**AFCEPF – Cours du 29 octobre 2018 .**

Table des matières

[AFCEPF – Services web - Cours du 29 octobre 2018 – D. Defrance 2](#_Toc529174077)

[Présentation de base des web-services 2](#_Toc529174078)

[RMI – page 113 du pdf EJB3\_JPA\_RMI\_JMS\_CDI\_jboss7\_et\_wildFly9 3](#_Toc529174079)

[TP sur les RMI (remote method invocation) 3](#_Toc529174080)

[Projet avec EJB 6](#_Toc529174081)

[WSDL : WebService Description Language 10](#_Toc529174082)

[TP servEjb 10](#_Toc529174083)

[SopaUI 11](#_Toc529174084)

[Création d'un projet servSoap 12](#_Toc529174085)

[CXF 13](#_Toc529174086)

[Projet servSpringCxf 13](#_Toc529174087)

[Projet en Rest 17](#_Toc529174088)

[Nouveau projet WSRest – Commencé par le prof : à importer 18](#_Toc529174089)

# AFCEPF – Services web - Cours du 29 octobre 2018 – D. Defrance

## Présentation de base des web-services

SOAP : passe par XML

REST : passe par JSon

Il peut y avoir une application « cœur de métier » qui blablate avec d’autres applications, que l’entreprise va se procurer directement sur le marché (pas personnalisée),  
Le format des données peut varier, on utilise alors des adaptateurs (JSon)

On doit modéliser : UML (Unified Modeling Language) et **BPML**, ce dernier est beaucoup plus proche des webservices  
BPM : Business Process Management.  
SOA : Service Oriented Application

Un service est une URL programmée pour lancer un process, que les applications appellent pour obtenir le service.

Les classes java DTO permettent de véhiculer les données entre les serveurs d’application.  
DTO : Data Transfert Object.

Existence d’un serveur intermédiaire parfois, justifié par le côté sécurités ou le côté statistique.

CXF : technologie Java permettant de programmer de ws REST et SOAP.  
On peut aussi utiliser Spring MVC.

SOAP : les requêtes sont encodées en XML et reviennent en XML  
WSDL : Web Service Description Language

REST : utilisent http au maximum, pas de format de données (moins typé que SOAD du coup).

Les services REST sont plus simples et efficaces, ce qui explique le déclin du SOAP en faveur du REST.

SOA récents :

* Client SPA : Single Page Application 🡪 comme Angular, appelle un ws REST.
* WOA : Web Oriented Architecture
* Architecture « micro-service »

## RMI – page 113 du pdf EJB3\_JPA\_RMI\_JMS\_CDI\_jboss7\_et\_wildFly9

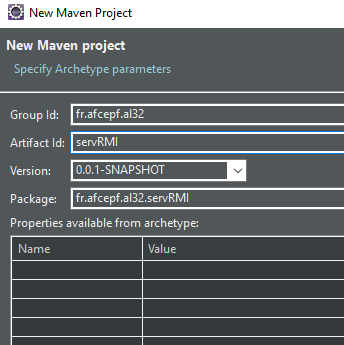
RMI : Remote Method Invocation

On va lancer un serveur de noms,

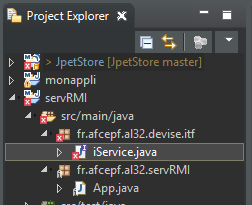
Exception technique : java.rmi.remoteException  
On peut avoir un cable mal branché par ex.

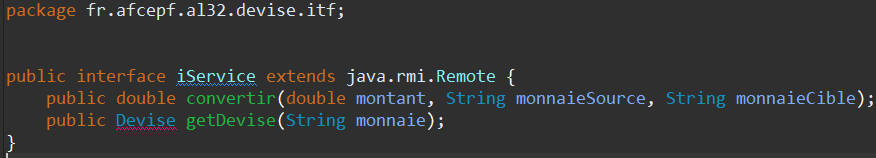
On doit implémenter dans les interfaces : remote et satellisable

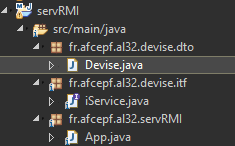
## TP sur les RMI (remote method invocation)



