Formation Architecture Logicielle

Exercices

Formateurs :

Frantz Degrigny, frantz.degrigny@keyconsulting.fr

Méric Garcia, meric.garcia@keyconsulting.fr

Téléphone : 02.30.96.02.75

http://www.keyconsulting.fr

Sommaire

Table des illustrations

2. Interagir avec le SI

1. JMS - premier tests.

1. Installation

Récupérer apache ActiveMQ :

http://activemq.apache.org/activemq-5120-release.html

Décompresser l'archive.

Démarrer le service : en ligne de commande à la racine du dossier :

⇨ ./bin/activemq qtart

Vérifier que le service est bien démarré : http://127.0.0.1:8161/admin/

Login : admin, Mdp : admin

2. Ecriture et lecture d'un message :

La classe JmsServiceHelper permet d'accès aux méthode d'enregistrement et de lecture des files JMS.

public void send(String text) : permet d'envoyer un message sous forme de texte dans la première file

public void sendForListener(String text) : permet d'envoyer un message sous forme de texte dans la file sur laquelle écoute un listener

public String next() : récupère le message suivant disponible dans la première file.

Dans un premier temps, nous allons envoyer la liste des auteurs de calculs dans la file JMS. Au prochain redémarrage de l'application nous récupérerons cette liste et l'afficherons.

Les deux fonctions à modifier se trouve dans le contrôleur (classe Controller) :

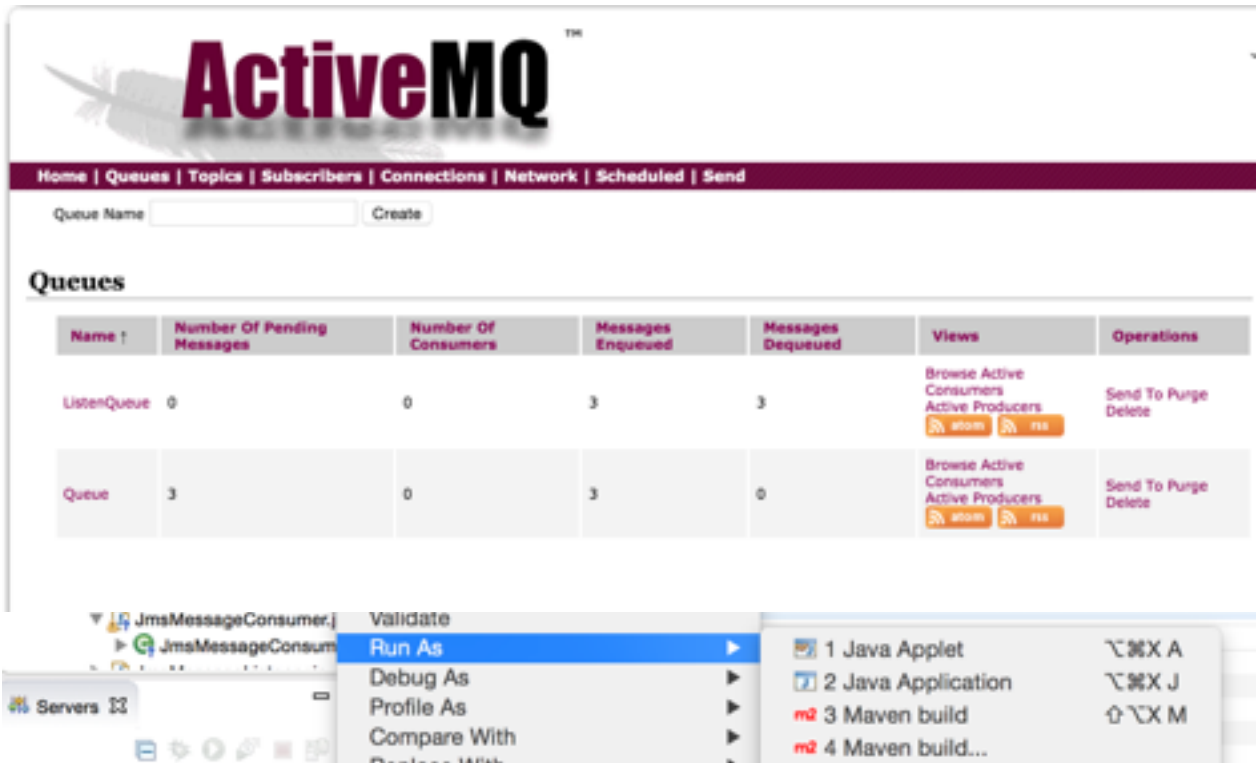
private void addEnQueuedMessageTo(StringJoiner sj)

et

private void sendAuthorToJmsServices(Calcul calcul)

Une fois les modifications effectuées, lancer le programme :

Si le développement est correct, le résultat sera le suivant après trois calculs effectués :



Que s'est-il passé ? Quelle la différence entre le listener et le consumer (classe JmsMessage-Listener et JmsMessageConsumer)?

2. JMS - passage d'un objet.

1. Développpement

Des files JMS ont été ajouté dans la configuration afin de pouvoir gérer les calculs.

Il faut désormais modifier trois classes afin d'utiliser ses files :

JmsMessageSender : méthode send(Calcul calcul)

JmsMessageConsumer : méthode getFollowingCalcul()

Controller : méthode sendCalculAndAuthorToJmsServices(Calcul calcul) et addEnQueuedCalculToList(List<Calcul> calculs)

Les lignes à modifier sont indiquées par des commentaires.

Une fois les modifications effectuées, lancer l'application comme dans l'exercice précédent et observer l'évolutions des files JMS.

La liste des calculs de la session précédente doit se recharger à chaque redémarrage de l'application (la file JMS devient un moyen de persister temporairement les calculs).

