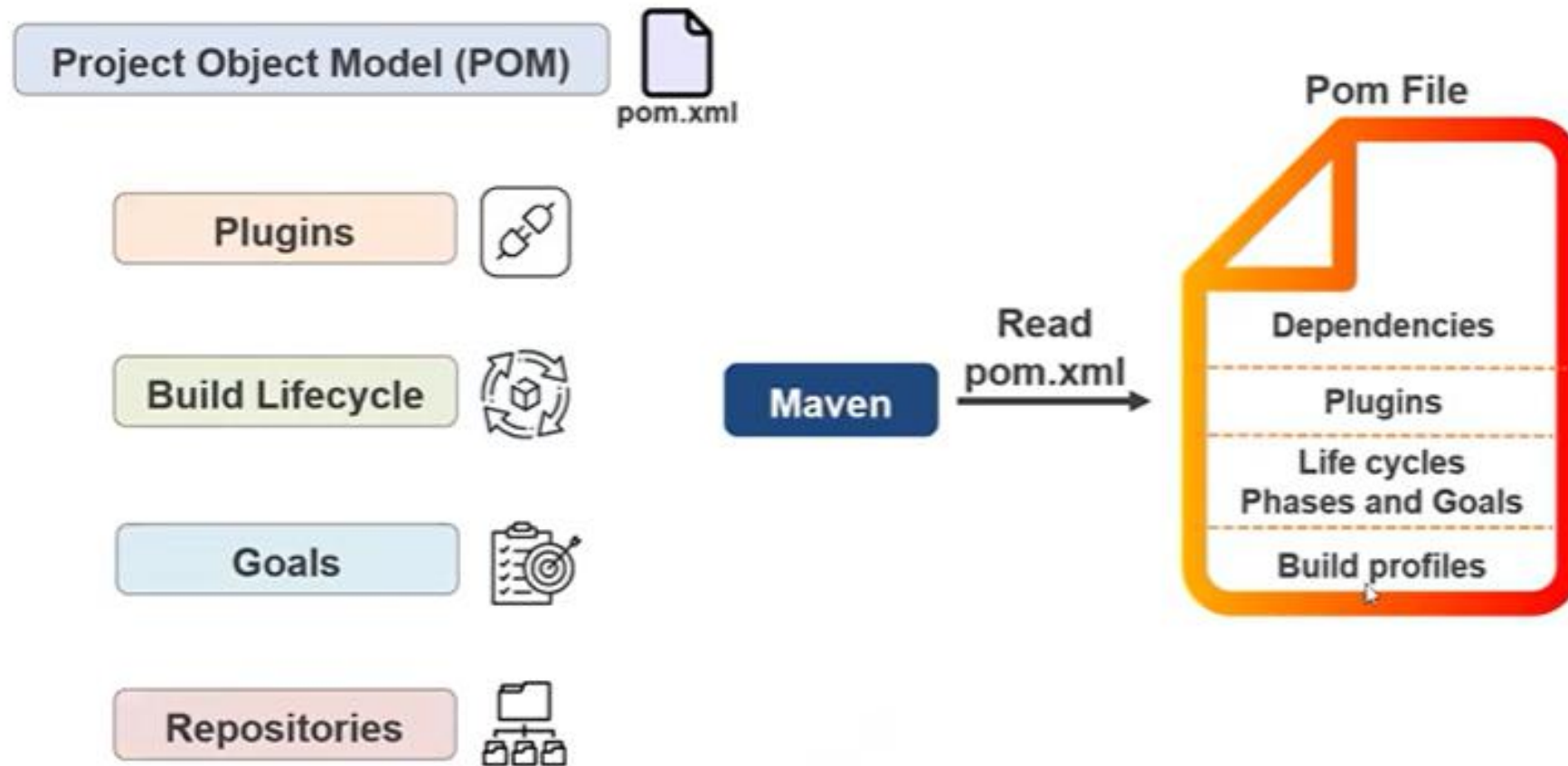


# Popularité de Maven vs Ant





# Les éléments clé d'une architecture Maven



# Comment il fonctionne ?

- Modèle de projet basé sur des conventions (**POM**)
- Utilisant un fichier de configuration **pom.xml** (pour Project Object Model)
- **POM (Project Model Object)**: Il s'agit d'un fichier XML qui contient des informations sur le projet et des détails de configuration utilisés par Maven pour construire le projet. Il contient une partie pour définir les dépendances utilisées dans un projet.



```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example.pom</groupId>
  <artifactId>sampleProject</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>sampleProject</name>
  <description>This is a sample maven project for adding dependencies.</description>

  <dependencies>
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-parent</artifactId>
      <version>3.0.8.RELEASE</version>
    </dependency>
  </dependencies>

</project>
```

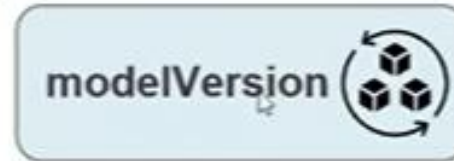
# Project Object Model (pom.xml)

Un fichier XML qui définit les **détails** du projet, les **configurations**, les **dépendances** et les **paramètres de compilation**



Maven utilise le fichier POM pour **compiler**, **gérer** et **exécuter** les tâches du projet.

Elements used for Creating pom.xml file:



# Les éléments clé de POM.xml

## Group ID

Identifie le groupe ou l'organisation qui a créé le projet. Il est généralement basé sur le nom de domaine de l'entreprise, par exemple **com.example.myproject**.

## Artifact ID

Désigne le nom spécifique du projet ou de l'artefact. Il est généralement court et descriptif, par exemple **my-library**