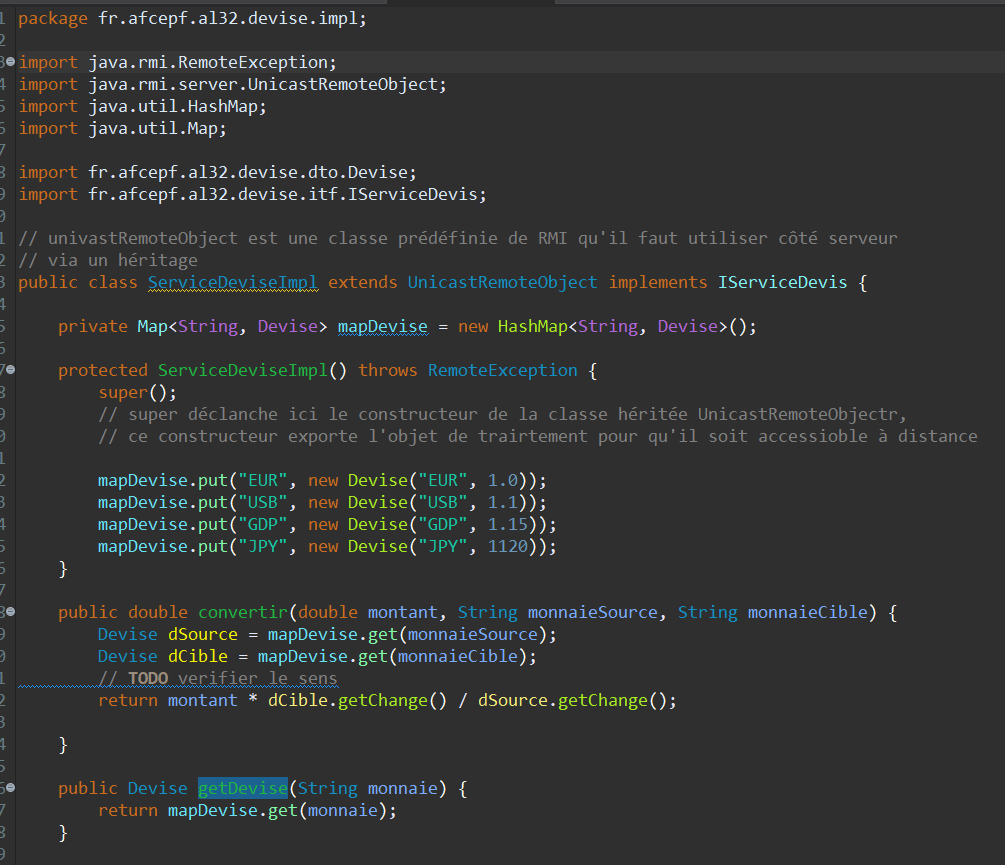
On configure le pom.xml

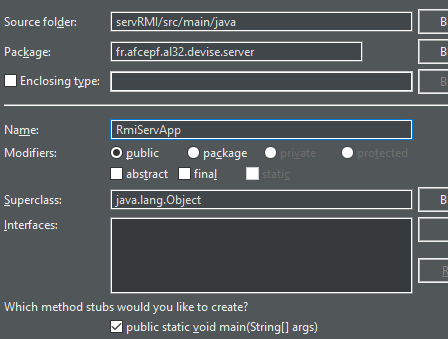
On ajoute un package pour les interfaces, et on y crée notre première interface qui extends l'interface java.rmi.remote :  




Création de la classe Devise dans un autre package :  


Implémentation :



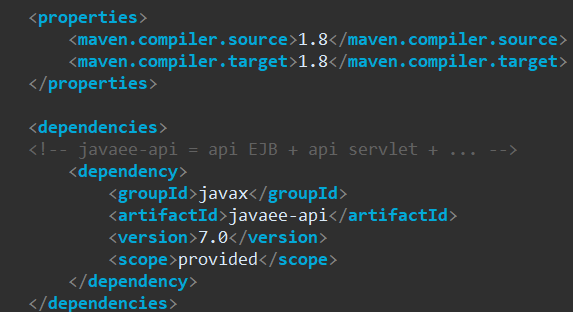
Création ensuite dans un package fr.afcepf.al32.devise.server d'un main  


