1. **PRESENTATION DE MAVEN**

Apache Maven (couramment appelé Maven) est un outil de [gestion et d'automatisation de production des projets logiciels](https://fr.wikipedia.org/wiki/Moteur_de_production) [Java](https://fr.wikipedia.org/wiki/Java_(langage)) en général et [Java EE](https://fr.wikipedia.org/wiki/Java_EE) en particulier. Il est utilisé pour automatiser l'[intégration continue](https://fr.wikipedia.org/wiki/Int%C3%A9gration_continue) lors d'un développement de logiciel. Maven est géré par l'organisation [Apache Software Foundation](https://fr.wikipedia.org/wiki/Apache_Software_Foundation). L'outil était précédemment une branche de l'organisation [Jakarta Project](https://fr.wikipedia.org/wiki/Apache_Jakarta).

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.

Un élément clé et relativement spécifique de Maven est son aptitude à fonctionner en réseau. Une des motivations historiques de cet outil est de fournir un moyen de synchroniser des projets indépendants : publication standardisée d'information, distribution automatique de modules [jar](https://fr.wikipedia.org/wiki/JAR_(format_de_fichier)). Ainsi en version de base, Maven peut dynamiquement télécharger du matériel à partir des dépôts logiciels connus. Il propose ainsi la synchronisation transparente de modules nécessaires.

Maven1 et Maven2 ont été développés en parallèle mais les versions ultérieures sont basées sur la structure de la deuxième version.

1. **Project Object Model (POM)**

Maven utilise un paradigme connu sous le nom de Project Object Model afin de décrire un projet logiciel, ses dépendances avec des modules externes et l'ordre à suivre pour sa production. Il est livré avec un grand nombre de tâches pré-définies, comme la compilation de code Java ou encore sa modularisation. Chaque projet ou sous-projet est configuré par un POM qui contient les informations nécessaires à Maven pour traiter le projet (nom du projet, numéro de version, dépendances vers d'autres projets, bibliothèques nécessaires à la compilation, noms des contributeurs, etc.). Ce POM se matérialise par un fichier pom.xml à la racine du projet. Cette approche permet l'héritage des propriétés du projet parent. Si une propriété est redéfinie dans le POM du projet, elle recouvre celle qui est définie dans le projet parent. Ceci introduit le concept de réutilisation de configuration. Le fichier pom du projet principal est nommé *pom parent*. Il contient une description détaillée de votre projet, avec en particulier des informations concernant le versionnage et la gestion des configurations, les dépendances, les ressources de l'application, les tests, les membres de l'équipe, la structure et bien plus.

1. **Configuration d’un projet MAVEN**

Maven impose une arborescence et un nommage des fichiers du projet. Ces conventions permettent de réduire la configuration des projets, tant qu'un projet suit les conventions. Si un projet a besoin de s'écarter de la convention, le développeur le précise dans la configuration du projet.

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

* /src : les sources du projet
* /src/main : code source et fichiers source principaux
* /src/main/java : code source
* /src/main/resources : fichiers de ressources (images, fichiers annexes, etc.)
* /src/main/webapp : [webapp](https://fr.wikipedia.org/wiki/Application_web)du projet
* /src/test : fichiers de test
* /src/test/java : code source de test
* /src/test/resources : fichiers de ressources de test
* /src/site : informations sur le projet et/ou les rapports générés suite aux traitements effectués
* /target : fichiers résultat, les binaires (du code et des tests), les packages générés et les résultats des tests

1. **Cycle de vie de Maven**

Dans le cycle de vie ‘par défaut’ d’un projet Maven, les phases les plus utilisées sont :

* validate : vérie les prérequis d’un projet maven
* compile : compilation du code source
* test : lancement des tests unitaires
* package : assemble le code compilé en un livrable
* install : partage le livrable pour d’autres projets sur le même ordinateur
* deploy : publie le livrable pour d’autres projets dans un ‘repository’ distant

Les phases s’exécutent de façon séquentielle de façon à ce qu’une phase dépende de la phase précédente. Par exemple, le lancement par l’utilisateur de la phase test (mvn test) impliquera le lancement préalable par maven des phases ‘validate‘ et ‘compile‘.