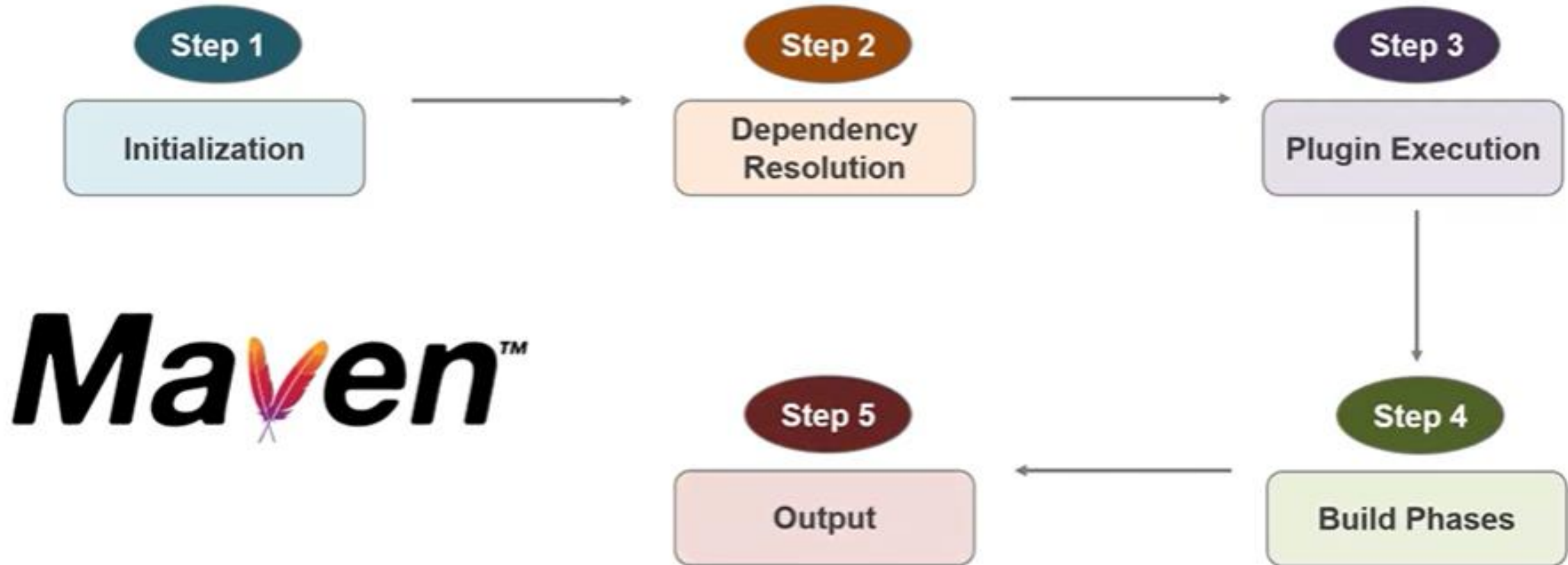
## Version

Indique la version du projet ou de l'artefact. Elle suit généralement un schéma de versionnement (par exemple, 1.0, 2.1.3, SNAPSHOT).

## Packaging

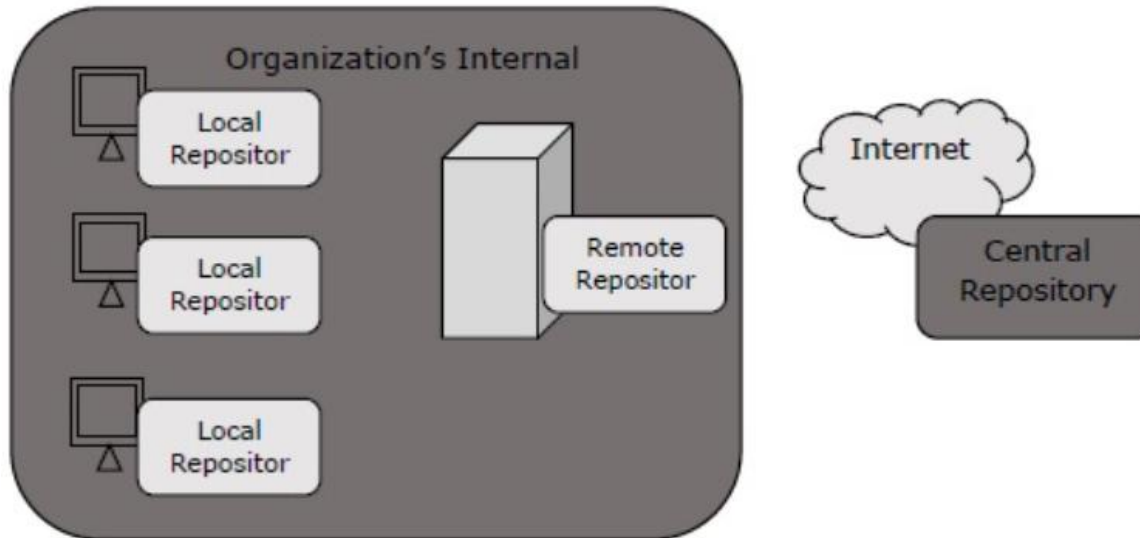
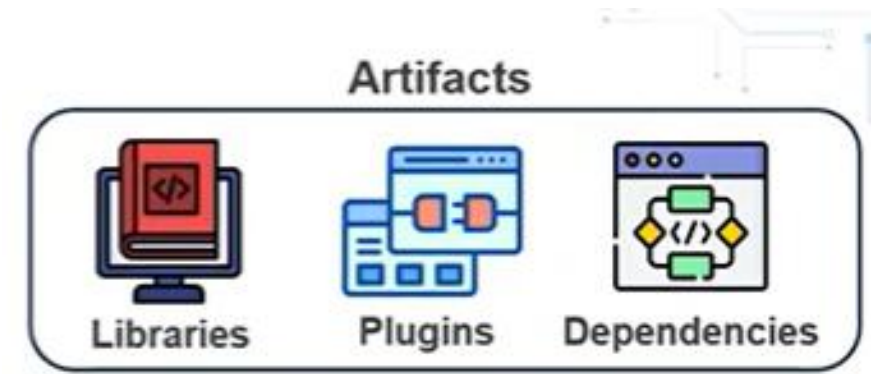
Spécifie le format de l'artefact produit par le projet. Les types de packaging les plus courants sont **jar, war, etc.** Par exemple, un jar est un package de classes Java, tandis qu'un war est un package pour les applications web.

# Le flux d'exécution de Maven



# Maven repositories

- En Maven, les dépôts sont des emplacements où les artefacts (bibliothèques, plugins, etc.) sont stockés et accessibles.
- Il existe trois types principaux de dépôts :
  - Local
  - Central
  - distant.



- Le dépôt local est un répertoire sur la machine de l'utilisateur, tandis que les dépôts central et distant sont situés sur des serveurs accessibles via le réseau.

# Maven Local Repository

- C'est un répertoire sur votre ordinateur où Maven stocke les artefacts téléchargés à partir des dépôts distants. Il sert de cache pour éviter de télécharger plusieurs fois les mêmes artefacts et permet de travailler hors connexion.

