École Doctorale en Sciences Economique, Gestion et Informatique
Faculté des Sciences Économiques et de Gestion de Sfax
Département Informatique

Auditoire

Mastère Professionnel Audit et Sécurité Informatique

Architectures des Systèmes d'Information (JavaEE)

Enseignante

AÏDA KHEMAKHEM

Année Universitaire

2019 – 2020

Introduction générale

Qu’est ce que la plateforme Java EE

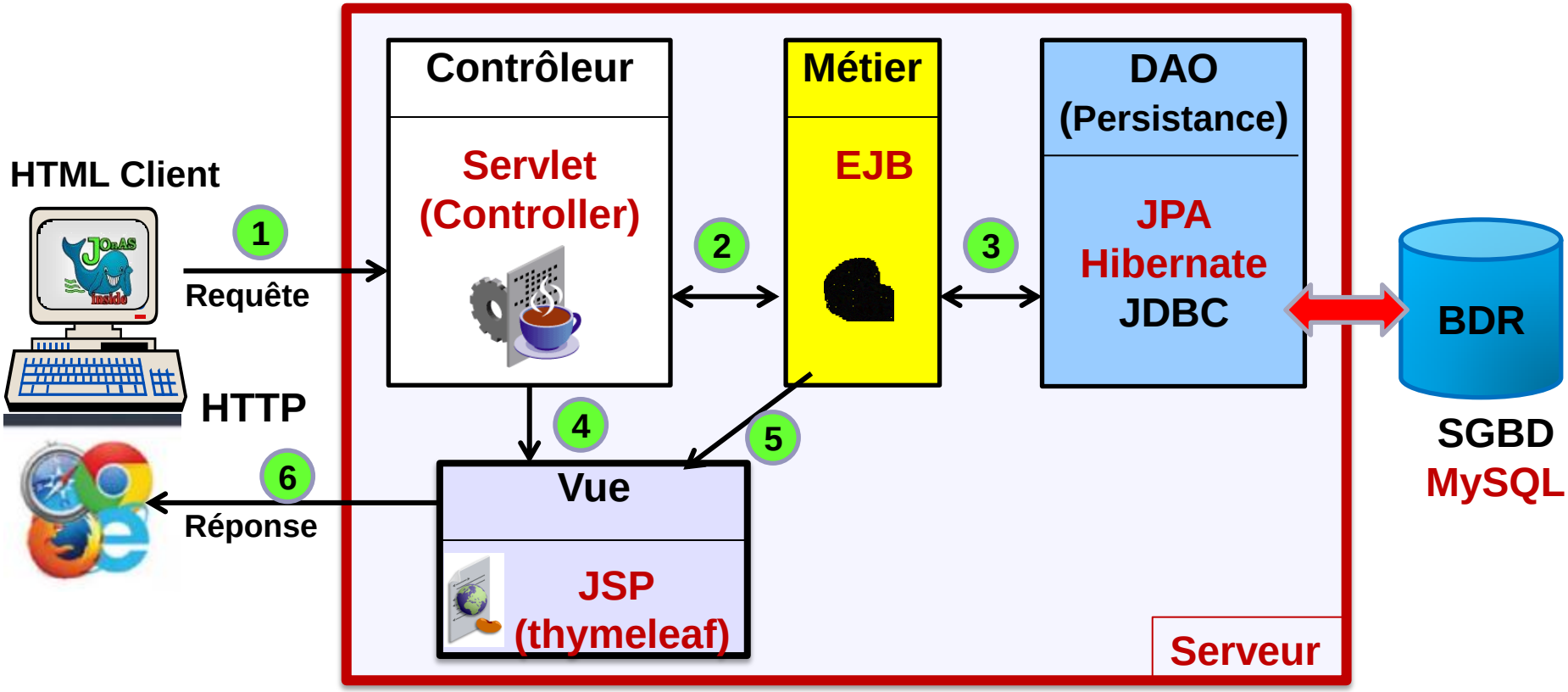
- Le terme « Java EE » signifie *Java Enterprise Edition* et fait référence à une extension de la plate-forme standard
- La plateforme Java EE est construite sur le langage Java et la plate-forme Java Standard Edition, et elle y ajoute un grand nombre de bibliothèques remplissant des fonctionnalités que la plate-forme standard ne remplit pas d’origine

L’objectif majeur de Java EE est de faciliter le développement d’applications web robustes et distribuées, déployées et exécutées sur un serveur d’applications

Pourquoi utiliser un FrameWork ?