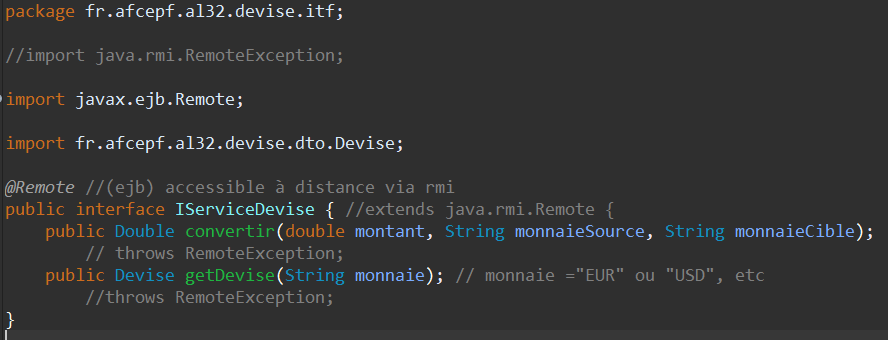
Il manquait dans notre code des throws exception etc.

TP dispo ainsi que sa solution.

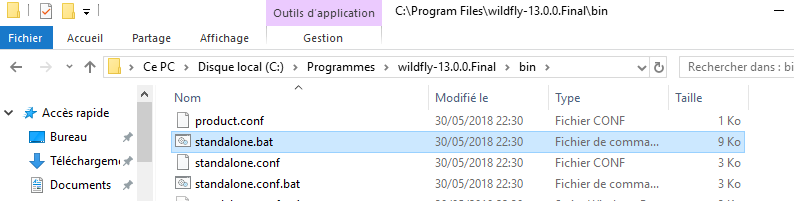
## Projet avec EJB

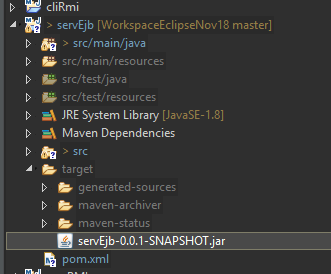
Dans le pom.xml on ajoute toujours le compiler / target en 1.8, mais on ajoute l'ejb

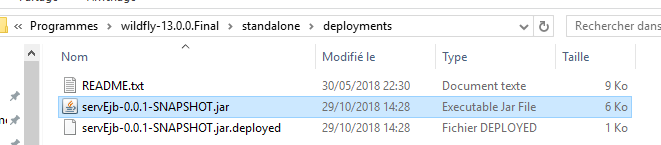


Dans iServiceDevise récupéré du projet précédent, on rajoute @Remote et on enlève les throws  


Idem dans l'implémentation :  

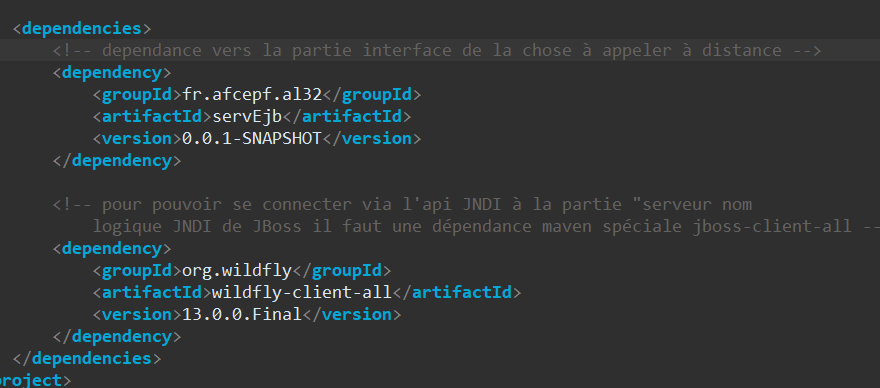

On installe Wildfly  


On peut ensuite tester que tout est ok,  
On run le standalone.bat (ci-dessus)  
On récupère le .jar compilé par Eclipse (dans les target)  


On le copie-colle dans le dossier pathWildly 🡪 standalone 🡪 deployments  


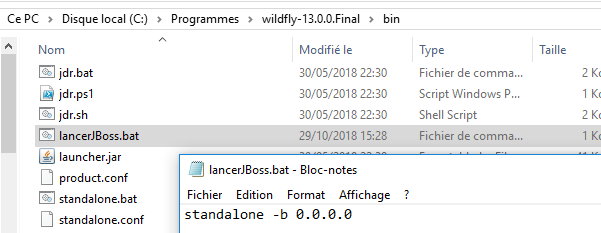
Le coller là après avoir lancé le standalone.bat (donc le serveur) permet de direct vérifier qu'on a bien géré.