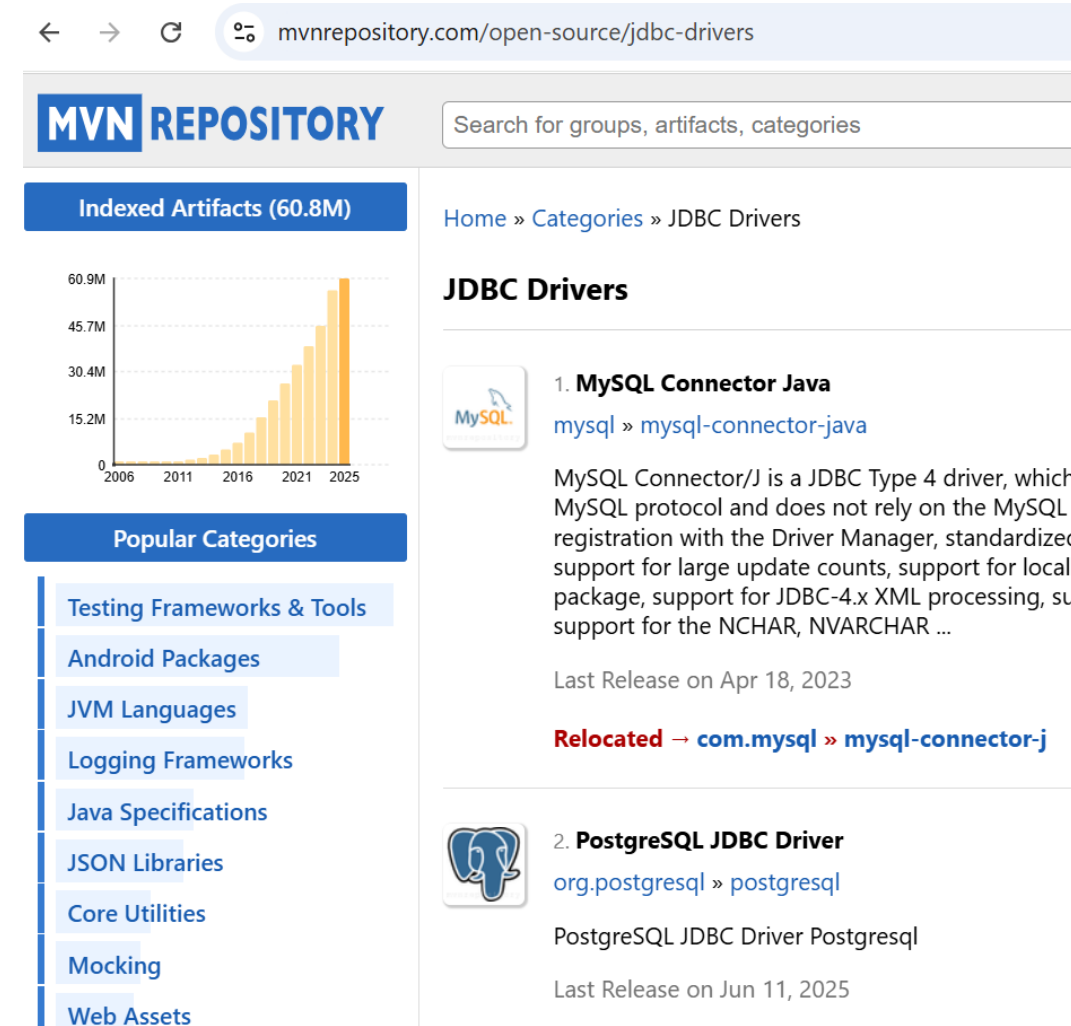


Ce PC > Disque local (C:) > Utilisateurs > HP-EliteBook > .m2 > repository >			
Trier Afficher ...			
Nom	Modifié le	Type	Taille
.locks	02/05/2025 15:05	Dossier de fichiers	
antlr	27/02/2025 12:06	Dossier de fichiers	
aopalliance	30/04/2025 18:39	Dossier de fichiers	
backport-util-concurrent	30/01/2025 08:39	Dossier de fichiers	
ch	24/04/2025 14:15	Dossier de fichiers	
com	24/04/2025 14:15	Dossier de fichiers	
commons-beanutils	30/04/2025 18:09	Dossier de fichiers	
commons-chain	30/04/2025 18:09	Dossier de fichiers	
commons-codec	30/01/2025 08:39	Dossier de fichiers	
commons-collections	30/04/2025 18:09	Dossier de fichiers	



# Maven central repository

- C'est le dépôt principal géré par la communauté Maven, hébergé sur Maven Central. Il contient un large éventail de bibliothèques et de dépendances couramment utilisées.
- <https://mvnrepository.com/>





The screenshot shows the Maven Repository website at the URL [mvnrepository.com/open-source/jdbc-drivers](https://mvnrepository.com/open-source/jdbc-drivers). The page features a search bar at the top with the text "Search for groups, artifacts, categories". Below the search bar, there is a section titled "Indexed Artifacts (60.8M)" with a bar chart showing the growth of indexed artifacts from 2006 to 2025. The chart shows a steady increase, with a significant jump around 2021. Below the chart, there is a section titled "Popular Categories" with a list of categories: Testing Frameworks & Tools, Android Packages, JVM Languages, Logging Frameworks, Java Specifications, JSON Libraries, Core Utilities, Mocking, and Web Assets. The main content area is titled "JDBC Drivers" and lists two items: 1. MySQL Connector Java and 2. PostgreSQL JDBC Driver. The MySQL Connector Java entry includes a description, a link to the MySQL website, and a note about its relocation to com.mysql:mysql-connector-j. The PostgreSQL JDBC Driver entry includes a link to the PostgreSQL website and a note about its last release date.

← → ↺ mvnrepository.com/open-source/jdbc-drivers

**MVN REPOSITORY** Search for groups, artifacts, categories

Home » Categories » JDBC Drivers

### JDBC Drivers

-  **1. MySQL Connector Java**  
[mysql](#) » [mysql-connector-java](#)  
MySQL Connector/J is a JDBC Type 4 driver, which MySQL protocol and does not rely on the MySQL registration with the Driver Manager, standardized support for large update counts, support for local package, support for JDBC-4.x XML processing, support for the NCHAR, NVARCHAR ...  
Last Release on Apr 18, 2023  
**Relocated** → [com.mysql](#) » [mysql-connector-j](#)
-  **2. PostgreSQL JDBC Driver**  
[org.postgresql](#) » [postgresql](#)  
PostgreSQL JDBC Driver Postgresql  
Last Release on Jun 11, 2025

