

Table des matières

MAVEN ET LE GENIE LOGICIEL	4
Introduction	4
Installation de Apache Maven	4
Préparation de l'environnement de développement	5
Présentation du contexte de travail.....	5
Mise en œuvre 1	6
Mise en œuvre 2	10

Liste des figures

Figure 1 : Configuration des variables d'environnement utilisateur	5
Figure 2 : Vérification installation Maven 3.6.3	5
Figure 3 : commande mvn archetype:generate.....	6
Figure 5 : projet Calcul généré.....	7
Figure 6 : répertoire src du projet Calcul.....	7
Figure 7 : IDE Eclipse Kepler	8
Figure 8 : Importation du projet dans Eclipse	8
Figure 9 : Echec d'Importation du projet dans Eclipse/Projet Introuvable par Eclipse	8
Figure 10 : Résultat commande mvn eclipse:eclipse.....	9
Figure 11 : Fichiers eclipse générés	9
Figure 12 : Importation du projet réussie dans Eclipse	9
Figure 13 : Dépôt Local.....	10
Mise en œuvre 2	10
Figure 14 : Classe CalculMetier+code	10
Figure 15 : Classe CalculMetierTest+code pour le test unitaire de CalculMetier.....	11
Figure 16 : Fichier pom.xml du projet Calcul.	11
Figure 17 : Résultat commande « mvn compile ».....	12
Figure 18 : Dossier /target et contenu générés par la commande « mvn compile »	12
Figure 19 : Résultat commande « mvn test »	13
Figure 20 : Résultat commande mvn package.....	13
Figure 21 : Fichier .jar de Calcul généré par la commande « mvn package ».....	14
Figure 22 : Installation/envoi du fichier .jar de Calcul généré dans/vers le repository : « mvn package »	14
Figure 23 : Structure de CalculWeb dans Eclipse	15
Figure 24 : Dépendances ajoutées à CalculWeb	18
Figure 25 : commande « mvn install » sur CalculWeb	20
Figure 26 : package .war de CalculWeb dans le repository	20
Figure 27 : Définition du packaging en .war dans pom.xml	20
Figure 28 : Présentation de apache-tomcat 8.....	21
Figure 29 : Configuration d'un utilisateur et rôles associés à celui-ci dans tomcat-users.xml.....	21
Figure 30 : Exécution du fichier startup.bat	21
Figure 31 : Démarrage de apache-tomcat.....	22
Figure 32 : Démarrage de apache-tomcat en localhost sur le port 8080 dans Google Chrome.....	22
Figure 33 : Page d'accueil de apache-tomcat après authentification.....	23
Figure 34 : Phase 1 de déploiement : espace de déploiement	23
Figure 35 : Phase 2 de déploiement : upload du .war du repository du PC vers tomcat	24

Figure 36 : Phase 3 de déploiement : CalculWeb.war déployé	24
Figure 37 : Phase de test : Opération de Somme (A+B)	24
Figure 38 : Phase de test : Opération du produit (A*B)	24
Figure 39 : Déploiement de CalculWeb avec Tomcat embarqué de Maven	26
Figure 40 : Génération des fichiers html du site du projet CalculWeb	27
Figure 41 : Site du projet CalculWeb	28
Figure 42 : Liste des dépendances du projet CalculWeb vue du site généré.....	28
Figure 43 : Documentation du projet CalculWeb vue du site généré.....	29
Conclusion.....	30
Figure 44 : Quelques Goal de Maven pour tomcat.....	30

MAVEN ET LE GENIE LOGICIEL

Introduction

Maven est un outil de construction de projets (build) open source développé par la fondation Apache, initialement pour les besoins du projet Jakarta Turbine. Il permet de faciliter et d'automatiser certaines tâches de la gestion d'un projet Java. Le site officiel de Maven : <http://maven.apache.org>.

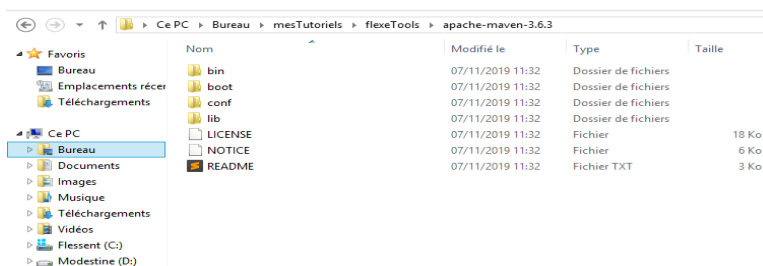
Maven permet non seulement d'automatiser les tâches de build mais aussi :

- Automatiser les tâches de: compilation, tests unitaires, gestion de dépendances (bibliothèques) et déploiement du projet (inclus les projets qui le composent) ;
- Générer la documentation technique du projet.

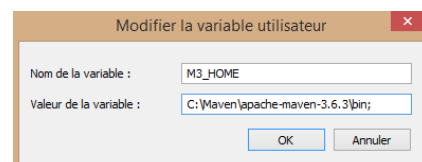
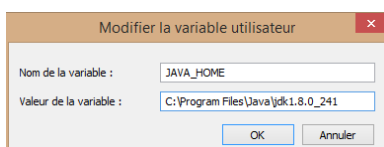
Il existe plusieurs versions de Maven à ce jour mais dans le cadre de ce cours, nous utiliserons la version 3.6.3.

Installation de Apache Maven

L'installation de Maven est des plus simples. Téléchargeons l'archive au format zip : apache-maven-3.6.3. Une fois dézipée nous avons :



Nous avons trois variables d'environnement utilisateur à créer et à remplir : JAVA_HOME, M3_HOME(car nous utilisons la version 3. Donc M2_HOME pour la version 2) et PATH.



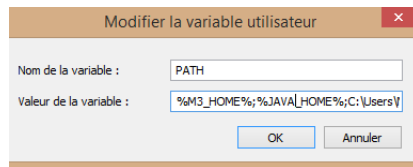


Figure 1 : Configuration des variables d'environnement utilisateur

Les figures sont assez claires et lisibles mais nous allons brièvement les commenter. La variable utilisateur JAVA_HOME sa valeur est le chemin absolu du repertoire de votre jdk(ici , nous utilisons la version 1.8) ; la variable M3_HOME spécifie le repertoire /bin du dossier maven obtenu précédemment après extraction de l'archive et enfin la variable PATH contient les deux autres variables(JAVA_HOME et M3_HOME), sa valeur vaut :
%M3_HOME%;%JAVA_HOME%;

C'est tout ! Maven est désormais installé et nous pouvons le vérifier(inclus parfois le redemarrage du système d'exploitation) en tapant notre première magique commande maven :
`mvn --version`

```
C:\Users\Modestine>mvn --version
Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)
Maven home: C:\Maven\apache-maven-3.6.3\bin\..
Java version: 1.8.0_241, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0_241\jre
Default locale: fr_FR, platform encoding: Cp1252
OS name: "windows 8.1", version: "6.3", arch: "amd64", family: "windows"
```

Figure 2 : Vérification installation Maven 3.6.3

Bravo ! Vous venez d'installer Maven. Nous pouvons constater sur cette figure que cette commande ne donne pas juste la version de Maven mais aussi la version du jdk , les informations sur le système d'exploitation et autres.

Préparation de l'environnement de développement

Maintenant que nous avons installé Maven, il ne nous reste plus qu'à passer à l'action. Mais ne soyons pas si pressés car il nous faut un IDE (Environnement de Développement Intégré) qui pourtant n'est pas nécessaire (pour certaines actions) pour moi car on peut tout gérer en ligne de commande avec Maven. L'IDE va nous aider juste comme un simple éditeur de texte à écrire le code. Mais pour ceux qui veulent, ils peuvent ignorer l'IDE et foncer sur un simple éditeur de text tels que Sublime Text ou NotePad++, et enregistrer ses fichiers avec l'extension .java

Dans notre cas, nous utiliserons la ligne de commande pour les taches d'automatisation et Eclipse Kepler comme IDE pour la partie code. Mon conseil ? Moi j'aime la méthode par ligne de commande car cela permet de connaitre les enjeux des actions qu'on entreprend et c'est indépendant de la plate-forme(IDE). Puisque cela n'implique pas apprendre un nouvel IDE à chaque fois.

Présentation du contexte de travail

Dans ce cours ou tutoriel si vous voulez, nous allons créer deux projets mais pas de suite. Un projet (Calcul) sera utilisé dans un autre projet(CalculWeb) (complètement différent) comme dépendance de celui-ci. Donc le projet Calcul sera une dépendance du projet CalculWeb. En

quelque sorte, Calcul va offrir ses services à CalculWeb entend que librairie .jar qui contient des méthodes à appeler.

Mise en œuvre 1

Faut que nous soyons au même niveau. Je veux dire : PRESENCE D'UNE CONNEXION INTERNET.

Pourquoi ? Tout simplement parce que Maven ira chercher les dépendances de notre projet sur le serveur central. Mais... Nous avons un répertoire /lib dans le dossier Maven pourquoi ne pas l'utiliser directement ? Oui Nous avons ici une réponse partielle à votre question sur la nécessité d'un accès Internet : Maven va télécharger les bibliothèques indiquées, à partir d'une source fiable, plutôt que de se contenter des fichiers JAR présents dans le répertoire /lib et dont la **version** et **l'origine** sont incertaines.

Nous allons commencer trêve de bavardage☺.

Sous CMD(windows) ou sous un terminal (en root sous les distributions Linux) lancez la commande :

-mvn archetype:generate

Que fais cette commande ? Excellente question ! Cette commande va aller sur le serveur central(ou repository central) télécharger et générer le catalogue ou le model des projets (Par exemple 23 : projet maven web J2EE Simple, 68 : projet maven+Spring+Hibernate,etc...) et vous n'aurez plus qu'à choisir un numéro correspondant au type de projet que vous souhaitez. Cette commande fonctionne en mode interactif et donc vous aurez des décisions ou des choix à prendre ou à faire.