- Un ***framework*** est un ensemble de composants qui servent à créer l'architecture et les grandes lignes d'une application
- Peut être assimilé à une boîte à outils géante, conçue pour faciliter le travail des développeurs
- Il existe des *frameworks* dans beaucoup de langages et plate-formes

Architecture JavaEE



1 - Le client envoie la requête au contrôleur
2 - Le contrôleur stocke les données avec Beans puis faire les traitements
3 - Au niveau métier assure la persistance des données en utilisant DAO
4 - Le contrôleur fait un forword vers la vue JSP
5 - La vue récupère les résultats
6 - La vue envoie le résultat au client

Plan

- Couche Métier : EJB (Entreprise JavaBean)
- Couche DAO : JPA (Java Persistance API)
- Couche Contrôleur : Controller (Servlet)
- Couche Vue : Thymleaf (JSP : Java Servet Page)
- FrameWork : Spring Boot
- Atelier de développement : Spring Tool Suite….
- Installation et configuration
- Maven
- Structure d’un projet : Spring Starter Project

I. Couche Métier

- Elle implémente la logique métier d'une entreprise
- Elle se charge de récupérer, à partir de différentes sources de données, les données nécessaires pour assurer les traitement métiers déclenchés par le Contrôleur
- Il est cependant important de séparer la partie accès aux données (couche DAO) de la partie traitement de la logique métier (Couche métier) pour les raisons suivantes :
  - La couche métier est souvent stable. Alors que la couche DAO n'est pas stable : il arrive souvent de changer le SGBD ou répartir et distribués les BD
  - Faciliter la répartition des tâches entre les équipes de développement
  - Déléguer la couche DAO aux frameworks spécialisés dans l'accès aux données comme : Hibernate, Toplink….

I.1. les composants métier

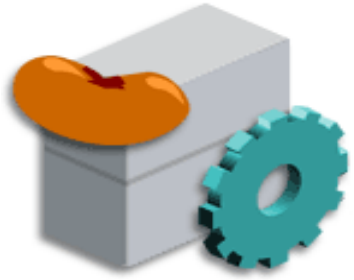
Les composants du niveau métier :

- Sont des composants EJB
- Gèrent la logique applicative
- Reçoivent les données provenant des programmes client
- Extraient les données de la base
- Traitent les données et communiquent avec la base et le programme client
- Peuvent être appelés par les composants des autres couches

I.2. Enterprise JavaBeans (EJB)

Les caractéristiques de EJB :

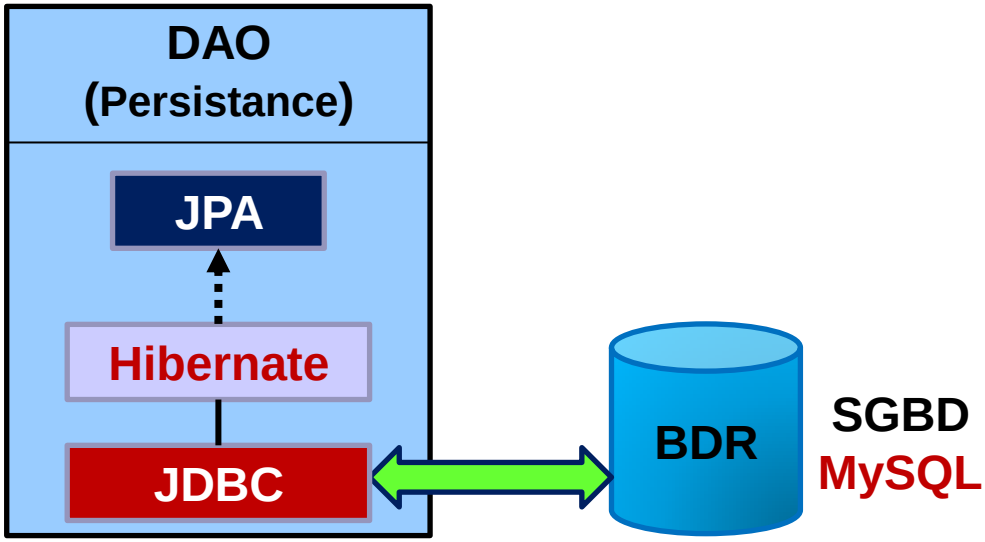
- Sont des composants côté serveur écrits en langage Java
- Contiennent la **logique applicative** d'une application d'entreprise
- Sont hébergés dans des **conteneurs** EJB
- Sont basés sur la communication RMI (Remote Method Invocation)
- Sont **indépendants** de la plate-forme
- Fournissent des **services distants** aux clients
- Peuvent être exposés en tant que services Web
- Utilisent JDBC pour la connexion à une base de données