Création d'un autre projet maven, son pom :



Voici un p'tit bayou utile :  


On peut créer un .bat spécial pour lancer jboss avec moins de sécurité



Payload : charge utile, comme le DTO (data transfert object), c'est ce qui est facilement transportable via le réseau

## WSDL : WebService Description Language

On n'a pas forcément le même langage du côté client et serveur, les types de données ne sont pas forcément non plus les mêmes (mêmes noms),  
Il faut se mettre d'accord au niveau des types compatibles ;  
WSDL est une norme reconnue.  
C'est une sorte de contrat entre l'appelant et l'appelé.

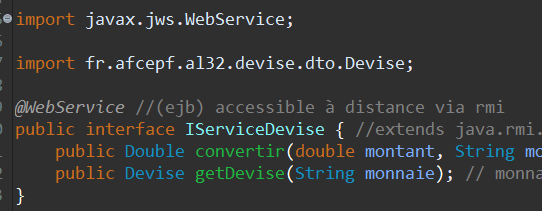
EndPoint : reçoit une requête SOAP sur une URL, transforme en java, et renvoie la réponse en SOAP (XML).

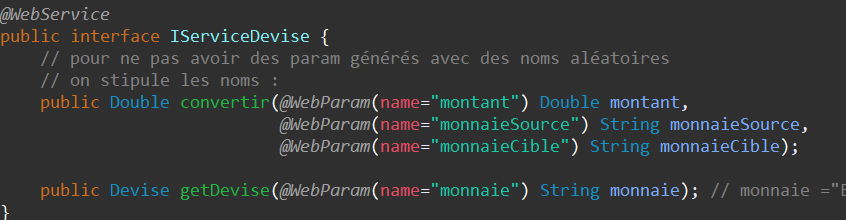
Même principe qu'avec RMI, sauf que communique au milieu en java.

SEI : Service Endpoint Interface

On crée un nouveau projet

## TP servEjb



Utilisation de @webParam(name="nomParam") :  


@WebService(targetNameSpace="", endPointInterface="")

Programme :

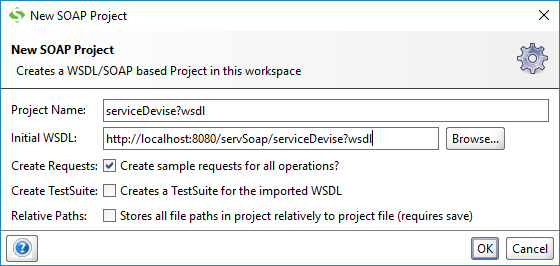
* RMI, EJB, @Remote (années 2000, déprécié)
* SOAP, WSDL 🡪 servSoap (jdk 1.8)  
   soap-ni  
   appel java/cliSoap  
   Spring + CXF
* REST : avec JAX-AS et CXF   
   ou avec SpringBoot + SpringMVC
* REST : appel java, appel js/jquery, appel Angular 6 + RxJs
* Sécurité avec les tokens

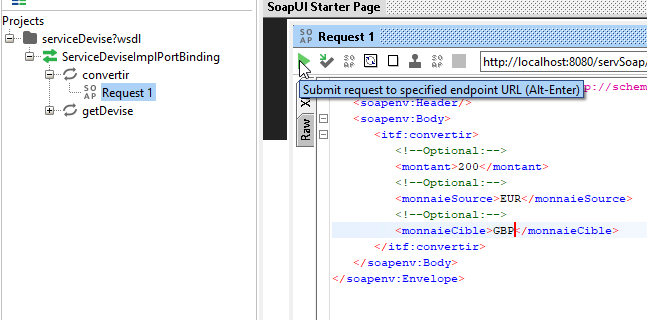
Jetty : mini serveur Tomcat inclus dans la jdk