Voici un extrait du résultat :

```
1572: remote -> org.apache.karaf.archetypes:karaf-command-archetype (A Karaf command archetype.)
1573: remote -> org.apache.karaf.archetypes:karaf-feature-archetype (This archetype sets up an empty karaf features project.)
1574: remote -> org.apache.karaf.archetypes:karaf-kar-archetype (This archetype sets up an empty karaf kar project.)
1575: remote -> org.apache.karaf.eik.archetypes:eik-camel-archetype (Camel PDE plugin to be used in EIK)
1576: remote -> org.apache.marmotta:marmotta-archetype-module (This Maven Archetype allows creating the basic structure for an Marmotta Module)
1577: remote -> org.apache.marmotta:marmotta-archetype-webapp (Web Application bundle (WAR file) containing Apache Marmotta)
1578: remote -> org.apache.maven.archetypes:maven-archetype-archetype (An archetype which contains a sample archetype.)
1579: remote -> org.apache.maven.archetypes:maven-archetype-j2ee-simple (An archetype which contains a simplified sample J2EE application.)
1580: remote -> org.apache.maven.archetypes:maven-archetype-marmalade-mojo (-)
1581: remote -> org.apache.maven.archetypes:maven-archetype-mojo (An archetype which contains a sample a sample Maven plugin.)
1582: remote -> org.apache.maven.archetypes:maven-archetype-plugin (An archetype which contains a sample Maven plugin.)
1583: remote -> org.apache.maven.archetypes:maven-archetype-plugin-site (An archetype which contains a sample Maven plugin site. This archetype can be layered upon an existing Maven plugin project.) ##### 1591
1584: remote -> org.apache.maven.archetypes:maven-archetype-portlet (An archetype which contains a sample JSR-268 Portlet.)
1585: remote -> org.apache.maven.archetypes:maven-archetype-profiles (-)
1586: remote -> org.apache.maven.archetypes:maven-archetype-quickstart (An archetype which contains a sample Maven project.)
1587: remote -> org.apache.maven.archetypes:maven-archetype-simple (An archetype which contains a sample Maven project.)
1588: remote -> org.apache.maven.archetypes:maven-archetype-site (An archetype which contains a sample Maven site which demonstrates some of the supported document types like API, XDoc, and FHL and demonstrates how to link your site. This archetype can be layered upon an existing Maven project.)
1589: remote -> org.apache.maven.archetypes:maven-archetype-site-simple (An archetype which contains a sample Maven site.)
1590: remote -> org.apache.maven.archetypes:maven-archetype-site-skin (An archetype which contains a sample Maven Site Skin.)
1591: remote -> org.apache.maven.archetypes:maven-archetype-webapp (An archetype which contains a sample Maven Webapp project.)
1592: remote -> org.apache.myfaces.buildtools:myfaces-archetype-codi-jsf12 (Archetype to create a new JSF 1.2 webapp based on MyFaces CODI)
1593: remote -> org.apache.myfaces.buildtools:myfaces-archetype-codi-jsf20 (Archetype to create a new JSF 2.0 webapp based on MyFaces CODI)
1594: remote -> org.apache.myfaces.buildtools:myfaces-archetype-core-integration-test (Archetype to create a new MyFaces core project used for integration tests via cargo and HttUnit).
```

Figure 3 : commande mvn archetype:generate

Prenons le numéro (entrons le numéro correspondant à **remote ->**

org.apache.maven.archetypes:maven-archetype-quickstart (An archetype which contains a sample Maven project.)) Maven propose généralement ce model par défaut et il faudrait juste taper sur la touche Enter. Chez moi, ce model correspond au numéro 1586.

Une fois le numéro entré ou la touche Enter pressée, nous avons à répondre à quatre questions :

- version ?
Il s'agit de la version de notre projet. Par défaut Maven propose le dernier numéro généralement la version 1.1. Tapons alors juste sur la touche Enter.
- groupId ?

Il s'agit de l'organisation ou de l'équipe ou encore du propriétaire du projet. Mettons org.flexe ;

- artifactId ?
Il s'agit du nom proprement dit du projet. Mettons Calcul (comme nous l'avons précédemment dit).
- package ?
Il s'agit du nom du package où seront les fichiers source. Par défaut Maven propose le même nom que le groupId. Entrons org.flexe.calcul ;

Les quatre questions ont été répondues et maintenant laissons Maven générer et configurer notre projet😊. Si nous jetons un coup d'œil au projet :

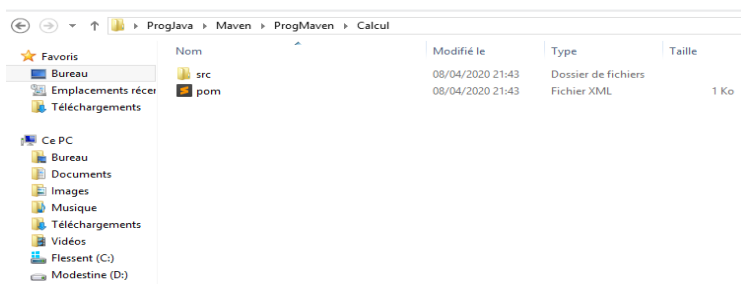


Figure 5 : projet Calcul généré

Maven a généré notre projet Calcul qui contient un répertoire src et un fichier xml :POM(Project Object Model) ;

Le dossier src lui contient deux sous-dossiers **main** (pour les fichiers source : logique métier) et **test** (pour les fichiers source des tests unitaires) :

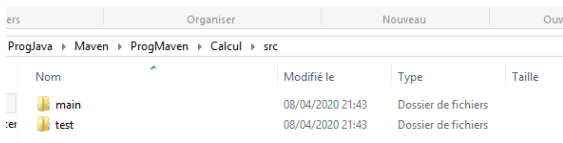


Figure 6 : répertoire src du projet Calcul

Si vous parcourez src/main/java/org/flexe/calcul vous verrez un fichier .java contenant une simple classe App.java et de même dans src/test/java/org/flexe/calcul vous verrez AppTest.java représentant le test unitaire de la classe App.java ;

Maintenant notre projet a été généré et nous l'avons parcouru. Maintenant importons-le dans Eclipse Kepler.

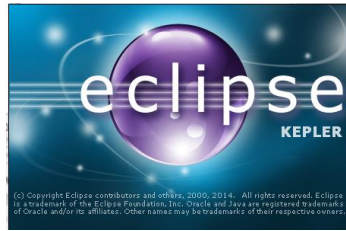


Figure 7 : IDE Eclipse Kepler

Allons à l'onglet File ensuite import après General et enfin Existing Projects into Workspace :

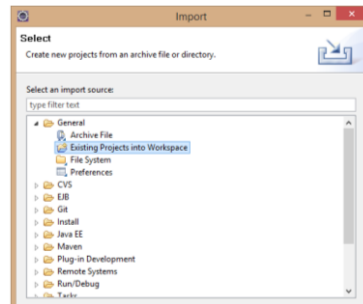


Figure 8 : Importation du projet dans Eclipse

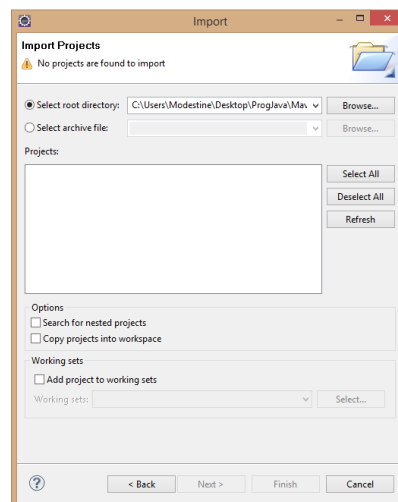


Figure 9 : Echec d'Importation du projet dans Eclipse/Projet Introuvable par Eclipse

Mais Quoi ??? Pourquoi Eclipse ne trouve pas notre projet ? Une erreur c'est sûr. Mais NON ! NON ! et NON ! Y'a pas d'erreurs. N'oubliez pas que nous travaillons avec un IDE et chaque IDE a ses propres fichiers de reconnaissance de projet. Pour Eclipse, il faut deux fichiers fondamentaux : fichier .classpath et le fichier .project ;

Mais où trouvez ses fichiers alors ? Ne vous inquiétez pas ! Maven le fera pour vous. Génial pas vrai ? ☺. Alors prouvons-le !

-mvn eclipse:eclipse

Cette commande permet de générer la structure d'un projet eclipse dans un projet Maven. Positionnons-nous dans notre projet Calcul via la commande **cd**

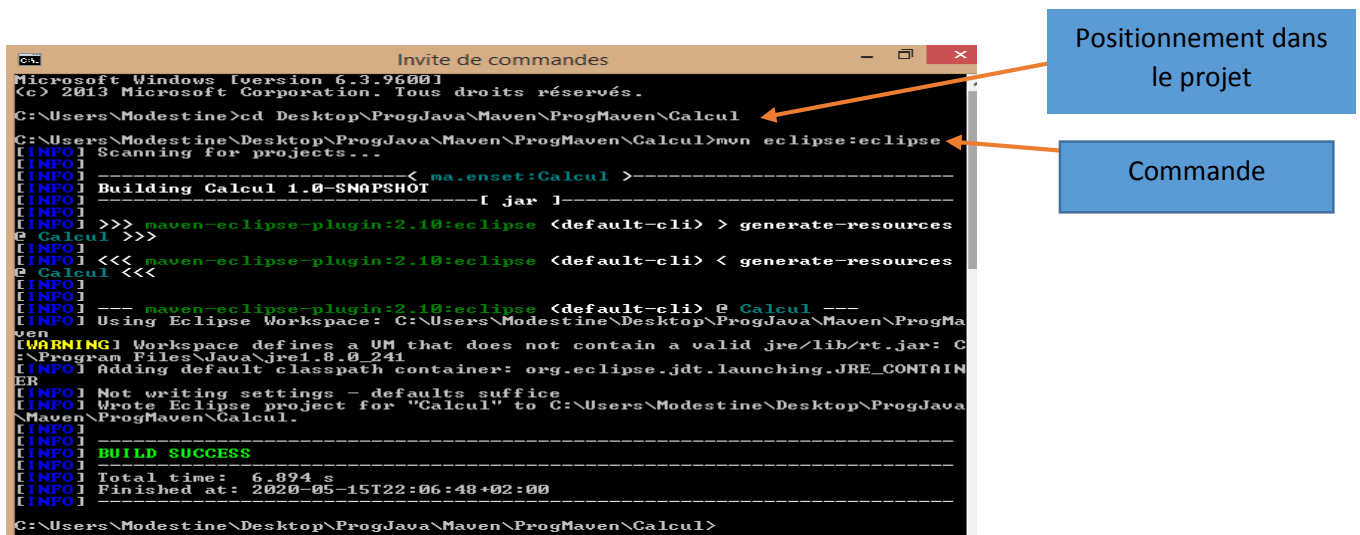


Figure 10 : Résultat commande mvn eclipse:eclipse

La commande va aller télécharger le plugin eclipse. Une fois quelques téléchargements effectués, ouvrons à nouveau notre projet Calcul. Oups !!! Nos deux fichiers ont été générés :

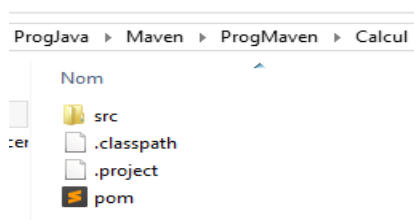


Figure 11 : Fichiers eclipse générés

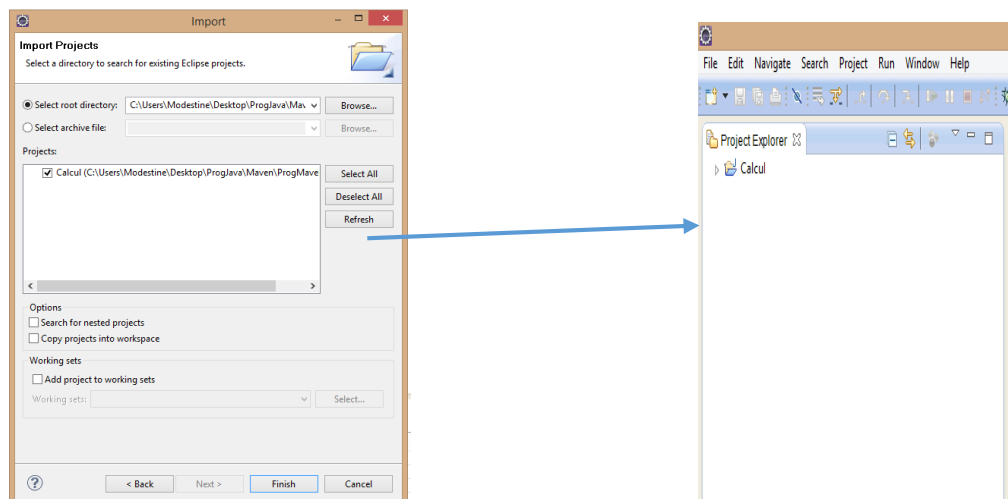


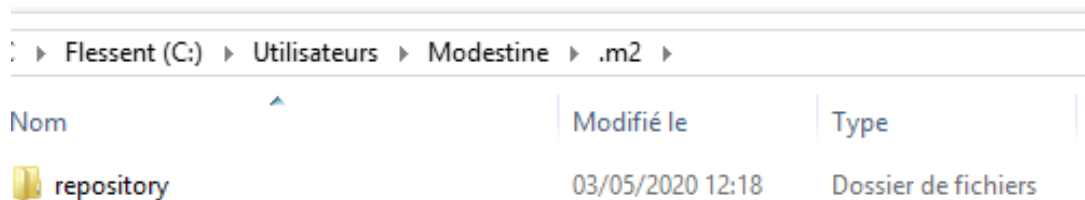
Figure 12 : Importation de nouveau du projet réussie dans Eclipse

Alors ! Jusqu'ici tout va bien ! Mais devons-nous tout le temps être dépendant d'internet ?