II. Couche DAO : JPA - Hibernate - JDBC

- Cette couche assure l'enregistrement des données de l'application sur un support
- Cette couche représente l'intelligence de l'application. Elle est composée d'un ensemble d'interfaces locales (local) et distantes (remote).
- Les DAO (Data Access Object) permettent d'accéder aux objets et proposent des méthodes de CRUD (Create, Read, Update, Delete).

JPA - Hibernate - JDBC



JDBC (Java Data Base Connectivity) : Pilotes java pour permettre l'accès aux bases de données

JPA (Java Persistence API). Interface qui permet de gérer la persistance des EJB Entity. Le serveur d'application fournie un Framework implémentant la spécification JPA pour assurer le mapping objet relationnel (Pour JBOSS c'est Hibernate qui est utilisé)

Hibernate : assure l'implémentation des spécification de JPA pour assure la persistance des données tout en assurant le Mapping Objet Relationnel

III. Couche de contrôleur : Servlet

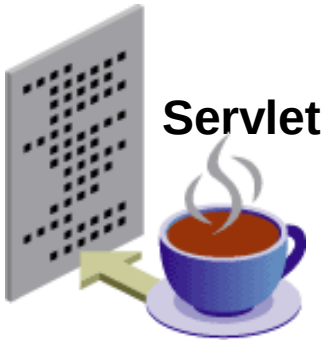
- Le contrôleur est chargé pour choisir le traitement à déclencher et des informations à afficher en fonction des entrées

- Une servlet est un objet qui permet d'intercepter les **requêtes** faites par un client, et qui peut personnaliser une réponse en conséquence. Il fournit pour cela des méthodes permettant de scruter les requêtes HTTP

- Cet objet n'agit jamais directement sur les données, il faut le voir comme un simple aiguilleur : il intercepte une requête issue d'un client, appelle éventuellement des traitements effectués par le modèle, et ordonne en retour à la vue d'afficher le résultat au client

III.1. Qu'est-ce qu'une servlet ?

Navigateur



Servlet

Informations client (nom d'hôte, données de panneau)

Demande

Traitement des résultats (accès à la base de données)

Mise en forme des résultats et création des pages HTML

Renvoi des pages au client

Réponse

Réussite ou échec

III.2. Présentation d'une servlet

- C'est une classe Java compilée (.class) qui s'exécute dans un conteneur de Servlets (moteur), hébergé par le serveur Web (côté serveur)
- Permet d‘étendre les fonctionnalités du serveur Web de manière portable et efficace
- Fonctionnement d’une servlet :
  - **reçoit** des requêtes HTTP,
  - **génère** une réponse éventuellement en utilisant la logique métier contenue dans des EJBs ou en interrogeant directement une base de données
  - **retourne** enfin une réponse HTML ou autre au demandeur

IV. Couche Vue (Présentation)

- Couche vue est la partie d'interface graphique pour l'utilisateur
- Objectifs :
  - Créer et contrôler l'interface présentée à l'utilisateur
  - Afficher les données aux clients
  - Collecter les informations qu'il saisit
  - Valider ses actions
- Il y a plusieurs types de clients JavaEE :
  - Clients Web (clients **légers**) : Navigateur cote client affichant des pages générées cote serveur
  - Applets (clients **lourds**): application Java exécutée par le navigateur cote client
  - Applications clientes (clients **riches**)
    - Exécutées sur la machine cliente
    - Plus riche en terme d'interface utilisateurs : formulaires
    - riches et interface graphiques complexes (Swing, AWT, …)