1. **CAS PRATIQUE DE CREATION D’UN PROJET ARCHETYPES SUR GITHUB**
2. **Création d’un projet Maven ARCHETYPES**

La création d'un archétype est un processus assez simple. Un archétype est un artefact très simple, qui contient le prototype de projet que vous souhaitez créer. Un archétype est composé de :

* un [descripteur d'archétype](http://maven.apache.org/archetype/archetype-models/archetype-descriptor/archetype-descriptor.html) (archetype-metadata.xml dans le répertoire :) src/main/resources/META-INF/maven/. Il répertorie tous les fichiers qui seront contenus dans l'archétype et les catégorise afin qu'ils puissent être traités correctement par le mécanisme de génération de l'archétype.
* les fichiers prototypes qui sont copiés par le plugin archétype (répertoire : src/main/resources/archetype-resources/)
* la pom prototype (pom.xmldans : src/main/resources/archetype-resources)
* un pom pour l'archétype (pom.xmldans le répertoire racine de l'archétype).

Pour créer un archétype, procédez comme suit :

invoquer l'archétype via mvn archetype:generate en ligne de commande. Tout ce que vous devez spécifier est un groupId, artifactIdet version.

Voici un exemple pom.xmld'artefact d'archétype :

1. <groupId> org.example.demo </groupId>
2. <artifactId> maven </artifactId>
3. <version> 1.0- INSTANTANÉ </version>

ou mvn archetype:generate -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.1 en ligne de commande. Tout ce que vous devez spécifier est un groupIdetartifactId.

Voici un exemple pom.xmld'artefact d'archétype :

1. <groupId> org.example.demo </groupId>
2. <artifactId> maven </artifactId>
3. <version> 1.0-SNAPSHOT </version>
4. <packaging> jar </packaging>

Puis dans une console à la racine du projet (répertoire maven), là où se trouve le fichier pom.xml faire :mvn package. Il s’agit de laisser Maven compiler notre projet en générant le Jar.

Ou alors vous pouvez procéder comme suite :

* **Créez un nouveau projet et de pom.xml pour l'artefact d'archétype en ligne de commande** : mvn archetype:create-from-projet
* **Transformez l’archetype**
* **Mise à disposition de l’archetype en local**

Une fois que l’archetype a été modifié pour répondre aux besoins, il n’y a plus qu’à exécuter la commande mvn install pour déployer l’archetype dans le repository local et la commande mvn deploy pour le déployer dans le repository partagé.

Une fois l’archetype déployé dans le repository adéquate, il n’y a plus qu’à l’invoquer via la commande mvn archetype:generate.

1. **Déploiement sur Github**

Vous pouvez le déployer sur Github de la manière suivante :

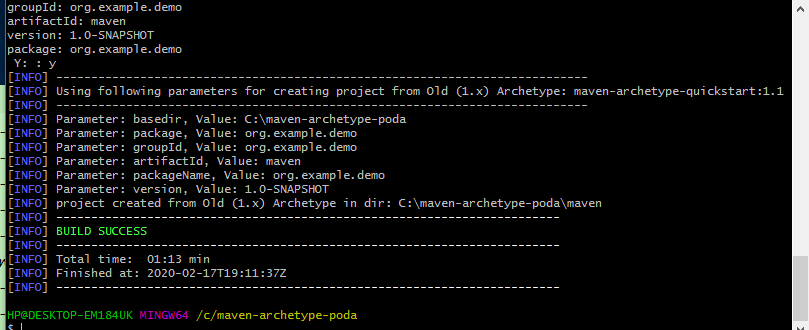
* **Créer et ajouter son projet sous *github (Create a New Repository sans fichier Readme)***

Exemple de création d’un projet "maven-archetype-poda" via des commandes *git* :

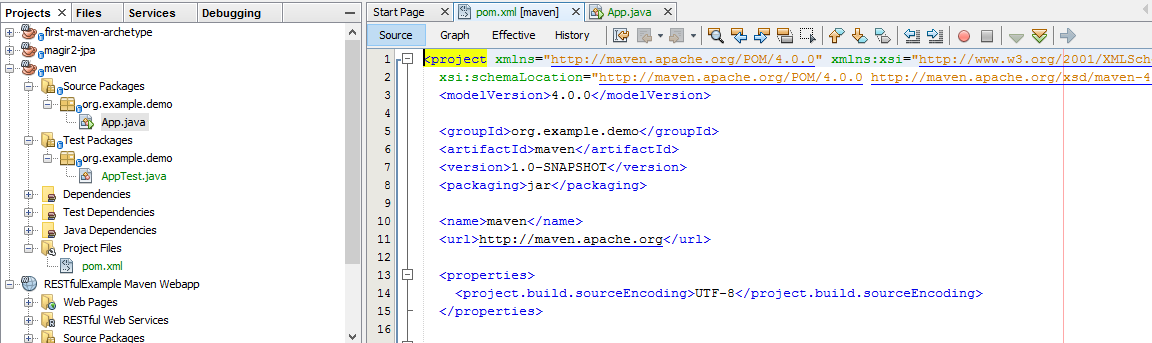
* ***git init***
* ***git add .***
* ***git commit –m ’’first commit’’***
* **git remote add origin https://github.com/IhnyEricka/maven-archetype-poda.git**
* ***git push -u origin master***