OUI et NON. OUI parce qu'il existera toujours de nouvelles dépendances, bibliothèques ou fichiers à ajouter au projet au fur et à mesure que le projet grandit et il faudra les télécharger

pour des raisons de sécurité invoquées plus haut. NON parce que, une fois un plugin téléchargé, vous n'avez plus besoin de le retélécharger quel que soit le nouveau projet. Mais pourquoi ?

Tout simplement parce que lors des premiers téléchargements, Maven crée un dossier nommé repository où il stocke les plugins déjà téléchargés. C'est une sorte de dépôt local. Vous pouvez le trouver dans :

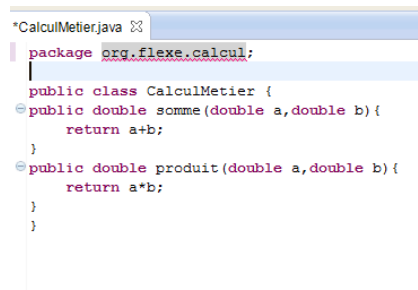


Flessent (C:) > Utilisateurs > Modestine > .m2 >			
Nom	Modifié le	Type	
repository	03/05/2020 12:18	Dossier de fichiers	

Figure 13 : Dépôt Local

Mise en œuvre 2

Maintenant que nous sommes dans Eclipse, créons une classe simple nommée CalculMetier et mettons ce code :

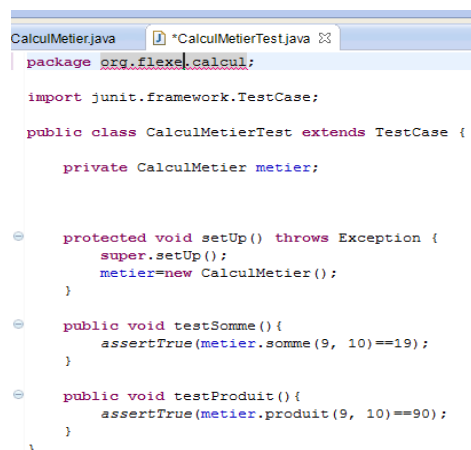


```
*CalculMetier.java
package org.flexe.calcul;

public class CalculMetier {
    public double somme(double a, double b) {
        return a+b;
    }
    public double produit(double a, double b) {
        return a*b;
    }
}
```

Figure 14 : Classe CalculMetier+code

Créons de meme une autre classe CalculMetierTest et mettons ce code :



```
CalculMetier.java  CalculMetierTest.java
package org.flexe.calcul;

import junit.framework.TestCase;

public class CalculMetierTest extends TestCase {

    private CalculMetier metier;

    protected void setUp() throws Exception {
        super.setUp();
        metier=new CalculMetier();
    }

    public void testSomme() {
        assertTrue(metier.somme(9, 10)==19);
    }

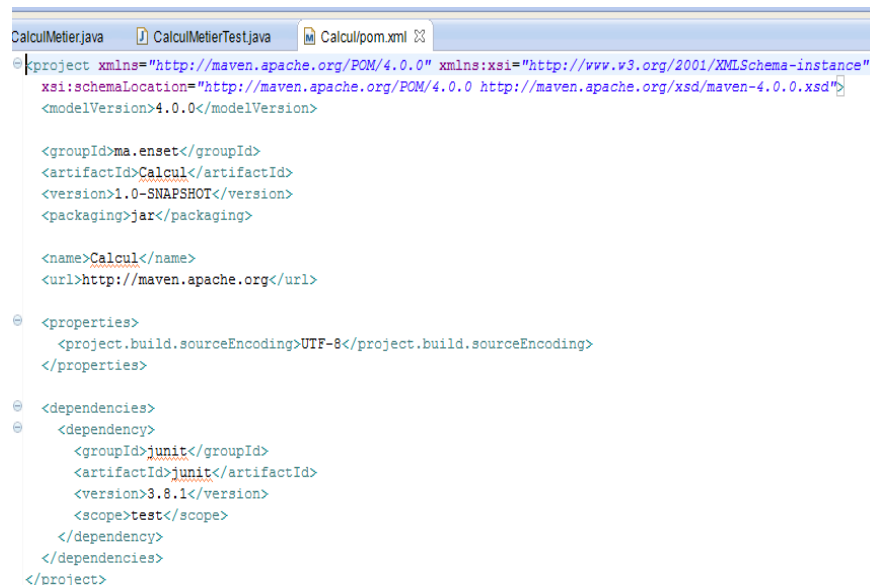
    public void testProduit() {
        assertTrue(metier.produit(9, 10)==90);
    }
}
```

Figure 15 : Classe CalculMetierTest+code pour le test unitaire de CalculMetier.

Vous l'aurez certainement peut-être déjà remarqué dans les importations que nous utilisons ici le framework JUnit pour les tests unitaires.

Avant de continuer, explorons ce curieux, mystérieux et fameux fichier pom.xml ;

Celui-ci contient la **DESCRIPTION DU PROJET**. Le point capital ici est que la description de la configuration et des commandes nécessaires à la construction correcte du livrable est entièrement documentée dans le POM lui-même. C'est ce fichier qui retrace tous les éléments (bibliothèques, projet, etc...) dont dépend le projet. Avec ce fichier, nous avons le squelette complet de l'application sans avoir besoin de fouiller dans les dossiers du projet.



```
<?xml version="1.0" encoding="UTF-8" ?>
<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>ma.enset</groupId>
  <artifactId>Calcul</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>Calcul</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Figure 16 : Fichier pom.xml du projet Calcul.

Imaginons un seul instant que vous travaillez sur un projet java sans utiliser Maven et vous l'abandonnez. Après 5 ans, un de vos amis vous demande de lui donner votre code source. Vous allez dans vos archives et trouvez le projet et vous le lui donnez. De son côté, il tente de continuer votre travail abandonné mais il se pose de nombreux problèmes :

- Problèmes de compatibilité : les chemins des bibliothèques/librairies de votre projet ne sont pas trouvables car votre ami travaille sur son PC à lui et non celui avec lequel vous avez créé le projet initialement ;
- Problèmes de bibliothèques/librairies : votre PC vous indique qu'il manque par exemple le fichier mail-1.2.jar. Votre ami court directement sur internet et télécharge le fichier mail-1.2.jar et l'ajoute au projet ;
- Problèmes de sécurité réseau: lors du téléchargement de mail-1.2.jar par votre ami ; Supposons que cette bibliothèque ait été corrompue lors de son téléchargement depuis le site de SUN qui la diffuse. Un transfert réseau n'est jamais 100 % garanti, et un seul bit modifié peut rendre la bibliothèque inutilisable, sans parler de ces charmants petits virus qui peuvent traîner un peu partout. L'identification du problème peut être extrêmement complexe, car la remise en cause de la bibliothèque sera probablement la toute dernière hypothèse que nous évoquerons pour justifier un dysfonctionnement ;

- Problème de sécurité ou erreurs dans la bibliothèque/librairie :
Après 5 ans, les choses évoluent. Surtout en informatique et voilà votre ami qui tente de continuer un projet datant de 5 ans. Après quelques recherches sur internet, il apprend que la version 1.2 de javamail avait un bug de sécurité ;
- Problèmes de compatibilité entre ancien code et bibliothèque mail.jar à jour téléchargée : Les correctifs de sécurité ont été appliqués et le code datant de 5 ans devient inutilisable car certaines méthodes utilisées ne sont plus acceptées ou bien sont devenues deprecated (obsolètes);
- Etc....
Vous remarquez aisément les difficultés d'un projet qui n'utilise pas un outil (par exemple Maven) de gestion de dépendance qui ira chaque fois sur internet (de manière sécurisée) chercher les éléments manquants et les mettre à jour instantanément et le projet sera facilement maintenable même après 50 ans et ce peu importe le nombre de développeurs impliqué ☺.

Revenons à notre projet Calcul.

Maintenant que nous avons fini nos méthodes, il faut **compiler**, **tester**, **packager** et **installer**.

-mvn compile

Cette commande va générer un dossier "target" dont l'arborescence (C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\Calcul\target\classes\org\flexe\calcul) contient les fichiers .class (dans un dossier) des classes métier et d'autres fichiers .class toujours dans un autre dossier pour les tests unitaire. Cette commande va faire le build, c'est-à-dire la construction de notre projet et cela inclus la vérifier si toutes les dépendances sont belles et bien présentes et faites.

```

C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\Calcul>mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] --->ma.enset:Calcul<---
[INFO] Building Calcul 1.0-SNAPSHOT
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ Calcul ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\Calcul\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ Calcul ---
[INFO] Nothing to compile - all classes are up to date
[INFO] BUILD SUCCESS
[INFO] Total time: 3.411 s
[INFO] Finished at: 2020-05-15T23:58:29+02:00
[INFO]
C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\Calcul>

```

Commande

Figure 17 : Résultat commande « mvn compile »

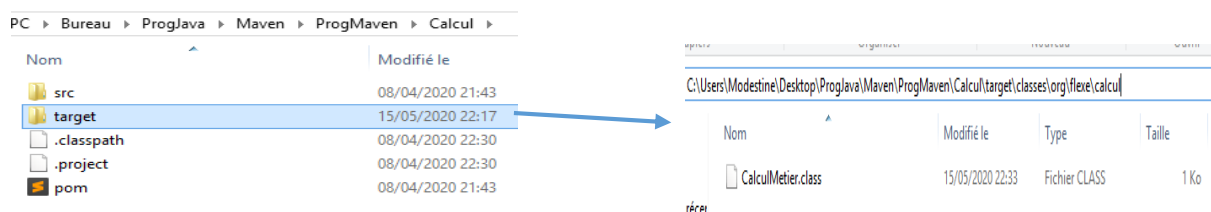


Figure 18 : Dossier /target et contenu générés par la commande « mvn compile »

NB : Tout est plugin sous Maven. Donc « compile », « eclipse », etc. Sont des plugins.

-mvn test

Cette commande concerne les tests unitaires. Ici « test » est un nouveau plugin. Maven doit donc le télécharger et exécuter les tests.