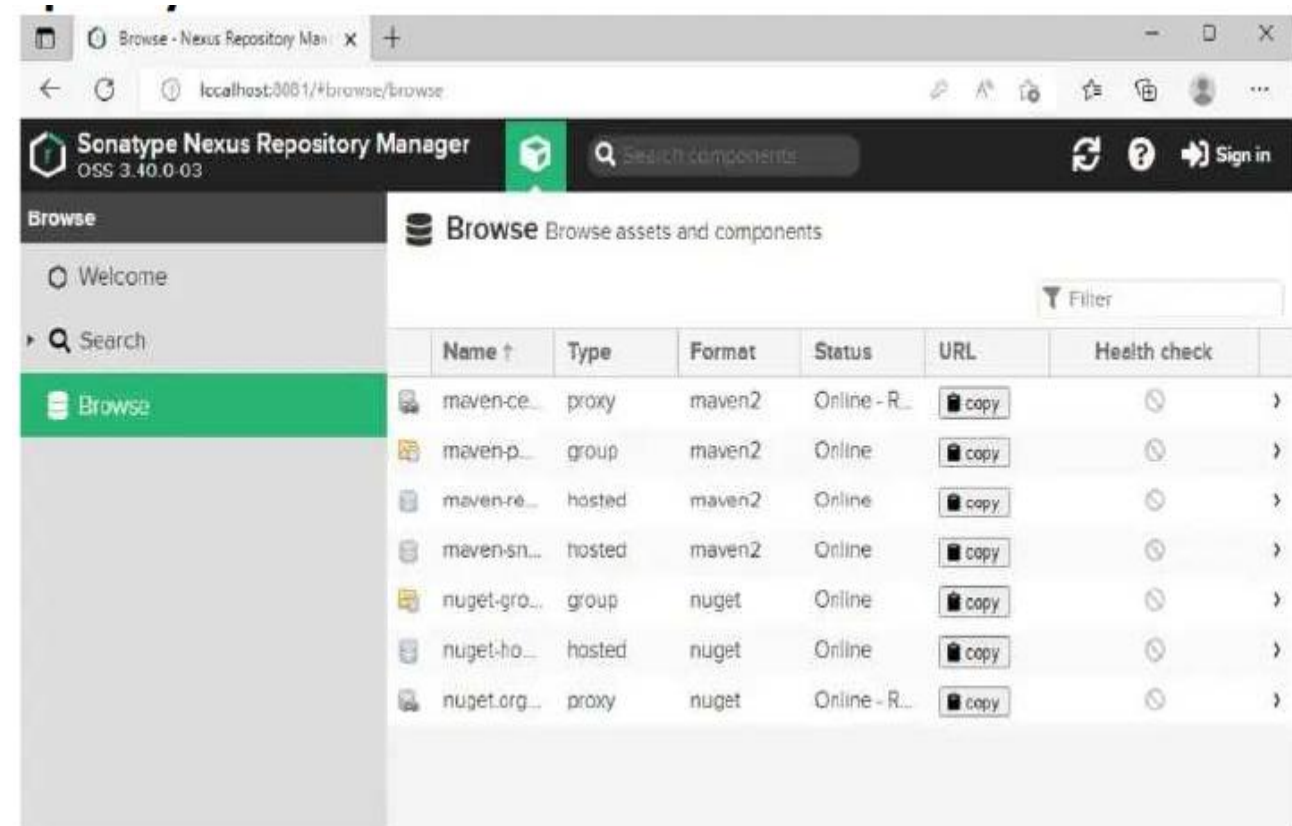


# Maven remote repository

- Ce sont des dépôts gérés par des organisations ou des tiers, qui peuvent contenir des artefacts spécifiques à leur domaine ou des versions personnalisées d'artefacts. Ils sont accessibles via des URL et sont spécifiés dans le fichier pom.xml du projet.

## Sonatype Nexus repository

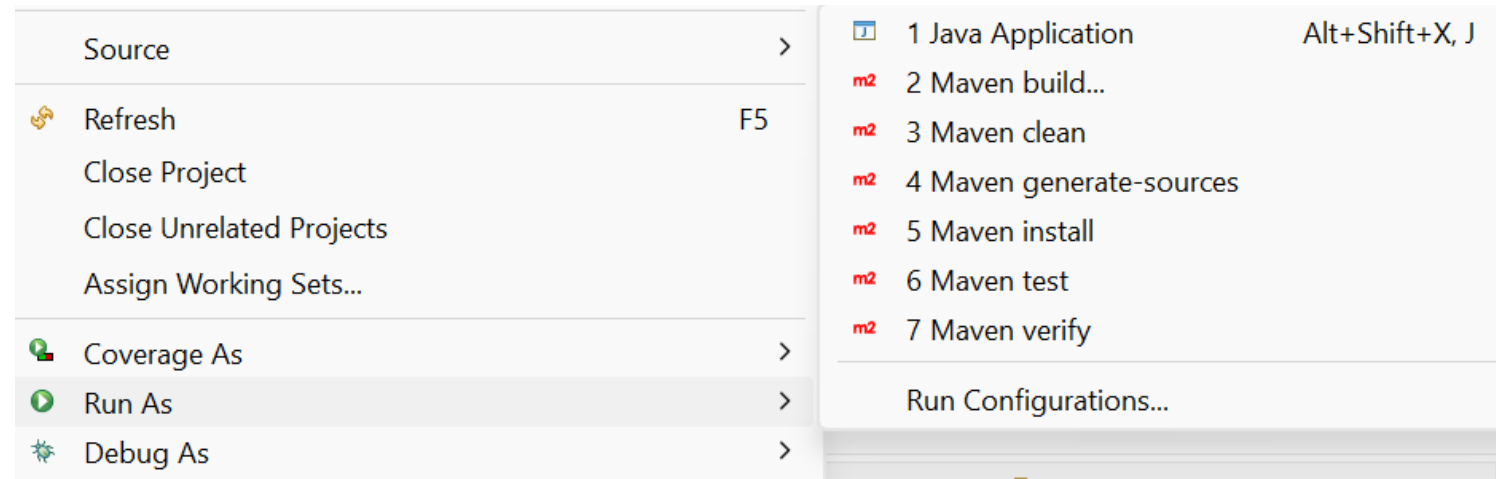
- Un dépôt Nexus est un serveur qui gère les dépôts d'artefacts logiciels, notamment pour Maven, mais aussi pour d'autres outils comme npm, Docker, etc.
- C'est un dépôt distant, ou un proxy, qui permet de stocker et de partager des artefacts (bibliothèques, binaires, etc.) entre différents projets et équipes de développement.



# Les commandes de Maven

- **mvn compile** : Créer les .class
- **mvn test** : Jouer les tests unitaires
- **mvn package** : Création du livrable dans target.
- **mvn install** : Copie du livrable dans le Repository local : `~\.m2\repository\...`
- **mvn deploy** : Copie du livrable sur le repository distant (e.g, nexus repository)
- **mvn clean** : Supprime le contenu du dossier target.
- Emplacement du livrable :
- `{emplacement Repository}/groupId/artifactId/version`
- Nom du package (jar en général) : `{artifactId}-{version}.{package}`

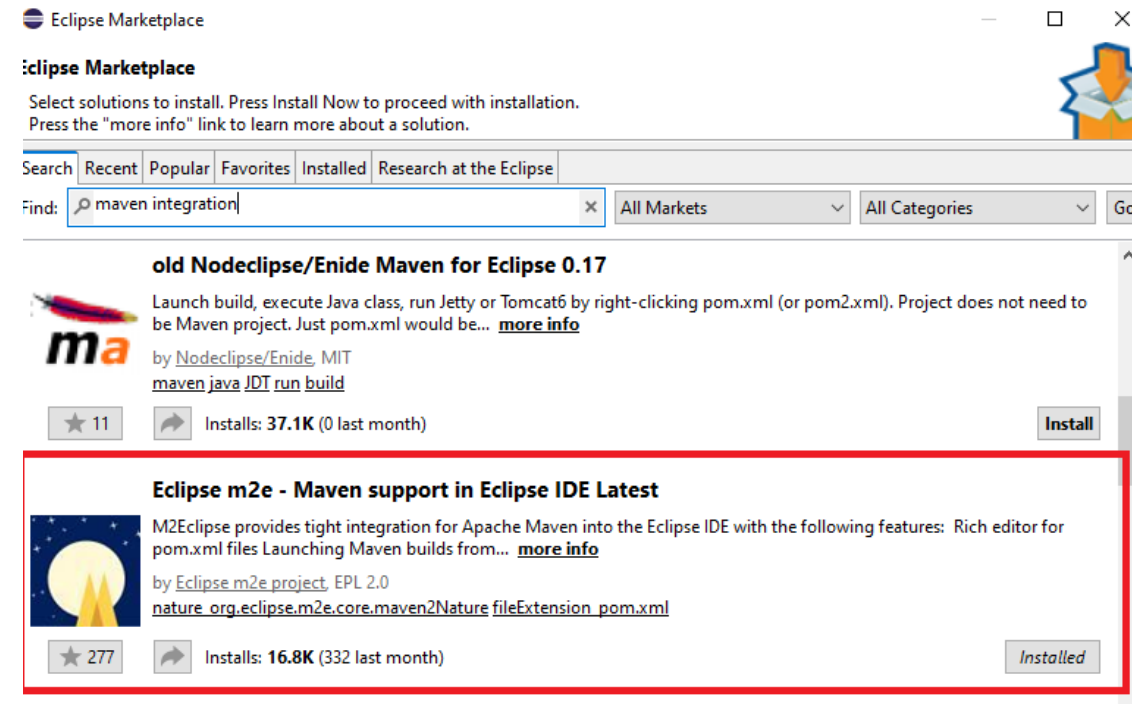
```
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.17</version>
</dependency>
```