Nous allons utiliser des clients légers avec Thymeleaf (comme JSP)

IV.1. Clients Web (légers)

- Un client léger est un navigateur web (Internet Explorer, Mozilla Firefox….)
- Ce type de client peut recevoir du
  - code html
  - ficher html ou
  - des page JSP (html et java)

- Une page JSP est destinée à la vue de l'utilisateur. Elle permet au concepteur de la page d'appeler de manière transparente des portions de code Java, **via des balises** et **expressions** ressemblant fortement aux balises de présentation html

IV.2. Java Server Page : JSP

- Une page JSP est un document de type texte qui inclut :
  - Du code **HTML**
  - Des **balises JSP**
  - Du **code Java** (notamment des appels de composants JavaBeans et de servlets)

- Elle sépare clairement la création du contenu de la logique de présentation
- Elle se concentre sur le développement rapide et la modification aisée de l'interface graphique
- Elle fournit une méthode de développement de servlets axée sur la présentation

IV.3. Pourquoi les JSP ?

- Les servlets sont très pratiques pour déclencher des traitements coté serveur suite à une requête HTTP
- Dans une servlet, on peut également générer une réponse HTTP en utilisant l'objet response. **Mais** , quant il s'agit de générer une page web complète, les Servlet ne sont pas pratique
- Il existe un autre moyen pour créer les servlet, c'est les page JSP
- La page JSP est très pratiques pour générer des pages web dynamique
- Une page JSP est une sorte de HTML dans laquelle, on peut écrire du code java
- Dans une application web JavaEE
  - Les **servlets** jouent le rôle du **contrôleur** de l'application pour les traitement dans le serveur
  - Les **JSP** joue le rôle des **vues** générées au serveur et affichées pour le client

V. FrameWork : Spring Boot

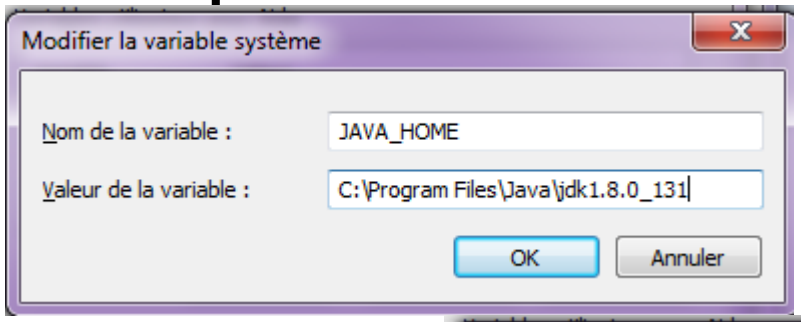
- Spring Boot est un Framework qui permet de créer des applications basées sur des micro services.
- Atouts de Spring Boot :
  - Faciliter le développement d'applications complexes.
  - Faciliter à l'extrême l'injection des dépendances
  - Réduire à l'extrême les fichier de configurations
  - Faciliter la gestion des dépendances Maven.
  - Auto Configuration : la plupart des beans sont créés si le ou les jar(s) adéquats sont dans le classpath.
  - Fournir un conteneur de servlet embarqué (Tomcat, Jetty)
  - Créer une application autonome (jar ou war)