```

C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\Calcul>mvn test
[INFO] Scanning for projects...
[INFO] Building Calcul 1.0-SNAPSHOT [ jar ]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ Calcul ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\Calcul\src\main\resources
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ Calcul ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ Calcul ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\Calcul\src\test\resources
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ Calcul ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ Calcul ---
[INFO] Surefire report directory: C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\Calcul\target\surefire-reports

T E S T S
Running ma.enset.calcul.CalculMetierTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.035 sec
Results :
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO] BUILD SUCCESS
[INFO] Total time: 5.034 s
[INFO] Finished at: 2020-05-16T00:23:42+02:00
[INFO]
C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\Calcul>
  
```

Commande

Début des tests et statistiques en dessous

Figure 19 : Résultat commande « mvn test »

-mvn package

Cette commande permet de packager soit en jar ou en war ou encore soit en ear(selon votre configuration dans le pom.xml); Cette commande va donc générer un fichier .jar(par défaut voir balise <packaging>jar</packaging> dans le pom.xml) dans le dossier "target".

```

C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\Calcul>mvn package
[INFO] Scanning for projects...
[INFO] Building Calcul 1.0-SNAPSHOT [ jar ]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ Calcul ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\Calcul\src\main\resources
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ Calcul ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ Calcul ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\Calcul\src\test\resources
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ Calcul ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ Calcul ---
[INFO] Surefire report directory: C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\Calcul\target\surefire-reports

T E S T S
Running ma.enset.calcul.CalculMetierTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.045 sec
Results :
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ Calcul ---
[INFO] Building jar: C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\Calcul\target\Calcul-1.0-SNAPSHOT.jar
[INFO] BUILD SUCCESS
[INFO] Total time: 4.934 s
[INFO] Finished at: 2020-05-16T00:28:26+02:00
[INFO]
C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\Calcul>
  
```

Commande

Reprise de la phase de tests et statistiques en dessous

Figure 20 : Résultat commande mvn package

Nous constatons que cette commande refais encore les tests unitaires avant de packager ; Cette commande va donc générer un .jar (comme spécifier dans pom.xml).

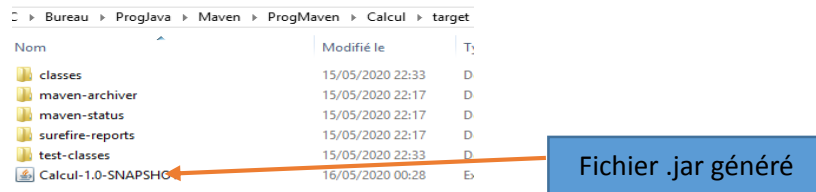


Figure 21 : Fichier .jar de Calcul généré par la commande « mvn package »

-mvn install

Cette commande ou ce plugin permet d'installer le fichier .jar généré précédemment avec la commande « mvn package ». Mais installer où ?

Excellente question ! Je savais que vous vous demanderez cela 😊. Euh... Euh... souvenez-vous du dépôt local dont je vous ai parlé plus haut ? Bien sûr que oui le repository.

Alors cette commande va prendre le fichier .jar généré et l'envoyer vers le repository (local/distant selon votre spécification dans setting.xml de Maven).

Nous avons maintenant notre projet envoyé dans le repository Calcul.jar (qui possède une classe et qui fait des calculs: somme, produit, etc.).

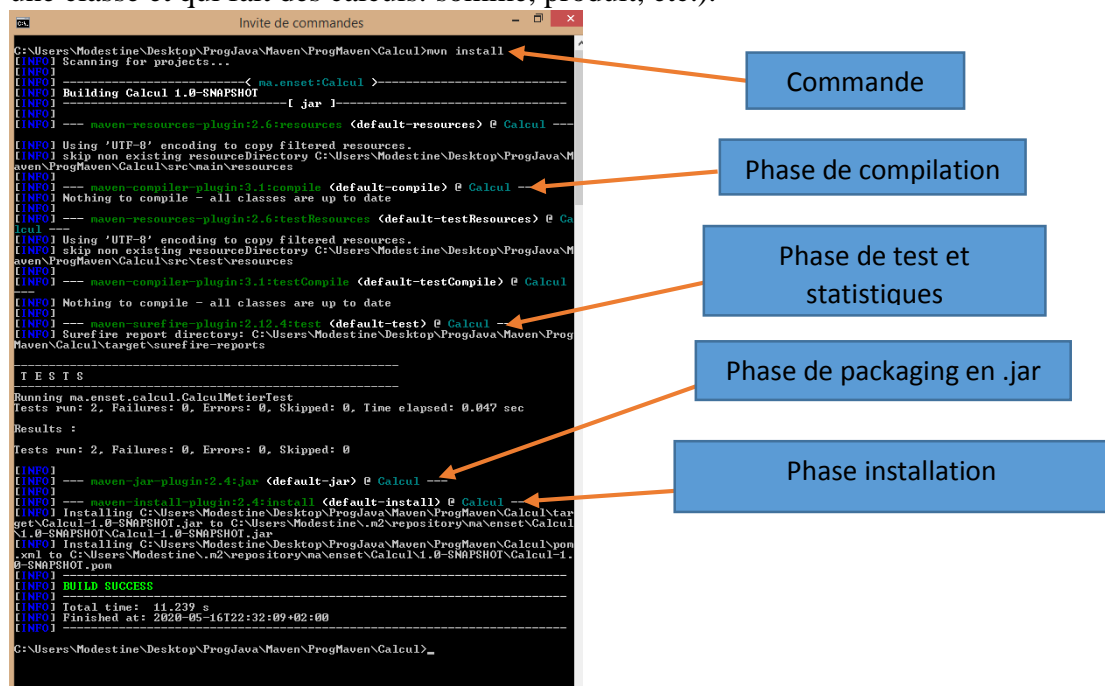


Figure 22 : Installation/envoi du fichier .jar de Calcul généré dans/vers le repository : « mvn package »

La figure ci-dessus (Figure 22) est très intéressante et bourrée d'informations utiles. Nous constatons que la commande « mvn install » reprend toutes les trois autres phases : la compilation, les tests unitaires, et le packaging. Si vous faites ceci dans un IDE, vous ne verrez pas ces étapes cruciales. Pour moi, c'est ça l'essentiel : voir comment ça marche en profondeur.

Voilà pourquoi j'ai voulu qu'on travaille en ligne de commande 😊 !

Bien ! Félicitations ! Vous venez de réaliser plus ou moins le cycle de gestion d'un projet avec Maven.

Maintenant nous allons créer un autre projet. Cette fois ci un projet web dans lequel on va utiliser le Calcul.jar situé dans le repository comme dépendance et dans notre application web, on va pouvoir exécuter les opérations de calcul comme si elles étaient implémentées dans notre application Web. C'est parti ! Nous créons un projet suivant le modèle :

remote -> org.apache.maven.archetypes:maven-archetype-webapp (An archetype which contains a sample Maven Webapp project.)

Comme précédemment, vous répondez aux quatre questions ainsi :

- Version : valeur par défaut (touche Enter) ;
- groupId : org.flexe ;
- artifactId : CalculWeb ;
- package : org.flexe.

Vous reprenez scrupuleusement le même parcours que celui avec le projet Calcul jusqu'à importation dans l'IDE Eclipse.

Ici, aucun code java ne sera écrit. Donc CalculWeb est ici comme un client de Calcul.

Tout d'abord, voici la structure de CalculWeb après import dans Eclipse :

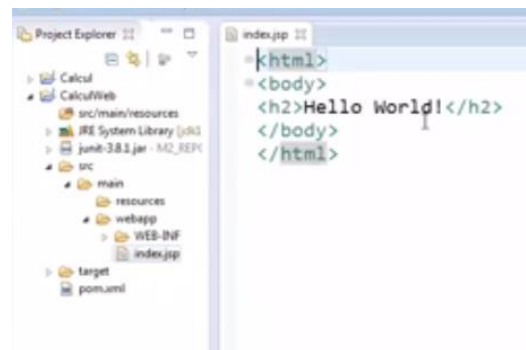


Figure 23 : Structure de CalculWeb dans Eclipse

Voilà ! le projet est importé et contient dans /webapp un fichier index.jsp.

Mettons ce code dans index.jsp :

```
<% @page import="org.flexe.calcul.CalculMetier"%>
```

```
<%
```

```
double a=0; double b=0; double res=0;
```

```
String action=request.getParameter("action");
```

```
if(action!=null){
```

```
    a=Double.parseDouble(request.getParameter("a"));
```

```
    b=Double.parseDouble(request.getParameter("b"));
```

```
    CalculMetier metier=new CalculMetier();
```

```
    if(action.equals("Somme")){
```

```
        res=metier.somme(a,b);
```

```
    }else{res=metier.produit(a,b);} }
```

```

}
%>
<html>
<body>
<title>Bienvenue sur les calculs avec Maven</title>
<form action="index.jsp" method="post">
<table>
<tr>
<td>
A:
</td>
<td>
<input type="text" name="a" value="<%=a%>"/>
</td>
<td>
B:
</td>
<td>
<input type="text" name="b" value="<%=b%>"/>
</td>
</tr>
<tr>
<td>
<input type="submit" name="action" value="Somme"/>
</td>
<td>
<input type="submit" name="action" value="Produit"/>
</td>
</tr>
<tr>
<td>
<td>Résultat : </td> <td><%=res%></td>
</tr>
</table>
</form>
</body>
</html>

```

Si nous lançons par erreur « mvn compile » ou par erreur grave « mvn test » ou par erreur intolérable « mvn package » ou encore par erreur pire, fatale et impardonnable « mvn install », nous obtenons de belles injures (erreurs 😊) ! Mais pourquoi encore ? Pourtant index.jsp ne mentionne pas d'erreurs. Eh oui ! Vous avez raison mais dommage pour vous que le problème ne peut être résolu au niveau de index.jsp. Premièrement Observez bien votre page index, elle a l'extension .jsp pourtant Maven ne connaît pas un tel format, de deux, vous avez créé un projet Web Maven et non un projet Web dynamique donc vous avez besoin de servlet car ici c'est une page jsp et pourtant Maven ne connaît ce qu'est une servlet, de trois observez la première ligne de votre index.jsp :

```
<% @page import="org.flexe.calcul.CalculMetier"%>
```

Cette ligne importe le fichier CalculMetier du package « org.flexe.calcul » pourtant Maven ne connaît pas qui est-ce en réalité ,de quatre si vous utilisez jsp , vous allez utiliser également la

librairie JSTL(Java server page Standard Tag Library :C'est un ensemble de tags personnalisés développé sous la JSR 052 qui propose des fonctionnalités(Tag de structure (itération, conditionnement ...)utilisation de documents XML, exécution des requêtes SQL,etc.) souvent rencontrées dans les JSP) Mais Maven n'est pas un magicien pour deviner ce que vous voulez dire, faire et allez le chercher pour vous voyons ! NON ! Nous allons un peu aider Maven en le lui expliquant dans pom.xml et lui se chargera d'aller chercher ces éléments pour nous : ce sont les DEPENDANCES de CalculWeb qu'il faut ajouter ;

Ajoutons ceci dans le tag ou dans les balises `<dependencies>` **Mettez ce qui suit ici**`<dependencies>`

- Dépendance servlet-api :

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>servlet-api</artifactId>
  <version>2.5</version>
  <scope>provided</scope>
</dependency>
```