## SopaUI

Open source,

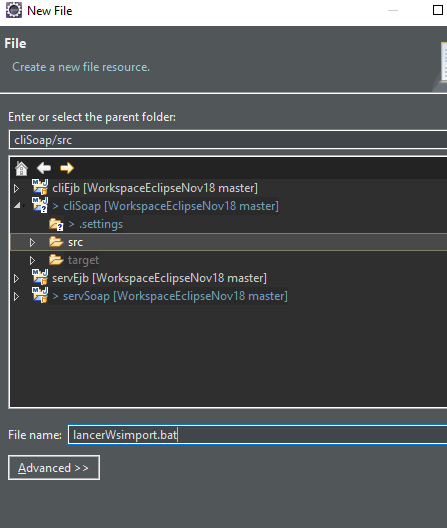
Analyse du wsdln, nouveau projet (Ctrl+N), on copie l'adresse wsdl de notre projet et tout se remplit automatiquement 🡪

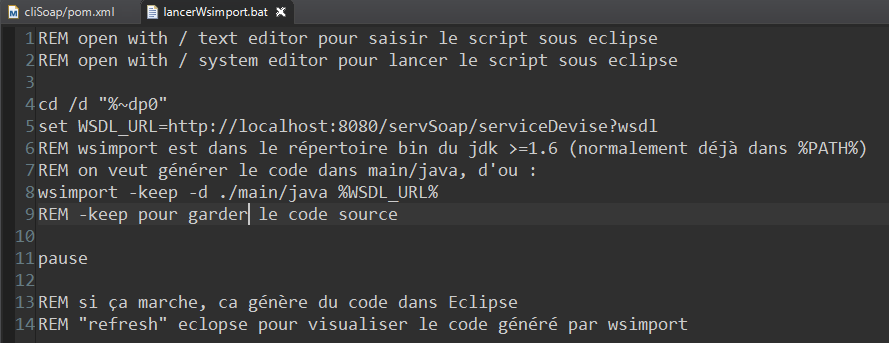


On peut aller dans l'architecture du projet, fonction convertir, on met des valeurs, la réponse apparait !!  


## Création d'un projet servSoap

On crée un .bat dans le dossier src



On y écrit ces douces lignes :  


On lance ainsi le serveur, et ces manips sont valables pour tout langage !

On a le code généré qui est dispo après refresh d'Eclipse !!

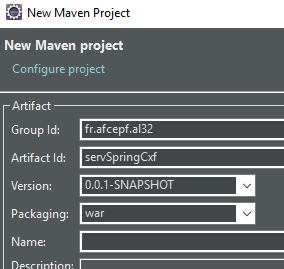
## CXF

Technologie qui permet d'encore plus simplement mettre en œuvre des p'tits webservices de grande qualité.

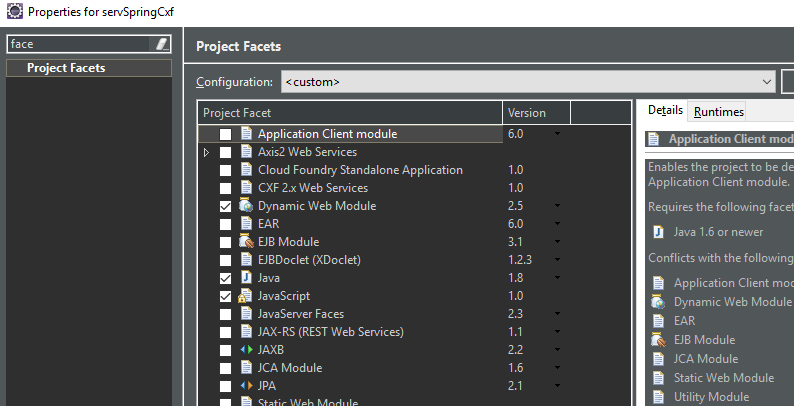
Permet si j'ai bien compris de créer une bdd dans la foulée grace à la liaison de Spring et de CXF