# 1. Installation de Maven

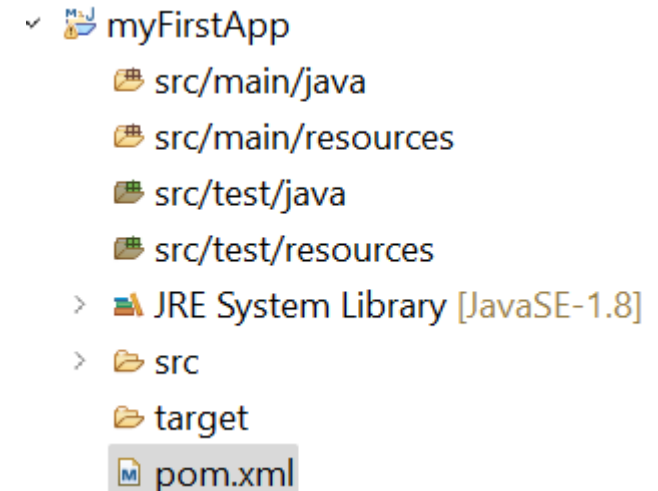
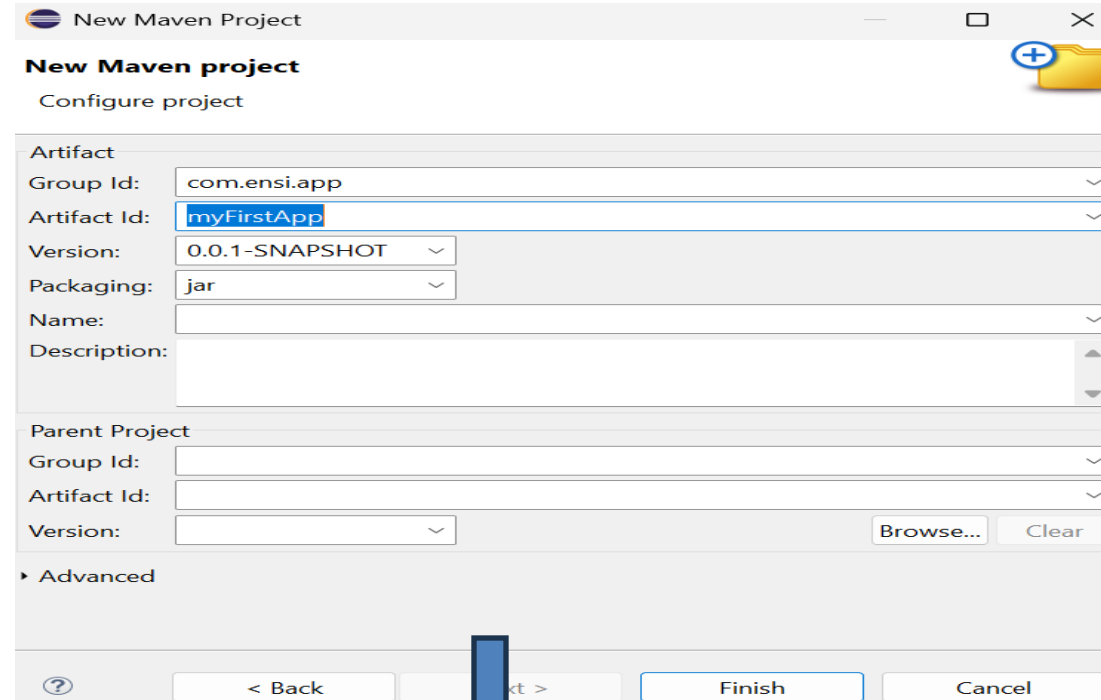
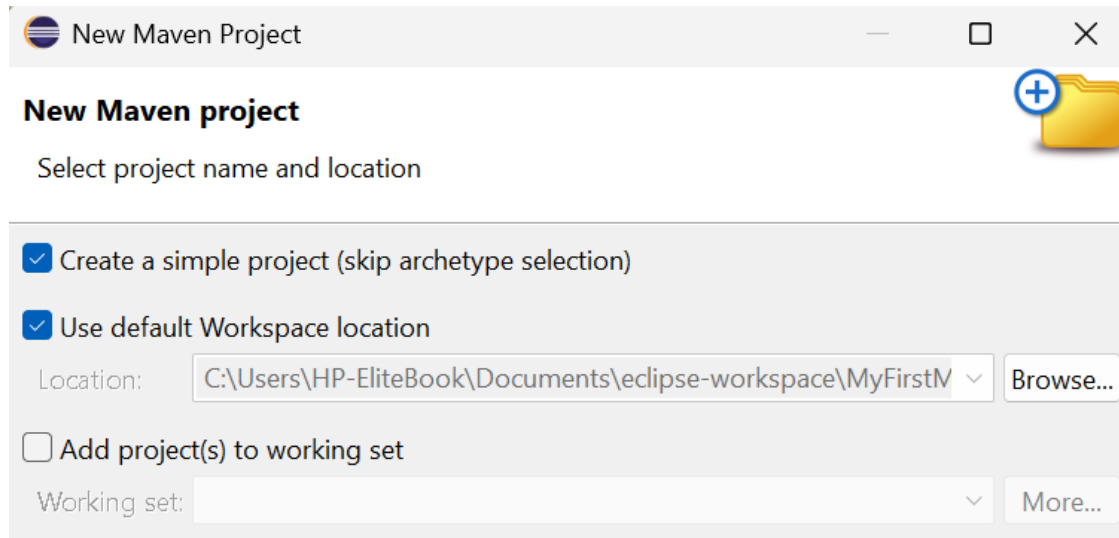
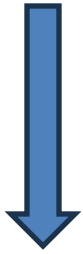
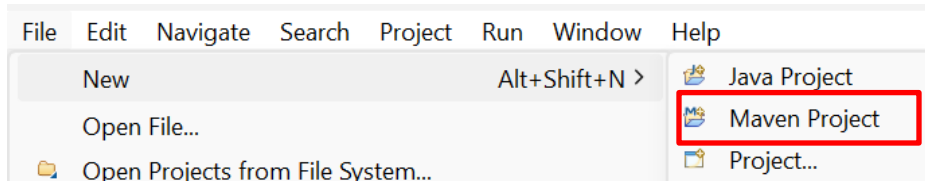
# Comment installer Maven ?