- Dépendance jsp-api :

```
<dependency>
  <groupId>javax.servlet.jsp</groupId>
  <artifactId>jsp-api</artifactId>
  <version>2.1</version>
  <scope>provided</scope>
</dependency>
```

- Dépendance JSTL :

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
  <scope>compile</scope>
</dependency>
```

Cette dépendance a tout son sens. Observez index.jsp et vous constatez la présence des conditions (if (...) {...}). Le support permettant ainsi d'écrire ces conditions dans JSP est la JSTL.

- Dépendance du projet Calcul crée au début du cours :

```
<dependency>
  <groupId>org.flexe</groupId>
  <artifactId>Calcul</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>
```

- Dépendance pour les tests unitaires :

```
<dependency>
```

```
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.11</version>
<scope>test</scope>
</dependency>
```

Concernant la dépendance des tests unitaires ci-dessus, vous constatez qu'elle était déjà présente lors de la création de Calcul dans son pom.xml. Pourtant ici, nous l'avons ajoutée manuellement. Sachez donc que généralement dans un projet web dynamique y'aura toujours ces dépendances là !

Voilà le strict minimum nécessaire et suffisant à inclure avant de lancer nos commandes Maven. Enregistrons notre pom.xml et relançons la commande « mvn eclipse:eclipse ». Il faut toujours exécuter cette commande à chaque fois que nous ajoutons une nouvelle dépendance à notre pom.xml. En effet la commande « mvn eclipse:eclipse » va lire tout d'abord le fichier pom.xml et ensuite construire les dépendances (ajouter ou mettre à jour les dépendances). Vous aurez donc ainsi compris pourquoi il faut toujours relancer cette commande à chaque nouvelle dépendance/modification qu'on ajoute/applique au pom.xml. Nous pouvons voir au niveau du projet ces dépendances après leurs téléchargements :

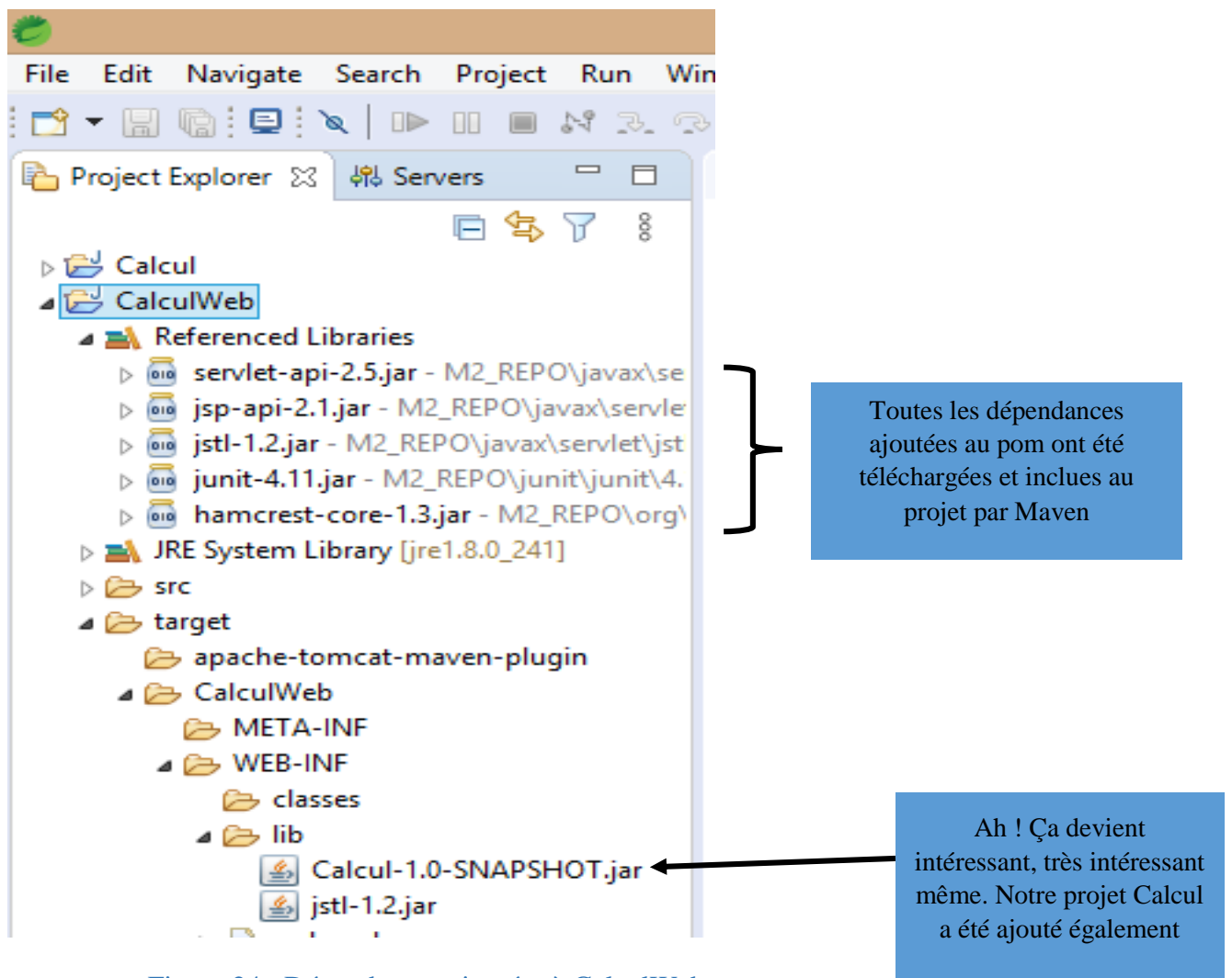


Figure 24 : Dépendances ajoutées à CalculWeb

Mais qu'est-ce qui s'est réellement passé ? Souvenez-vous avec la commande « mvn install » le package (en .jar) de notre projet Calcul a été envoyé dans le repository. Donc quand nous l'avons ajouté comme dépendance au pom.xml de notre CalculWeb, alors Maven, est juste allé chercher ce Calcul.jar envoyé lors de « mvn install » dans le repository et venir l'ajouter à CalculWeb. C'est tout !

Il est très important que vous sachiez que jusqu'ici je ne dépend d'aucun serveur. Puisque ce qui se passe généralement est que lors de la création d'un projet web, ce projet est déjà lié à Tomcat, donc le projet dépend déjà d'un ensemble de fichiers jar qui se trouvent déjà dans Tomcat. A la fin de mon projet, je pourrais maintenant déployer où je veux (soit dans Tomcat ou autres).

Maintenant tout est près ! Nous devons compiler, tester, packager et installer ou tout simplement installer (car cela inclut toutes les autres phases). Avec directement « mvn install », il se peut qu'il y'ai des erreurs dans la compilation, dans les tests, etc...Donc le plus important est de lire les messages générés sur le terminal.

```
C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\CalculWeb>mvn install
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for
ma.enset:CalculWeb:war:1.0-SNAPSHOT
[WARNING] Reporting configuration should be done in <reporting> section, not in
maven-site-plugin <configuration> as reportPlugins parameter. @ line 111, column
20
[WARNING]
[WARNING] It is highly recommended to fix these problems because they threaten t
he stability of your build.
[WARNING]
[WARNING] For this reason, future Maven versions might no longer support buildin
g such malformed projects.
[WARNING]
[INFO] -----< ma.enset:CalculWeb >-----
[INFO] Building CalculWeb Maven Webapp 1.0-SNAPSHOT
[INFO] [ war ]
[WARNING] The POM for javax.servlet.jsp:jsp-api:jar:2.1 is invalid, transitive d
ependencies <if any> will not be available, enable debug logging for more detail
s
[WARNING] The POM for javax.servlet:jstl:jar:1.2 is invalid, transitive dependen
cies <if any> will not be available, enable debug logging for more details
[INFO] --- maven-resources-plugin:3.0.2:resources <default-resources> @ CalculWe
b ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\Modestine\Desktop\ProgJava\M
aven\ProgMaven\CalculWeb\src\main\resources
[INFO] --- maven-compiler-plugin:3.8.0:compile <default-compile> @ CalculWeb ---
[INFO] No sources to compile
[INFO] --- maven-resources-plugin:3.0.2:testResources <default-testResources> @
CalculWeb ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\Modestine\Desktop\ProgJava\M
aven\ProgMaven\CalculWeb\src\test\resources
[INFO] --- maven-compiler-plugin:3.8.0:testCompile <default-testCompile> @ Calcul
Web ---
[INFO] No sources to compile
[INFO] --- maven-surefire-plugin:2.22.1:test <default-test> @ CalculWeb ---
[INFO] No tests to run.
[INFO] --- maven-war-plugin:3.2.2:war <default-war> @ CalculWeb ---
[INFO] Packaging webapp [CalculWeb] in [C:\Users\Modestine\Desktop\ProgJava\Mav
en\ProgMaven\CalculWeb\target\CalculWeb]
[INFO] Processing war project
[INFO] Copying webapp resources [C:\Users\Modestine\Desktop\ProgJava\Maven\ProgM
aven\CalculWeb\src\main\webapp]
[INFO] Webapp assembled in [119 msecs]
[INFO] Building war: C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\CalculW
eb\target\CalculWeb.war
[INFO] --- maven-install-plugin:2.5.2:install <default-install> @ CalculWeb ---
[INFO] Installing C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\CalculWeb\t
arget\CalculWeb.war to C:\Users\Modestine\.m2\repository\ma\enset\CalculWeb\1.0-S
NAPSHOT\CalculWeb-1.0-SNAPSHOT.war
[INFO] Installing C:\Users\Modestine\Desktop\ProgJava\Maven\ProgMaven\CalculWeb\p
om.xml to C:\Users\Modestine\.m2\repository\ma\enset\CalculWeb\1.0-SNAPSHOT\Cal
culWeb-1.0-SNAPSHOT.pom
[INFO] BUILD SUCCESS
```

Commande

Phase de compilation

Phase de test

Phase de packaging en war

Phase d'installation

Figure 25 : commande « mvn install » sur CalculWeb

Voilà ☺ !!! Tout s'est bien passé. Je vais plus revenir sur la lecture des différentes phases que la commande install de Maven exécute et traverse. Mais voyons, vous êtes quand-même mes amis. J'ai laissé quelques commentaires ou observations près de la figure pour vous. On peut vérifier dans le repository la présence du war de notre CalculWeb :

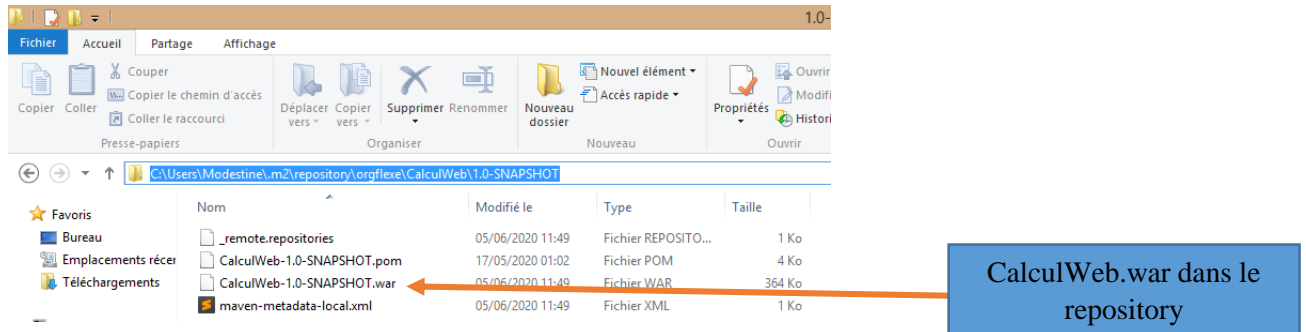


Figure 26 : package .war de CalculWeb dans le repository

Donc c'est n'est plus un .jar mais un .war car Maven sait que c'est un projet web. En effet vous pouvez le voir dans le pom.xml au niveau de la balise packaging :