## Projet servSpringCxf

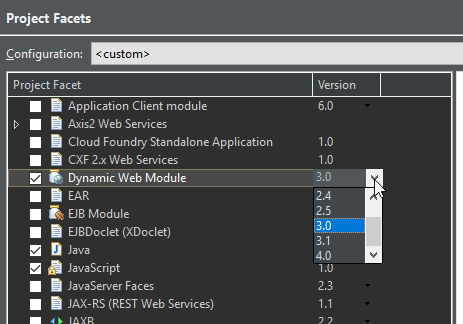
Packaging war

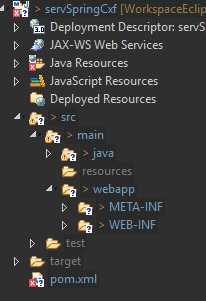


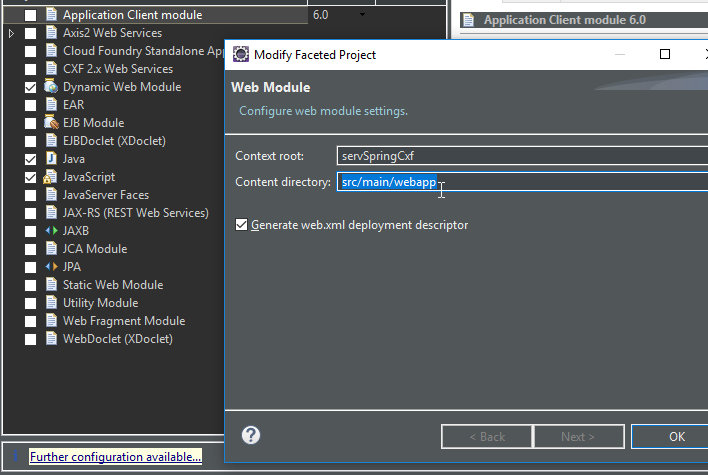
Dans le pom.xml  


Project facet :  


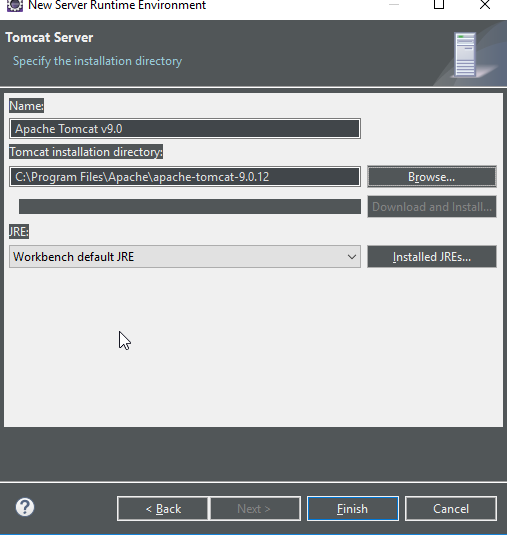
En gros c'est pour générer le dossier META-FIN et WEB-INF incluant le dossier webapp ainsi que le fichier web.xml.



Après avoir décoché, apply ;  
Recoché, choisi la version 3.0, apply, les dossiers sont créés !  


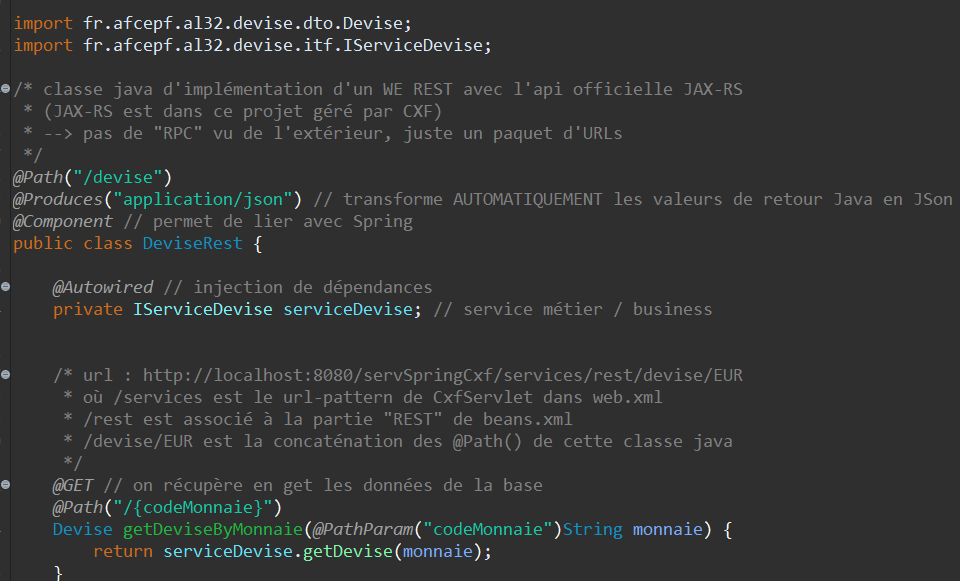
Nb : on aura préalablement choisis quelques configs, coché la case, etc 🡪  