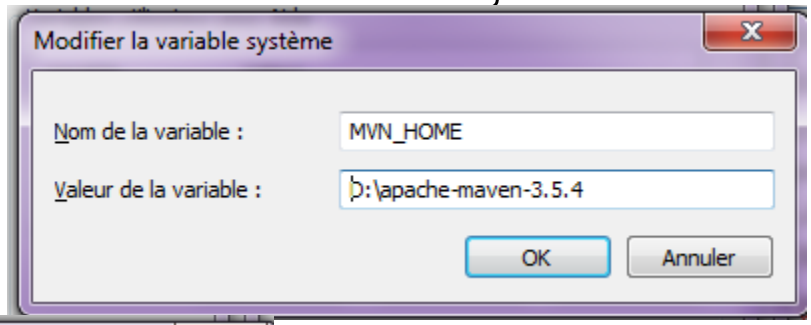
VI. Atelier de développement

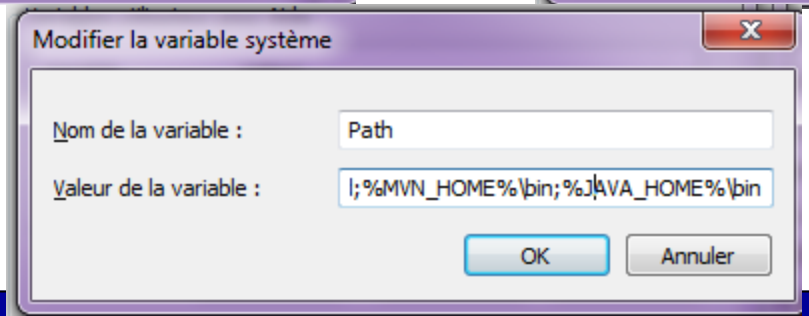
- **JDK 1.8 (**Java Developement Kit**)** ensemble d'outils et d'API pour permettre le développement de programmes avec Java. Il contient lui-même le JRE.
- **Maven** 2 : un outil de build
  - Pour développer des applications Java EE, nous allons avoir besoin de bibliothèques externes (les fichiers .jar en Java)
  - La configuration d'une application web en vue de son déploiement est maintenue dans un fichier XML, appelé descripteur de déploiement
- **Spring Tool Suite** (STS) : IDE intégre les pluing pour JavaEE
  - C'est un environnement de développement
  - Extension d'eclipse
  - Assure la compilation et l'exécution

VII. Installation et configuration

- Installer JDK 1.8
- Installer Maven 2
- Définir les variables d ’environnement (**Panneau de configuration\Système et sécurité\Système**)
  - **JAVA_HOME** = C:\Program Files\Java\jdk1.8.0
  - **MVN_HOME** = D:\apache-maven-3.1.1
  - **path** = %JAVA_HOME\bin;%MVN_HOME%\bin;…







VIII. Maven

- **Maven,** géré par l'organisation *Apache Software Foundation.* (*Jakarta Project),* est un **outil pour la gestion et l'automatisation de production des projets logiciels Java en général et Java EE en particulier**.
- L'objectif recherché est de
  - produire un logiciel à partir de ses sources,
  - en optimisant les tâches réalisées à cette fin
  - et en garantissant le bon ordre de fabrication.
    - **Compiler, Tester, Contrôler, produire les packages livrables**
    - **Publier la documentation et les rapports sur la qualité**
- Apports :
  - Simplification du processus de construction d'une application
  - Fournit les bonnes pratique de développement
  - Tend à uniformiser le processus de construction logiciel
  - Vérifier la qualité du code
  - Faciliter la maintenance d'un projet