```
<groupId>org.flexe</groupId>
<artifactId>CalculWeb</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>war</packaging>
```

Figure 27 : Définition du packaging en .war dans pom.xml

Nous pouvons le modifier et décider que cela soit packager sous un autre format autre qu'un war.

Maintenant il faut faire tourner l'application web et voir si nous effectuons bien nos opérations de calcul situées dans le projet Calcul. Ici, nous avons plusieurs possibilités : soit tester directement avec un serveur Tomcat externe que vous aurez téléchargé, soit utiliser toujours Maven. En effet Maven dispose d'un plugin Tomcat pouvant tourner en serveur embarqué.

- ✓ Utilisation d'un Tomcat externe
Bien ! Ici, nous allons utiliser tomcat version 8 :

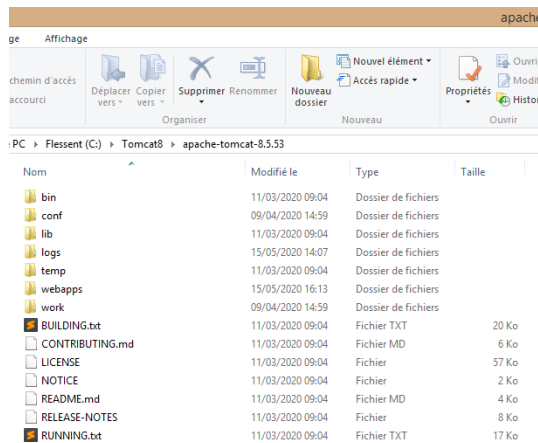


Figure 28 : Présentation de apache-tomcat 8

Nous avons quelques modifications à faire (inclus redémarrer le serveur):

Allons dans le dossier /conf observable sur la figure ci-dessus (figure 28) et ouvrons le fichier tomcat-users.xml. Nous allons configurer un utilisateur administrateur ici : Dans les balises tomcat-users mettons ceci:

```

17 -->
18 <tomcat-users xmlns="http://tomcat.apache.org/xml"
19             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
20             xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
21             version="1.0">
22
23 <role rolename="manager-gui"/>
24 <role rolename="admin-gui"/>
25 <role rolename="manager-script"/>
26 <user username="admin" password="admin" roles="manager-gui,admin-gui,manager-script"/>
27 </tomcat-users>

```

Configuration du role, du
username et du password

Figure 29 : Configuration d'un utilisateur et rôles associés à celui-ci dans tomcat-users.xml

Nous avons configuré trois rôles (manager-gui, admin-gui et manager-script), un seul utilisateur (username=admin et password=admin) et nous lui avons donné tous ces trois rôles (roles=manager-gui, admin-gui, manager-script).

Enregistrez ce fichier et positionnez-vous en ligne de commande dans le dossier /bin.
Exécutez le fichier startup.bat :

Figure 30 : Exécution du fichier startup.bat

L'exécution de ce fichier ouvrira un nouvelle fenêtre (fichier .jar) ou un nouveau terminal montrant ainsi les étapes de démarrage et le démarrage de tomcat :

```
Tomcat
mc8\apache-tomcat-8.5.53\webapps\CalculWeb-1.0-SNAPSHOT.war] s'est termin[0 en [12:41:43] ms
05-Jun-2020 12:44:13.620 INFOS [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory D:\ploiment du r\@pertoire d'application web [C:\Tomcat8\apache-tomcat-8.5.53\webapps\docs]
05-Jun-2020 12:44:13.752 INFOS [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Le d\@ploiment du r\@pertoire [C:\Tomcat8\apache-tomcat-8.5.53\webapps\docs] de l'application web s'est termin[0 en [132] ms
05-Jun-2020 12:44:13.754 INFOS [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory D:\ploiment du r\@pertoire d'application web [C:\Tomcat8\apache-tomcat-8.5.53\webapps\examples]
05-Jun-2020 12:44:15.599 INFOS [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Le d\@ploiment du r\@pertoire [C:\Tomcat8\apache-tomcat-8.5.53\webapps\examples] de l'application web s'est termin[0 en [174835] ms
05-Jun-2020 12:44:15.591 INFOS [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory D:\ploiment du r\@pertoire d'application web [C:\Tomcat8\apache-tomcat-8.5.53\webapps\host-manager]
05-Jun-2020 12:44:15.780 INFOS [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Le d\@ploiment du r\@pertoire [C:\Tomcat8\apache-tomcat-8.5.53\webapps\host-manager] de l'application web s'est termin[0 en [189] ms
05-Jun-2020 12:44:15.782 INFOS [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory D:\ploiment du r\@pertoire d'application web [C:\Tomcat8\apache-tomcat-8.5.53\webapps\manager] de l'application web s'est termin[0 en [144] ms
05-Jun-2020 12:44:15.927 INFOS [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory D:\ploiment du r\@pertoire d'application web [C:\Tomcat8\apache-tomcat-8.5.53\webapps\ROOT]
05-Jun-2020 12:44:16.034 INFOS [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Le d\@ploiment du r\@pertoire [C:\Tomcat8\apache-tomcat-8.5.53\webapps\ROOT] de l'application web s'est termin[0 en [107] ms
05-Jun-2020 12:44:16.048 INFOS [main] org.apache.coyote.AbstractProtocol.start [0]marrage du gestionnaire de protocole [http-nio-8080"]
05-Jun-2020 12:44:16.107 INFOS [main] org.apache.catalina.startup.Catalina.startup Server startup in 4960 ms
```

Figure 31 : Démarrage de apache-tomcat

Remarquez (icône au coin supérieur gauche de la figure 31) qu'il ne s'agit pas d'un terminal de ligne de commande ordinaire. C'est un fichier .jar.

Tomcat tourne par défaut sur le port 8080 en localhost dans ce cas. Ouvrons un navigateur et tapons l'adresse :http://localhost :8080

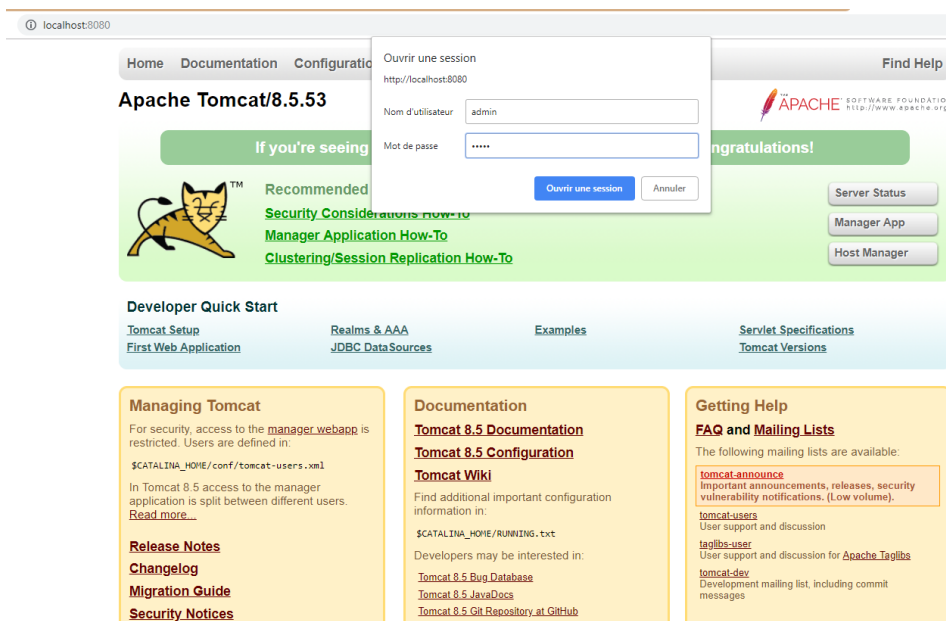




Figure 32 : Démarrage de apache-tomcat en localhost sur le port 8080 dans Google Chrome

Comme cette figure 32 l'indique, après démarrage de tomcat, cliquez en haut, à droite sur « Manager App » et du coup cette petite fenêtre modale d'authentification apparaît. Renseignez les champs Nom utilisateur et mot de passe comme précédemment configurés dans le fichier tomcat-users.xml.

← → ↻

localhost:8080/manager/html

🔍 ⭐

Gestionnaire d'applications WEB Tomcat

Message:

OK

Gestionnaire

Lister les applications

Aide HTML Gestionnaire

Aide Gestionnaire

Etat du serveur

Applications

Chemin	Version	Nom d'affichage	Fonctionnelle	Sessions	Commandes
/	Aucun spécifié	Welcome to Tomcat	true	0	<div>Démarrer Arrêter Recharger Retirer</div> <div>Expirer les sessions inactives depuis ≥ 30 minutes</div>
/CalculWeb-1.0-SNAPSHOT	Aucun spécifié	Archetype Created Web Application	true	0	<div>Démarrer Arrêter Recharger Retirer</div> <div>Expirer les sessions inactives depuis ≥ 30 minutes</div>
/docs	Aucun spécifié	Tomcat Documentation	true	0	<div>Démarrer Arrêter Recharger Retirer</div> <div>Expirer les sessions inactives depuis ≥ 30 minutes</div>
/examples	Aucun spécifié	Servlet and JSP Examples	true	0	<div>Démarrer Arrêter Recharger Retirer</div> <div>Expirer les sessions inactives depuis ≥ 30 minutes</div>
/host-manager	Aucun spécifié	Tomcat Host Manager Application	true	0	<div>Démarrer Arrêter Recharger Retirer</div> <div>Expirer les sessions inactives depuis ≥ 30 minutes</div>
/manager	Aucun spécifié	Tomcat Manager Application	true	1	<div>Démarrer Arrêter Recharger Retirer</div> <div>Expirer les sessions inactives depuis ≥ 30 minutes</div>

Figure 33 : Page d'accueil de apache-tomcat après authentification

Félicitations ! Vous avez maintenant devant vous votre serveur web Tomcat prêt à déployer votre projet. Faites défiler vers le bas la barre droite de défilement et arrêtez-vous comme indiquez sur cette figure 34 :

← → ↻

localhost:8080/manager/html

🔍 ⭐

Expirer les sessions inactives depuis ≥ 30 minutes

/CalculWeb-1.0-SNAPSHOT	Aucun spécifié	Archetype Created Web Application	true	0	<div>Démarrer Arrêter Recharger Retirer</div> <div>Expirer les sessions inactives depuis ≥ 30 minutes</div>
/docs	Aucun spécifié	Tomcat Documentation	true	0	<div>Démarrer Arrêter Recharger Retirer</div> <div>Expirer les sessions inactives depuis ≥ 30 minutes</div>
/examples	Aucun spécifié	Servlet and JSP Examples	true	0	<div>Démarrer Arrêter Recharger Retirer</div> <div>Expirer les sessions inactives depuis ≥ 30 minutes</div>
/host-manager	Aucun spécifié	Tomcat Host Manager Application	true	0	<div>Démarrer Arrêter Recharger Retirer</div> <div>Expirer les sessions inactives depuis ≥ 30 minutes</div>
/manager	Aucun spécifié	Tomcat Manager Application	true	1	<div>Démarrer Arrêter Recharger Retirer</div> <div>Expirer les sessions inactives depuis ≥ 30 minutes</div>

Deployer

Emplacement du répertoire ou fichier WAR de déploiement sur le serveur

Chemin de contexte (requis):

URL du fichier XML de configuration:

URL vers WAR ou répertoire:

Deployer

Fichier WAR à déployer

Choisir le fichier WAR à téléverser

Choisir un fichier

Aucun fichier choisi

Deployer

Configuration

Reliant les fichiers de configuration TLS

Nom d'hôte TLS (optionnel)

Relire

Figure 34 : Phase 1 de déploiement : espace de déploiement

Remarquez ici deux inscriptions principales : « Déployer » et « configuration ». Seule « Déployer » nous intéresse. Dans « Déployer », nous avons deux autres sous-inscriptions, mais celle qui nous intéresse est « Fichier WAR à déployer ». Cliquez sur « choisir un fichier » ; Cela fera un upload vers votre PC et vous allez choisir dans le repository le fichier .War de CalculWeb vu précédemment à la figure 26.

Fichier WAR à déployer

Choisir le fichier WAR à téléverser

Choisir un fichier

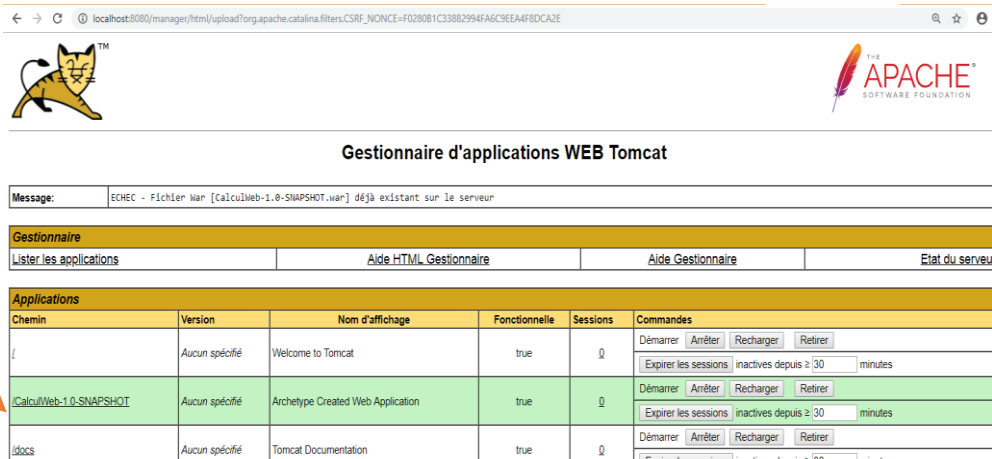
CalculWe . .SHOT.war

Deployer

Figure 35 : Phase 2 de déploiement : upload du .war du repository du PC vers tomcat
 Une fois le projet chargé par Tomcat, cliquez sur « Déployer ».

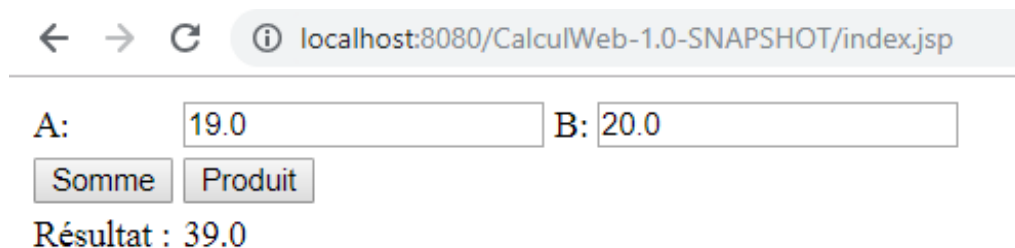
Tomcat déploie le projet et revient en haut de page et nous pouvons voir sur cette figure 36 notre projet CalculWeb déployé :

CalculWeb.war déployé



Chemin	Version	Nom d'affichage	Fonctionnelle	Sessions	Commandes
/	Aucun spécifié	Welcome to Tomcat	true	0	Démarrer Arrêter Recharger Retirer Expirer les sessions inactives depuis ≥ 30 minutes
/CalculWeb-1.0-SNAPSHOT	Aucun spécifié	Archetype Created Web Application	true	0	Démarrer Arrêter Recharger Retirer Expirer les sessions inactives depuis ≥ 30 minutes
/docs	Aucun spécifié	Tomcat Documentation	true	0	Démarrer Arrêter Recharger Retirer

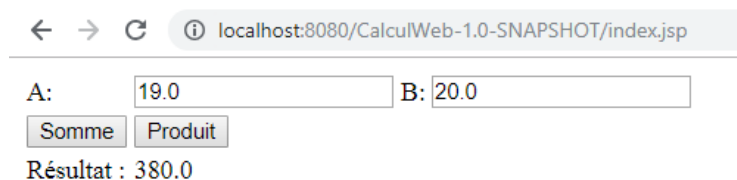
Figure 36 : Phase 3 de déploiement : CalculWeb.war déployé
 Cliquons sur notre projet. Nous sommes directement redirigés vers notre fameuse page index.jsp et nous pouvons faire nos opérations de calcul : Entrons nos chiffres A et B et cliquons sur le bouton « Somme », on obtient le résultat en dessous. Cliquons maintenant sur le bouton « Produit » et le résultat est instantanément mis à jour en dessous.



A: B:

Résultat : 39.0

Figure 37 : Phase de test : Opération de Somme (A+B)



A: B:

Résultat : 380.0

Figure 38 : Phase de test : Opération du produit (A*B)

Bravo ! Vous avez réussi ! On vient donc de gérer le cycle de vie complet d'un projet informatique : de sa création jusqu'à sa production en passant par son déploiement bien évidemment.

Ceci c'est avec un serveur web totalement externe à Maven. Nous allons voir comment faire ce même déploiement jusqu'à l'utilisation avec Maven.

✓ Utilisation de Tomcat embarqué à Maven

Tout est plugin sur Maven rappelez-vous ! Il existe en effet un plugin tomcat pour Maven. Il faut tout d'abord renseigner cette dépendance ou ce plugin dans le pom.xml afin que Maven le télécharge et l'ajoute à notre repository local pour nos futurs déploiements en mode Maven😊.

Dans les balises <plugins>Ajoutez ici</plugins>, ajoutez ceci :