Dans un projet maven, on fou la config xml dans le côté ressources, donc dans src/main/resources

Ajout du serveur pour lancer le :  


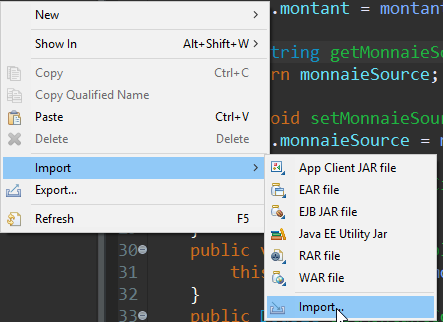
## Projet en Rest

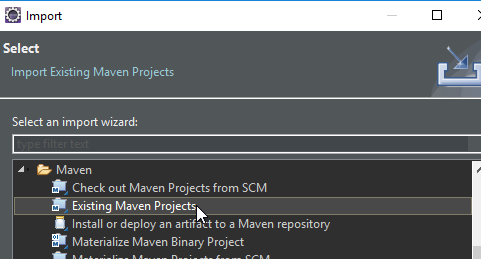
Entre autre, quelques annotations importantes pour lier (@Autowired), pour récupérer les datas @GET, @Path etc

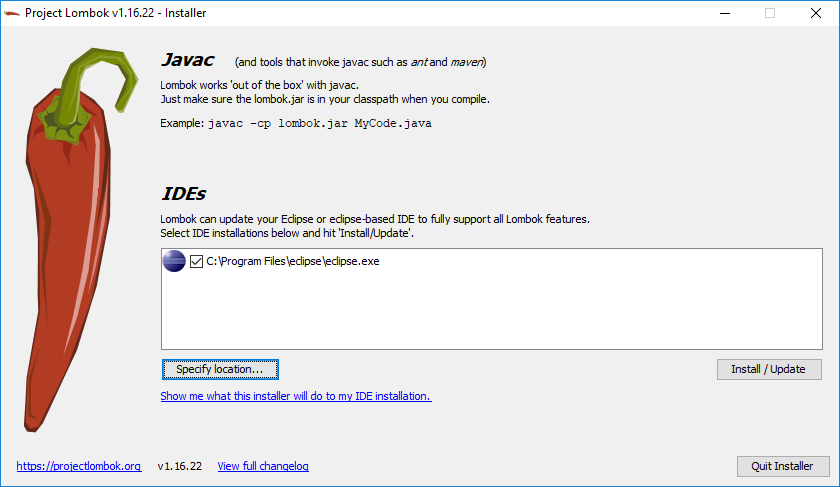


URI : Unified Resource Identifier (identité)  
URL : Unified Resource Localisation (juste son adresse)  
URN : Unified Resource Name

## Nouveau projet WSRest – Commencé par le prof : à importer



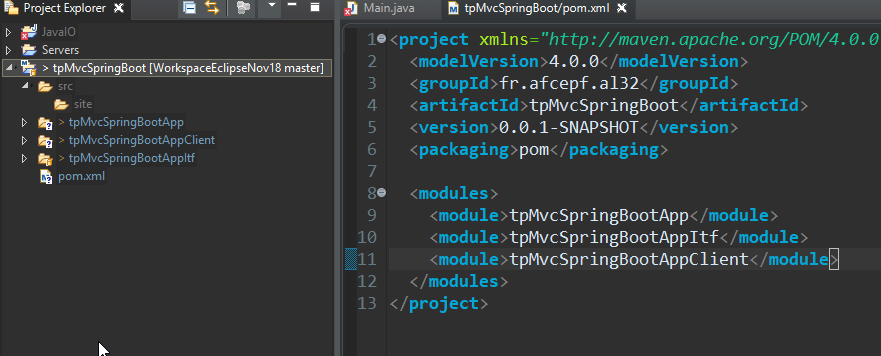
Existing Maven Project + chemin :  