1. **QUELQUES CAPTURES D’ECRAN**

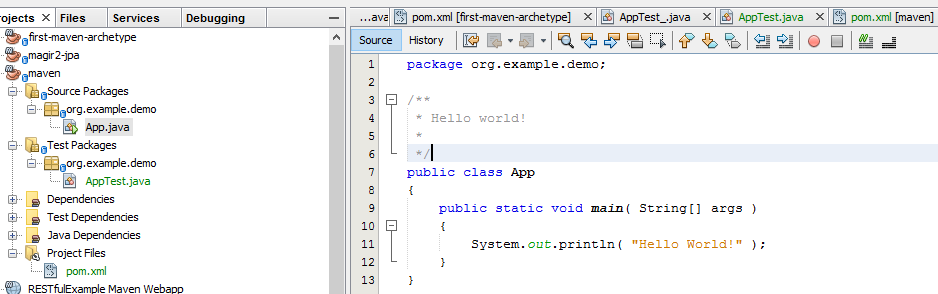
* Création du projet maven archetype : mvn archetype:generate



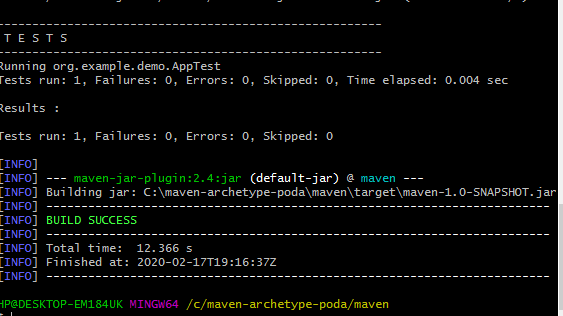
* Arborescence du projet et pom.xml



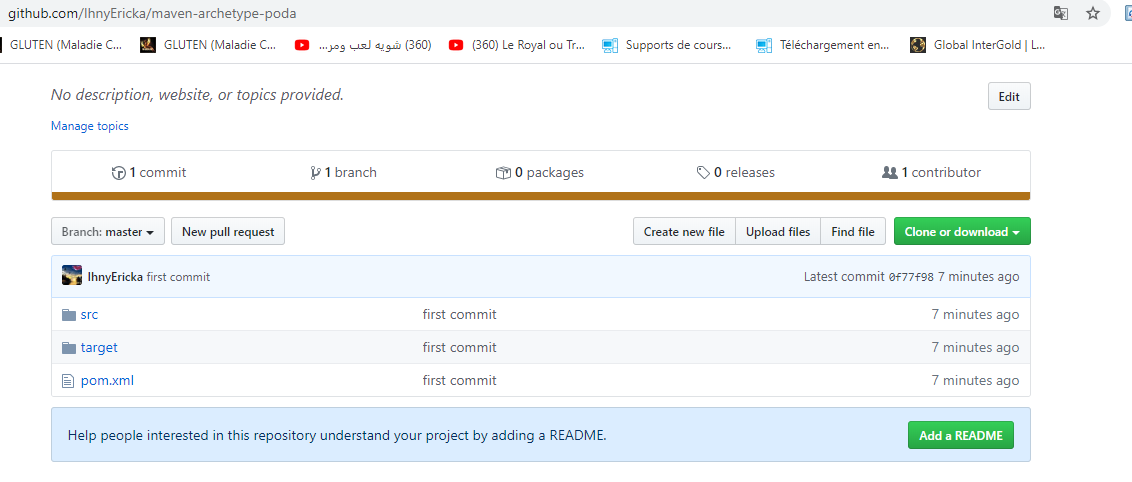
* Le fichier App.java



* Le JAR de notre projet pour exécuter le fichier App.java



* Dépôt sur Github



* Exécution du fichier App.java