```
<plugin>
  <groupId>org.apache.tomcat.maven</groupId>
  <artifactId>tomcat7-maven-plugin</artifactId>
  <version>2.3-SNAPSHOT</version>
  <configuration>
    <url>http://localhost:8080/manager/text</url>
  </configuration>
</plugin>
```

Et dans les balises <pluginRepositories>Ajoutez ici</ pluginRepositories>, ajoutez ceci :

```
<pluginRepository>
  <id> apache.snapshots</id>
  <name> Apache Snapshots</name>
  <url> http://repository.apache.org/content/groups/snapshots-group/ </url>
  <configuration>
    <releases>
      <enabled>false</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
```

</pluginRepository>

C'est ainsi qu'on ajoute les plugins dans Maven ! C'est très important cette étape :

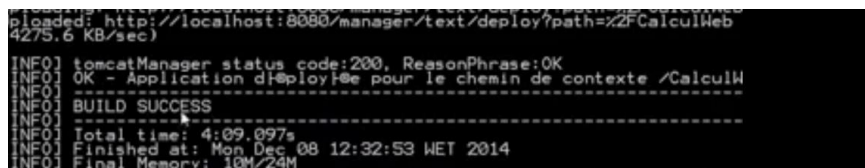
Dans la balise url du « pluginRepositories », nous avons un lien, une adresse. C'est cette adresse que doit utiliser Maven pour télécharger le plugin tomcat7. Cela veut dire qu'il y'a un nouveau serveur central ou un nouveau repository central qui doit être utilisé par Maven sur lequel il va télécharger ce plugin. Vous comprenez maintenant pourquoi la notion de repository est importante. Donc cette adresse pointe vers un repository central situé sur un serveur.

Dans la balise url du « plugin » nous avons également un autre lien mais cette fois ci en localhost : <url><http://localhost:8080/manager/text></url>. Il s'agit en fait des informations pour administrer tomcat via Maven en local.

Bien relançons notre bonne vieille commande « mvn eclipse:eclipse » pour la lecture des nouvelles dépendances ajoutées au pom.xml. Beaucoup de téléchargements vont se faire et après cela, exécutons cette commande de déploiement juste avec Maven et son tomcat embarqué ☺ :

« mvn tomcat7:deploy -Dtomcat.password=admin -Dtomcat.password=admin »

Nous avons deux informations importantes sur cette commande : le paramètre « deploy » et les informations concernant l'utilisateur ou l'administrateur. Le paramètre est ce qui est directement après le goal (nous l'avions déjà dit plus haut que le goal est une appellation des deux points « : ») et peut prendre plusieurs valeurs en fonction des actions à entreprendre.



```
Downloaded: http://localhost:8080/manager/text/deploy?path=/CalculWeb
4275.6 KB/sec)
INFO] tomcatManager status code:200, ReasonPhrase:OK
INFO] OK - Application deployment for the context path /CalculWeb
INFO] BUILD SUCCESS
INFO]
INFO] Total time: 4:09.097s
INFO] Finished at: Mon Dec 08 12:32:53 WET 2014
INFO] Final Memory: 10M/24M
```

Figure 39 : Déploiement de CalculWeb avec Tomcat embarqué de Maven

Il y'a beaucoup de téléchargements qui s'effectuent alors j'ai juste capturé ceci afin que vous voyiez que le déploiement s'est très bien déroulé et vous pouvez vérifier cela avec Google Chrome en tapant « http://localhost :8080 » comme spécifier dans le pom.xml dans la balise fille url de la balise plugin.

Rappelez-vous du paramètre après le goal. Je vous ai dit qu'il peut être modifié en fonction des actions que l'on veut entreprendre. Notre projet est déployé dans Tomcat grâce à Maven et nous voulons actuellement le retirer du serveur tomcat. Il existe un paramètre « [undeploy](#) ».

Il suffit donc juste de remplacer de « deploy » par « undeploy » sur la précédente commande :

« mvn tomcat7:[undeploy](#) -Dtomcat.password=admin -Dtomcat.password=admin »

Ouffffffff !!! Nous avons géré ce même cycle de vie de notre projet uniquement avec Maven jusqu'à la fin. C'est pas tout ! Maven a encore tellement de cartouches et de puissances.

Nous allons amorcer d'autres concepts très importants qui concernent la documentation et la génération du site de notre projet par Maven.

✓ Site du projet : [mvn site](#)

La commande « mvn site » est très fréquemment utilisée pour la génération du site du projet. En effet, il ne s'agit pas d'un site comme vous l'imaginez mais bien plus que cela. Positionnez-vous sur le répertoire /CalculWeb de votre projet et exécutez « mvn site ».

Il est en effet possible que vous ayez des erreurs ou des échecs ! Mais si tout se passe bien chez-vous, alors c'est bien, sinon suivez-moi 😊 :

Ajoutez-ceci comme vu précédemment dans la balise <plugins> (avec un 's') au pom.xml :

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-site-plugin</artifactId>
  <configuration>
    <reportPlugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-project-info-reports-plugin</artifactId>
        <version>2.0.1</version>
      </plugin>
    </reportPlugins>
  </configuration>
</plugin>
```

Lancez « mvn eclipse:eclipse ». Quelques téléchargements s'effectuent car c'est un nouveau plugin que nous venons d'ajouter au pom.xml.