VIII.1. Outils d'intégration continue

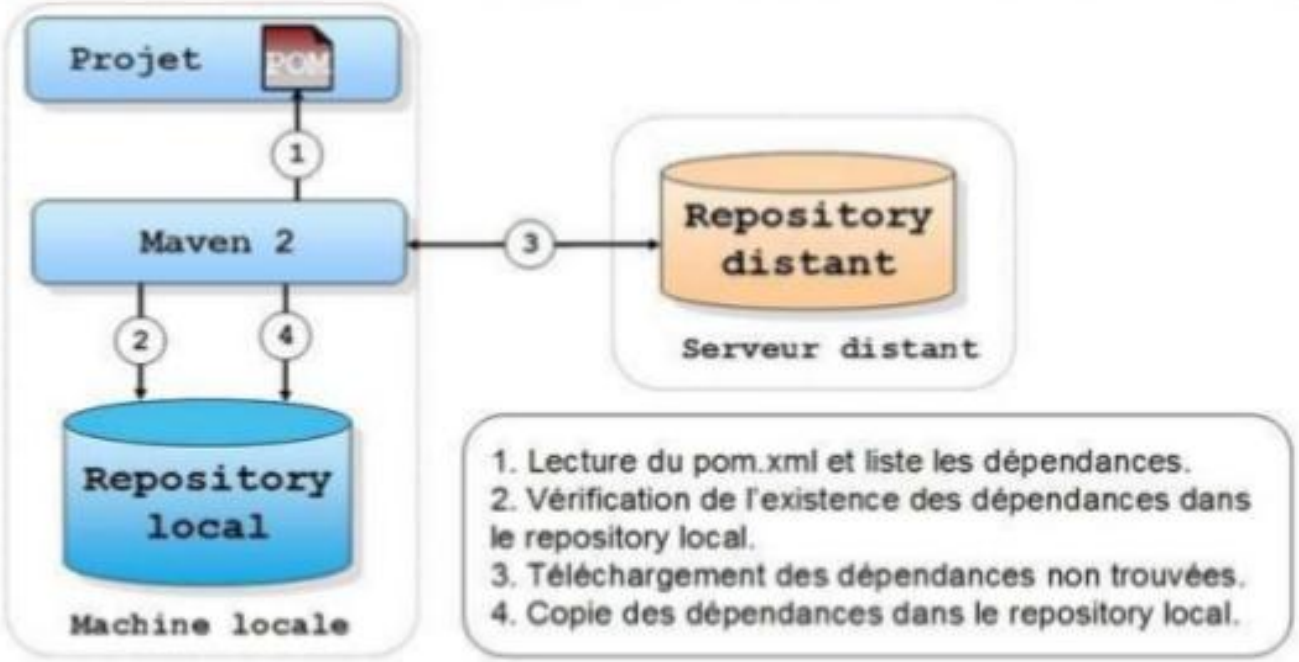
Ils jouent le rôle de **chef d'orchestre** pour piloter et automatiser le processus de développement logiciel :

- Gérer les dépendances
- Lancer la compilation des sources
- Lancer les Test Unitaires
- Générer les packages (jar, war, ear)
- Installer les packages dans le repository
- Déployer l'application dans le serveur
- Générer la documentation du projet
- ….

VIII.2. Gestion des dépendances

- Avec Maven toutes les dépendances d'un projet sont déclarées dans le fichier pom.xml
- Le plugin Maven de gestion de dépendances se charge de télécharger sur les repositories distants les fichiers jar indiqués comme dépendances, s'ils ne se trouvent pas dans le repository local.

Gestion des dépendances par Maven 2



VIII.3. Descripteur de projet : pom.xml

Project Object Model : POM

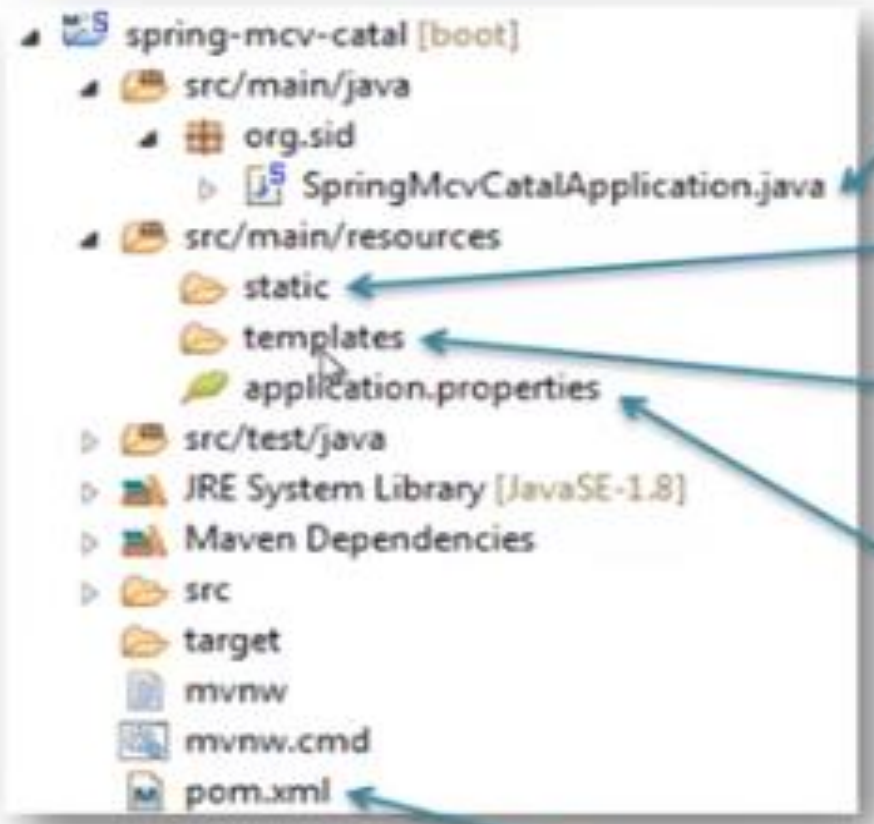
- Base de travail de Maven :
  - Un projet Maven est un module d'une application
  - Equivalent à un projet Eclipse
- Fichier XML (pom.xml) décrivant le projet Maven
  - Versions du projet
  - Description du projet
  - Liste des développeurs
  - Les dépendances
  - …
- Ce fichier est utilisé par maven pour construire l'application:
  - Dépendances de l'application (Laibrairies .jar)
  - Tâches (Gols) à exécuter
- Fournie des valeurs par défaut (Bonne pratique):
  - Exemple : Répertoire source (src/main/java)
- Un seul POM est nécessaire pour un projet
  - Le commandes de maven sont à exécuter à la racine du projet : l'emplacement du fichier pom.xml