- 2 façons pour installer «Maven »
  - Maven en tant que **Plugin** : Il est déjà installé par défaut dans Eclipse/IntelliJ.
- Maven en **standalone** à travers l'invite de commandes :  
<https://maven.apache.org/download.cgi>



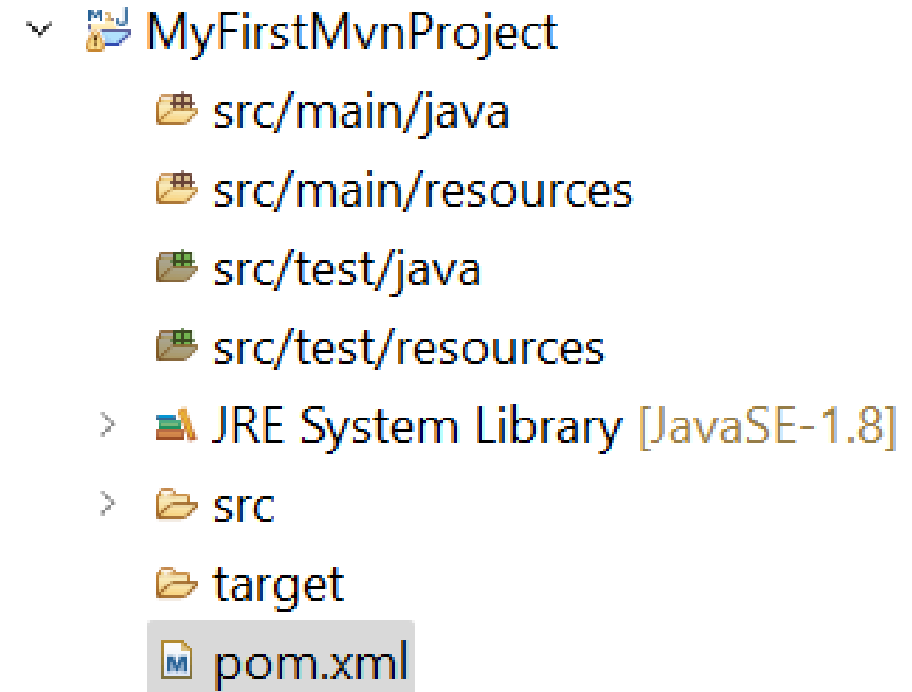
```
C:\Users\HP-EliteBook>mvn -version
Apache Maven 4.0.0-rc-3 (3952d00ce65df6753b63a51e86b1f626c55a8df2)
Maven home: C:\Users\HP-EliteBook\apache-maven-4.0.0-rc-3
Java version: 21.0.7, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-21
Default locale: fr_FR, platform encoding: UTF-8
OS name: "windows 11", version: "10.0", arch: "amd64", family: "winnt"
```

# Création d'un nouveau projet maven sous eclipse



# Structure d'un projet maven

- Un projet Maven sur disque a toujours la même structure :
  - pom.xml (fichier de configuration de Maven)
  - src/main/java : Fichiers sources
  - src/main/resources : Fichiers textes, images, associés
  - src/test/java : Fichier des tests JUnit ou TestNG
  - src/test/resources : Fichiers textes, etc pour les tests



# Le cycle de vie simplifiée

## Phases du cycle de vie (simplifiée)

- Compile : compile le code dans src/main/java
- Test-compile : compile le code dans src/test/java
- Test : execute les tests
- Package : génère le jar
- Install : installe dans le repo local
- Deploy : upload sur maven central

## mvn permet d'exécuter Maven

```
mvn --version
```

```
mvn test (compile et test)
```

```
mvn package (compile, test et package)
```

## si on veut sauter les tests

```
mvn -Dmaven.test.skip (compile pas les tests)
```

```
mvn -DskipTests (n'exécute pas les tests)
```