Ouvrons le dossier /target de notre projet, un dossier /site a été créé et contient des fichiers .html qui ont été générés :

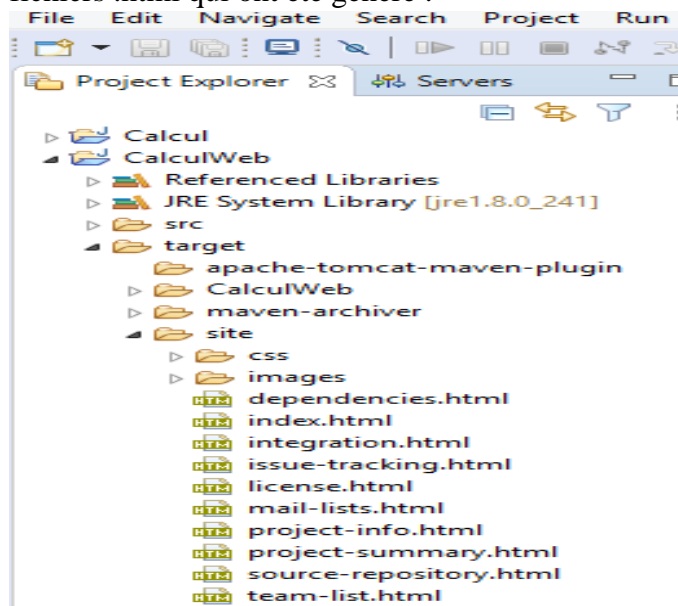


Figure 40 : Génération des fichiers html du site du projet CalculWeb

Ouvrons index.html directement dans Eclipse ou avec Chrome.

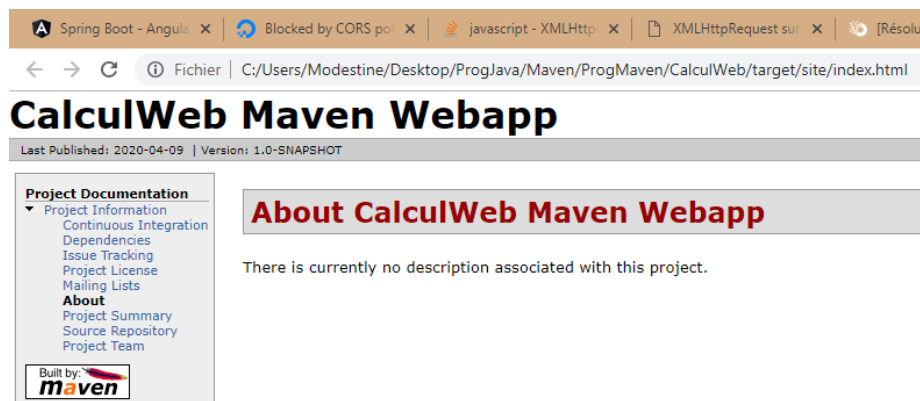


Figure 41 : Site du projet CalculWeb

Cliquons par exemple sur l'onglet à gauche « Dependencies », nous voyons sur cette figure 42, toutes les dépendances de notre projet. Nous voyons également le projet Calcul dans la liste des dépendances, nous voyons que nous avons utilisé Junit pour les tests unitaires, etc... Tout est là.

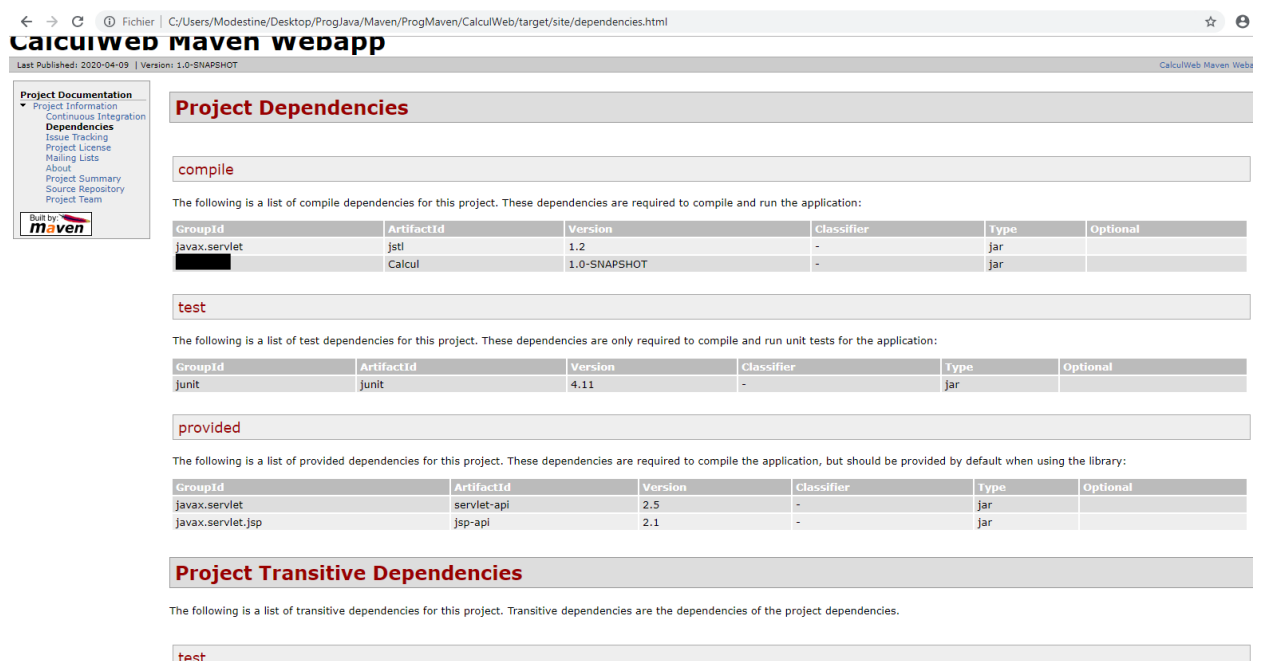


Figure 42 : Liste des dépendances du projet CalculWeb vue du site généré


✓ Documentation

La documentation est déjà là ! Il n'y a pas de commande pour la documentation mais juste un clic sur un onglet du site généré nous le donne. Cliquez sur « Project Information »

← → C Fichier | C:/Users/Modestine/Desktop/ProgJava/Maven/ProgMaven/CalculWeb/target/site/project-info.html ☆ e o

CalculWeb Maven Webapp

Last Published: 2020-04-09 | Version: 1.0-SNAPSHOT CalculWeb Maven Webapp

Project Documentation
 Project Information
 Continuous Integration
 Dependencies
 Issue Tracking
 Project License
 Mailing Lists
 About
 Project Summary
 Source Repository
 Project Team


Project Information

This document provides an overview of the various documents and links that are part of this project's general information. All of this content is automatically generated by Maven on behalf of the project.

Overview

Document	Description
Continuous Integration	This is a link to the definitions of all continuous integration processes that builds and tests code on a frequent, regular basis.
Dependencies	This document lists the projects dependencies and provides information on each dependency.
Issue Tracking	This is a link to the issue management system for this project. Issues (bugs, features, change requests) can be created and queried using this link.
Project License	This is a link to the definitions of project licenses.
Mailing Lists	This document provides subscription and archive information for this project's mailing lists.
About	There is currently no description associated with this project.
Project Summary	This document lists other related information of this project
Source Repository	This is a link to the online source repository that can be viewed via a web browser.
Project Team	This document provides information on the members of this project. These are the individuals who have contributed to the project in one form or another.

Copyright © 2020. All Rights Reserved.

Figure 43 : Documentation du projet CalculWeb vue du site généré

Félicitations ! Félicitations ! et Félicitations ! Vous avez fait et vu les bases et les fonctionnalités avancées de Maven. Vous connaissez maintenant gérer tout le cycle de vie d'un projet avec Maven de sa création à sa mise en production.

Conclusion

Nous sommes à la fin de ce cours ou tutoriel si vous le voulez et nous avons appris beaucoup de choses et de concepts importants du génie logiciel avec Maven. Notez qu'il existe des concurrents de Maven tels que Gradle, Ant et autres. Le choix vous revient car chaque outil a ses avantages et inconvénients. Toute fois notez qu'un projet informatique qui n'utilise pas l'un de ces outils d'optimisation ne saurait être pérenne dans le temps et la gestion ou l'évolutivité d'un tel projet est difficile voire impossible. La maintenance sera un enfer, sans oublier que la mise sur pied d'une équipe de développement sur un tel projet est un véritable casse-tête.

Je vous donne ce joli cadeau d'aurevoir pour mieux vous amuser avec tomcat embarqué dans Maven :

- `tomcat7:deploy` : Deploy a WAR to Tomcat.
- `tomcat7:deploy-only` : Deploy a WAR to Tomcat without forking the package lifecycle.
- `tomcat7:exec-war` : Create a self executable jar file containing all necessary Apache Tomcat classes. This allows for using just `java -jar mywebapp.jar` to run your webapp without needing to install a Tomcat instance.
- `tomcat7:exec-war-only` : Same as `exec-war` goal without forking the package lifecycle.
- `tomcat7:help` : Display help information on `tomcat7-maven-plugin`.
- `tomcat7:redeploy` : Redeploy a WAR in Tomcat.
- `tomcat7:redeploy-only` : Redeploy a WAR in Tomcat without forking the package lifecycle.
- `tomcat7:run` : Runs the current project as a dynamic web application using an embedded Tomcat server.
- `tomcat7:run-war` : Runs the current project as a packaged web application using an embedded Tomcat server.
- `tomcat7:run-war-only` : Same as `run-war` goal without forking the package cycle.
- `tomcat7:shutdown` : Shuts down all possibly started embedded Tomcat servers.
- `tomcat7:standalone-war` : Will create an executable war file with embedded Tomcat that is also capable of being deployed elsewhere.
- `tomcat7:standalone-war-only` : Will create an executable war file with embedded Tomcat that is also capable of being deployed elsewhere.
- `tomcat7:undeploy` : Undeploy a WAR from Tomcat.

Figure 44 : Quelques Goal de Maven pour tomcat

Aucune œuvre humaine n'étant parfaite, vos remarques, suggestions et améliorations sont les bienvenues.

Vous pouvez laisser vos mails à cette adresse : flessentkouamou@gmail.com