VIII.4. POM minimal

- La racine du projet : **<project>**
- La version du modèle de pom (**<modelVersion>**) : 4.0.0 pour Maven 2.x
- L'identifiant du groupe auquel appartient le projet : **<groupId>**
  - Généralement commun à tous les modules d'un projet
- L'identifiant de l'artifact à construire: **<artefactId>**
  - Généralement le nom du module du projet sans espace en miniscules.
- La version de l'artefact à construire **<version> :** Souvent SNAPSHOT sauf lors de la release
- Le type d'artefact à construire: <packaging> : pom, jar, war, ear

```
<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/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.bp</groupId>
  <artifactId>reclamations</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>
</project>
```

IX. Structure d'un projet : Spring Starter Project



Application Spring Boot

Ressources statiques de l'application web : HTML, CSS, Java Script, images, etc…

Les vues de l'applications qui seront interprétées coté serveur par le moteur de vue Tymeleaf

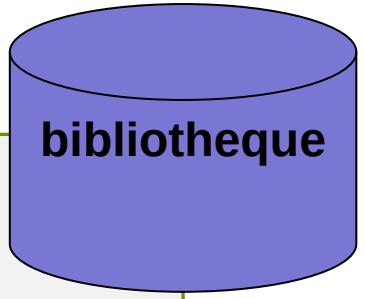
Fichier de configuration de l'application Spring Boot. Fichier lu au démarrage de Spring

Fichier de dépendances Maven

IX.1. Organisation des répertoires

- Maven propose une structure de fichier complète. Il s'agit de la configuration par défaut mais elle est surchargeable.
- Le principe général est de limiter le répertoire racine du projet à trois éléments:
  - Le fichier de description du projet **pom.xml** ,
  - Le répertoire **src** qui contient uniquement les sources du projet
  - et le répertoire **target** qui contient tous les éléments créé par Maven.

IX.2. Fichier de configuration : application.properties




```
# DataSource settings :
spring.datasource.url = jdbc:mysql://localhost:3306/bibliotheque
spring.datasource.username = root
spring.datasource.password =
spring.datasource.driver-class-name = com.mysql.jdbc.Driver

# Secify the DBMS
spring.jpa.database = MYSQL
#Show or not log for each sql query
spring.jpa.show-sql= true
# Hibernate ddl auto (create, create-drop,update)
spring.jpa.hibernate.ddl-auto = update

# Naming strategy
spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.ImprovedNamingStrategy

# Allows Hibernate to generate SQL optimized for a particular DBMS
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
```

Abréviation

- POO : Programmation Orientée Objet
- JDBC : Java Data Base Connectivity
- JEE / JavaEE : Java Entreprise Edition
- JPA : Java Persistance API
- JB : JavaBean
- EJB : Entreprise Java Beans
- JSP : Java Server Page
- STS : Spring Tools Suite