## 4. Atelier Maven

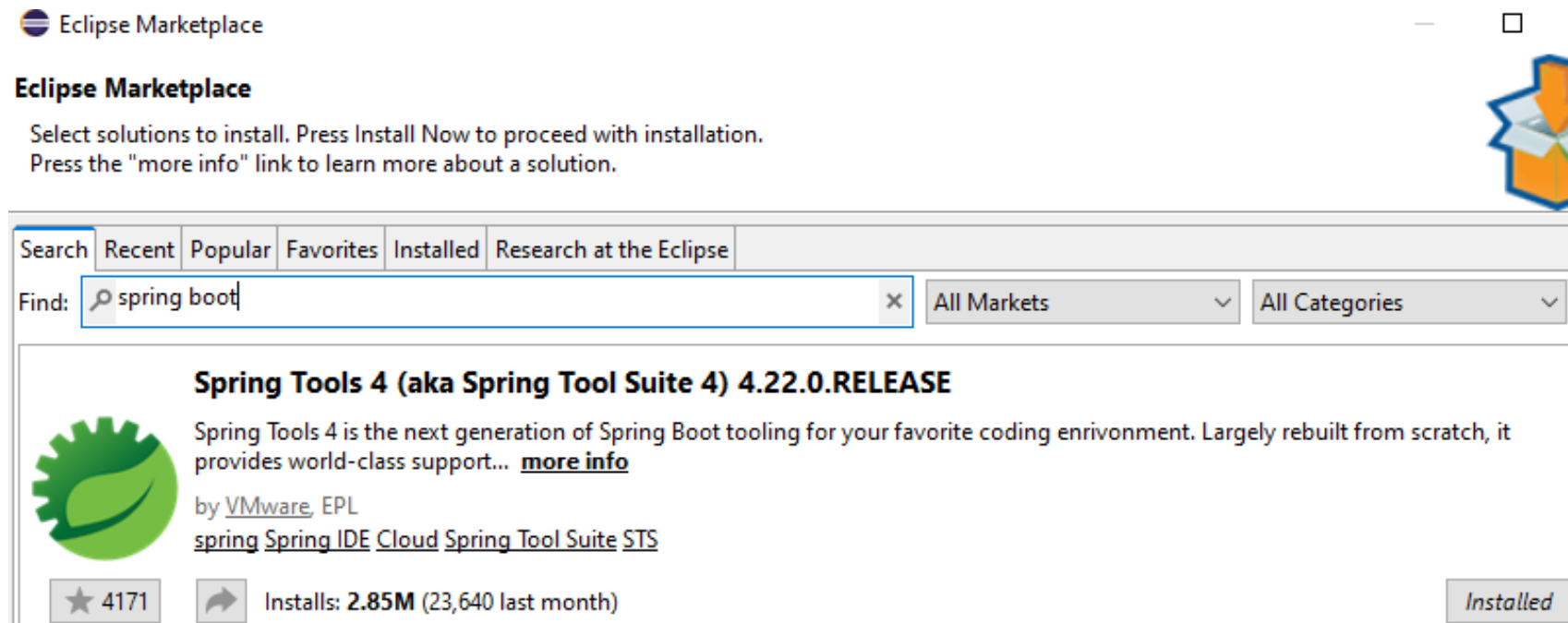


# Mise en place d'un micro-service

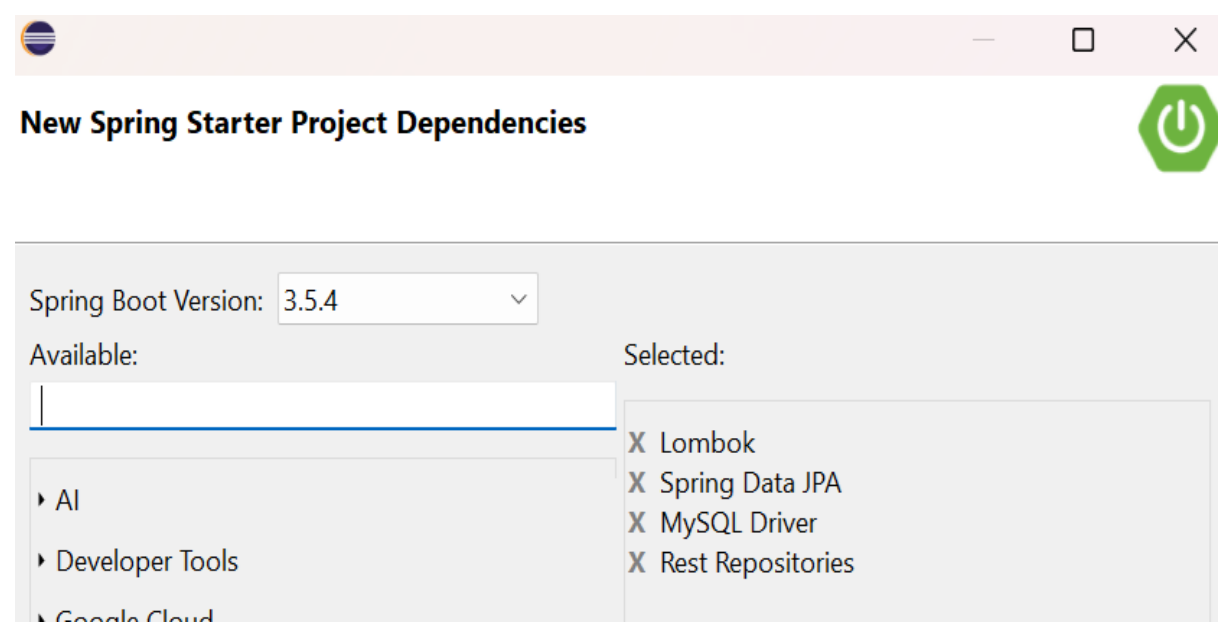
- Cet atelier présente la création d'un simple micro service «**country-service**» en utilisant le framework **Spring Boot**.
- L'objectif de cet atelier est de:
  - Créer un micro service
  - Explorer Spring Boot
  - Créer une base de données **MySQL**
  - Créer un Rest Controller
  - Tester l'API grâce à **Swagger-ui**
  - Lancer les commandes **Maven** pour compiler et générer le fichier jar
- Le micro-service « **country-service** » permet de :
  - Rajouter un pays
  - Récupérer tous les pays
  - Récupérer le pays via son id
  - Récupérer le pays via son nom
  - Supprimer un pays
  - Mettre à jour un pays

# Prérequis

- Cet atelier nécessite la mise en place du plugin Spring Boot.
- Ouvrir Eclipse IDE -> aller vers Help -> cliquer Eclipse Marketplace
- Chercher “spring boot”, installer le puis redémarrer votre Eclipse IDE.



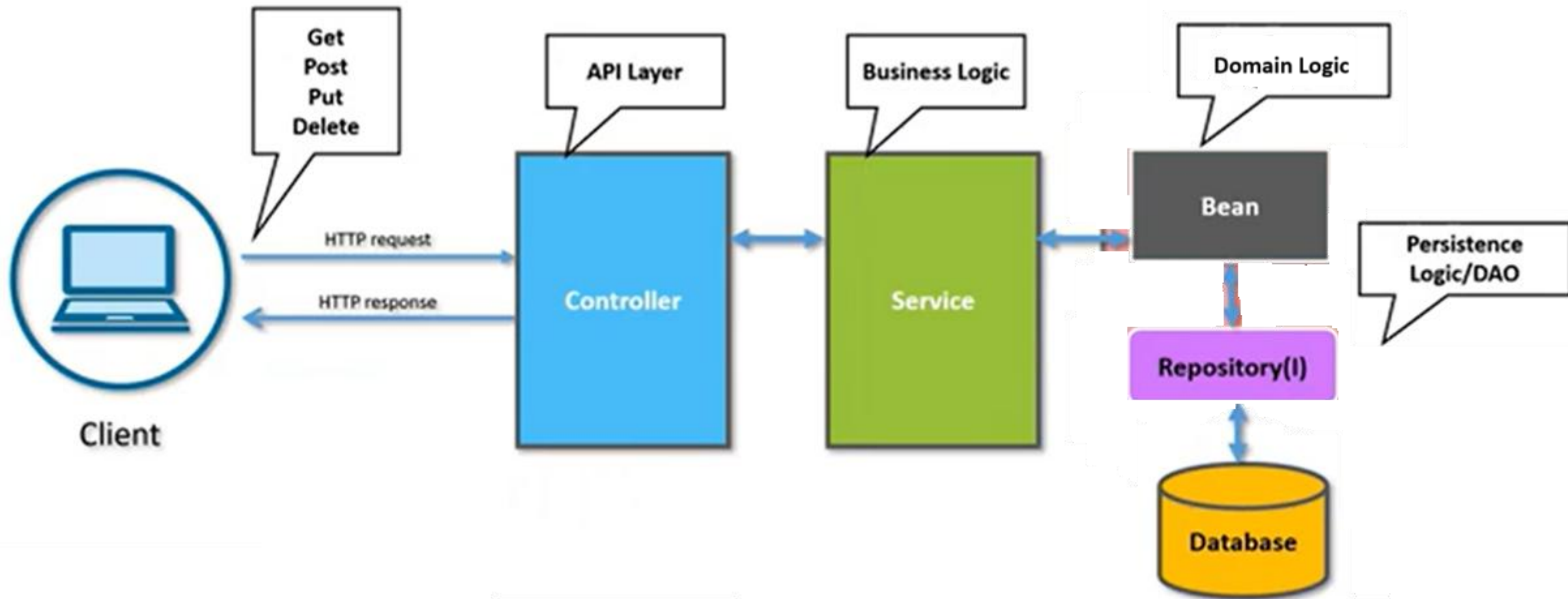
# Les dépendances Maven



- Les dépendances à rajouter dans le fichier POM.xml de votre projet spring boot : *Lombok, Spring Data JPA, MySQLDriver, Rest Repositories*
- Afin d'utiliser swagger ui, il faut rajouter cette dépendance dans le fichier POM.xml