On installe ensuite Lambok,  
Maven update 🡪 on regarde le chemin dans les dependencies 🡪 go explorer et lancer le .jar 🡪 on installe 🡪 redémarrer Eclipse  


5 novembre 2018 :  
On reste sur les WS, on teste avec le projet Blagues, avec du AJAX JSON JQuery.

On a mis en œuvre dans un projet le design pattern Business Delegate.

On crée un projet en packaging pom, qui incluera les tp déjà faits :



Il faut aussi que les sous projets, déclarent le projet principal en parent !  


Programme :  
code WS REST via Spring MVC (@RestController, …)  
appels via <a href> et tests via Postman  
appels javascript/ajax avec et sans jquery  
appels java via RestTemplate de Spring  
Business Delegate, réorganisation multi-module  
  
On passe à :  
Appli Spring-Boot complète avec Service + Dao + JPA  
Appels depuis Angular6 et RXJS

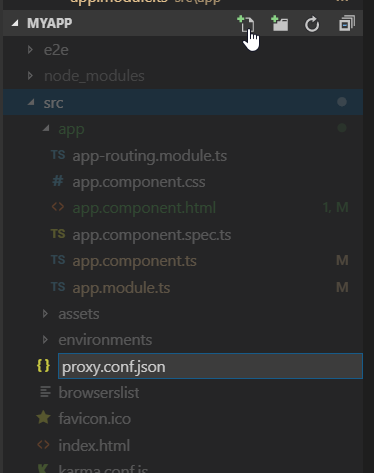
Public interface IDaoBlague extends **CrudRepository<BlaguePersistente, Long>**

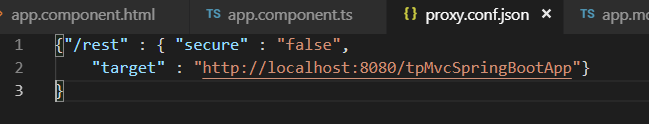
CrudRepository est inclus dans Spring-Data, permettant d'avoir directement des méthodes héritées (findById(), findAll(), save(), deleteById()…)

Dans un projet moderne, il est utile de faire implémenter nos interfaces de CrudRepository

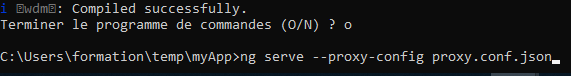
## Projet Angular

Astuce : création d'un fichier configs pour gérer les url



On y met :   


Voir le support de cours.

Ctrl + C dans la ligne de commande ou on était déjà, pour arrêter,  
Et on relance :  


Réinstaller le répertoire de module :  
npm install  
A ajouter dans le gitignore par exemple !!

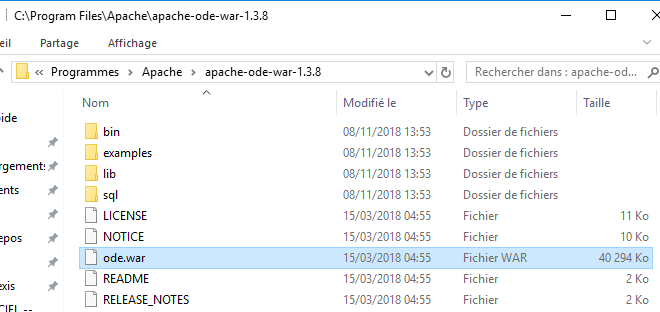
Pour coder en Angular, il nous faut npm.  
npm -g @angular/cli 🡪 pour installer toutes les commandes angular.

Ng new -monApp 🡪 démarrer une appli (page 10 du poly quoi)

Url de base : 4200

## Ode.war à installer

Pour pouvoir faire tourner un bail.



Pour le faire marcher avec Tomcat, il faut que le ode.war soit dans un rep d'un tomcat normal,  
Donc on le met dans notre webapps de notre tomcat.