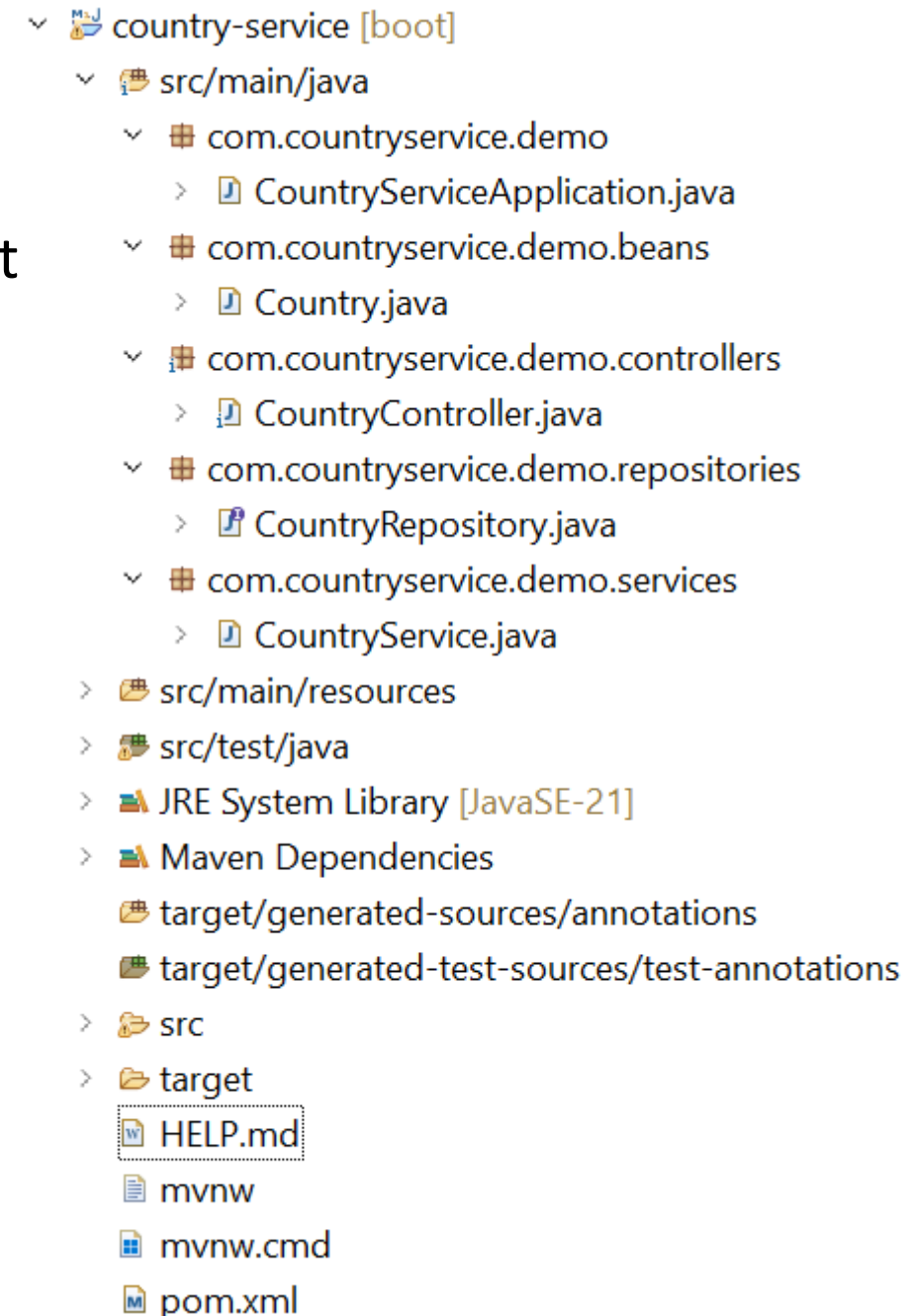
```
<dependency>  
<groupId>org.springdoc</groupId>  
<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>  
<version>2.8.6</version> <!-- Check for the latest version -->  
</dependency>
```

# Architecture logique de l'application



# Structure du projet

- Vous devez avoir la structure suivante pour votre projet
  - La classe **CountryController.java** implémente la couche Contrôleurs
  - La classe **CountryService.java** implémente la couche Services
  - L'interface **CountryRepository.java** implémente la couche de persistance (couche DAO)
  - La classe **Country.java** implémente l'entité métier



# Travail à faire

- Vous devez implémenter toutes les classes à savoir :
  - La classe **CountryController.java**
  - La classe **CountryService.java**
  - L'interface **CountryRepository.java**
  - La classe **Country.java**
- Vous devez avoir les services suivants dans l'interface swagger ui :

country-controller	
PUT	/updatecountry/{id}
POST	/addcountry
GET	/getcountries
GET	/getcountries/{id}
GET	/getcountries/countryname
DELETE	/deletecountry/{id}

- country-service [boot]
  - src/main/java
    - com.countryservice.demo
      - CountryServiceApplication.java
    - com.countryservice.demo.beans
      - Country.java
    - com.countryservice.demo.controllers
      - CountryController.java
    - com.countryservice.demo.repositories
      - CountryRepository.java
    - com.countryservice.demo.services
      - CountryService.java
  - src/main/resources
  - src/test/java
  - JRE System Library [JavaSE-21]
  - Maven Dependencies
    - target/generated-sources/annotations
    - target/generated-test-sources/test-annotations
  - src
  - target
    - HELP.md
    - mvnw
    - mvnw.cmd
    - pom.xml

# Exemples

- La classe **CountryController.java** : voici un exemple d'implémentation de la méthode **getCountries()** qui permet de récupérer tous les pays à partir du service

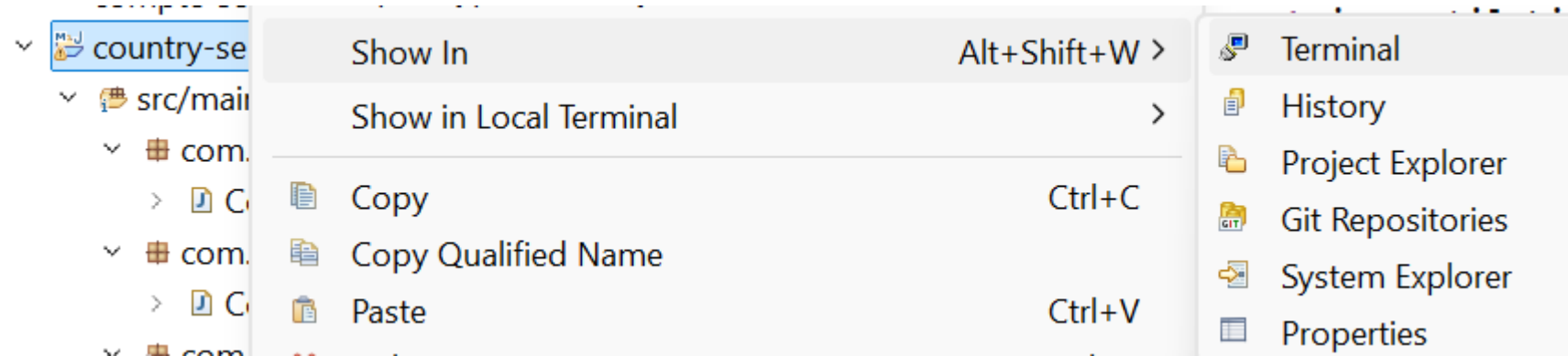
```
@GetMapping("/getcountries")
public ResponseEntity <List <Country>> getCountries() {
    try {
        List <Country> countries = countryservice.getAllCountries();
        return new ResponseEntity <List <Country>>(countries, HttpStatus.FOUND);
    }
    catch (Exception e) {
        return new ResponseEntity <>(HttpStatus.NOT_FOUND);
    }
}
```

- La classe **CountryService.java** : voici un exemple d'implémentation de la méthode **getAllCountries()** qui permet de récupérer tous les pays à partir du repository

```
public List <Country> getAllCountries() {
    List <Country> countries = countryRep.findAll();
    return countries;
}
```

# Commandes de Maven

- Ouvrez le terminal sous le projet « country-service » comme suit :



- Essayez de compiler le projet et de générer le fichier jar en utilisant les commandes Maven
- Vérifiez la génération du fichier jar sous le dossier **target** :
- Pour exécuter le fichier jar, il suffit de taper :



```
java -jar compte-service-0.0.1-SNAPSHOT.jar
```



- *"Apprendre par le projet, c'est découvrir par l'action, créer par la compréhension, et réussir par la persévérance."*



[amina.jarraya@ensi-uma.tn](mailto:amina.jarraya@ensi-uma.tn)

**ENSI Manouba**