3. Application SOA : WS et JMS

Se placer au point de départ : git checkout -f SOA1

Pour démarrer les webservices, il faut se placer dans le répertoire java-ws et lancer « mvn clean install » puis « mvn cargo:run ».

Les webservices sont alors disponible ici : http://localhost:8080/webservice/services

Nous allons modifier les webservices. Ils permettront de faire abstraction entre l'interface graphique et la file JMS. Nous allons donc ré-implémenter deux méthodes :

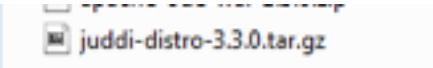
getAll et addAll pour qu'elles utilisent la file JMS plutôt que le module de persistance. Ces méthodes sont présentes dans le CalculService. Pour cela réutiliser le JMSServiceHelper.

Modifier ensuite le controller afin d'utiliser le webservice lors de l'initialisation (mise à jour du tableaux des calculs) et lorsque l'on lance un calcul (enregistrement via les webservices).

4. Exposer un webservice dans un annuaire UDDI.

1. Installation

Décompresser le dossier juddi-distro-3.3.0.tar.gz.



Copier votre JDK 7 dans juddi-distro-3.3.0\juddi-tomcat-3.3.0 avec comme nom : jdk7-win sous windows et jdk7-lin sous linux.

Ajouter le fichier setenv.sh ou setenv.bat disponible dans le github.

Exécuter la commande bin/setenv.sh ou bin\setenv.bat.

Démarrer en ligne de commande : bin\startup.bat ou bin/startup.sh.

L'annuaire de service est lancé ! S'y connecter : http://localhost:8080/juddi-gui/home.jsp.

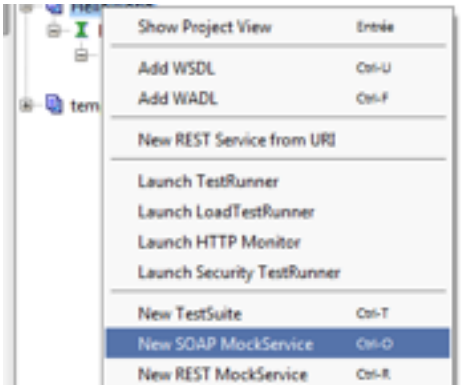
2. Ajout d'un service

Nous allons enregistrer un service généré par un mock SoapUI dans l'annuaire.

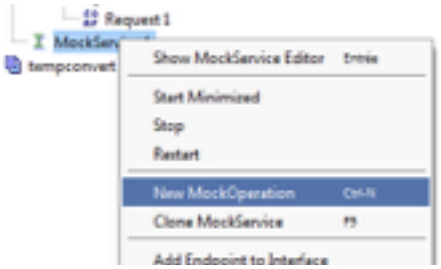
Démarrer un mock SoapUi généré à partir du WSDL HelloWorld.wsdl.

Pour cela créer un projet SoapUI en important cette WSDL :

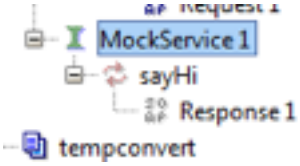
⇨ Créer un nouveau MockService



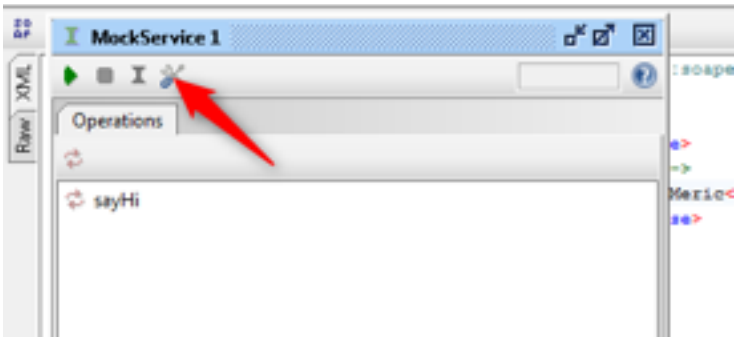
⇨ Ajouter une mock opération :

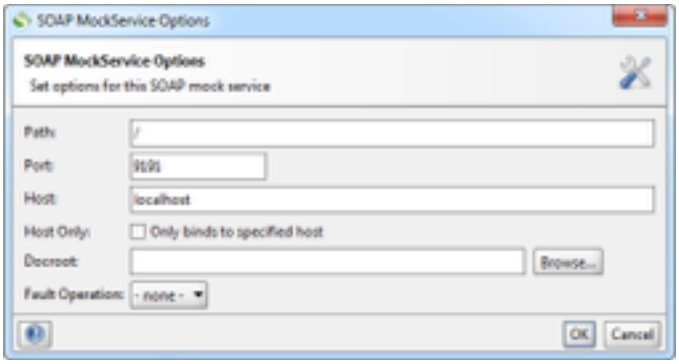


⇨ Doublie cliquer sur MockService1



⇨ Modifier les properties sur le service





⇨ Le lancer (flèche verte).

⇨ Effectuer une requête pour vérifier que le mock est UP : http://localhost:9191/web-service/services/HelloWorld

Nous allons pouvoir enregistrer ce webservice dans l'annuaire. Sa WSDL est exposée ici : http://localhost:9191/webservice/services/HelloWorld?wsdl

3. Enregistrement du service dans l'annuaire

L'interface d'admin se trouve ici : http://localhost:8080/juddi-gui/home.jsp.

Se logguer en tant qu'admin (admin - admin).

Aller dans create -> Register Services from WSDL.
Enregistrer la wsdl.
Rechercher la dans discover -> tModels.

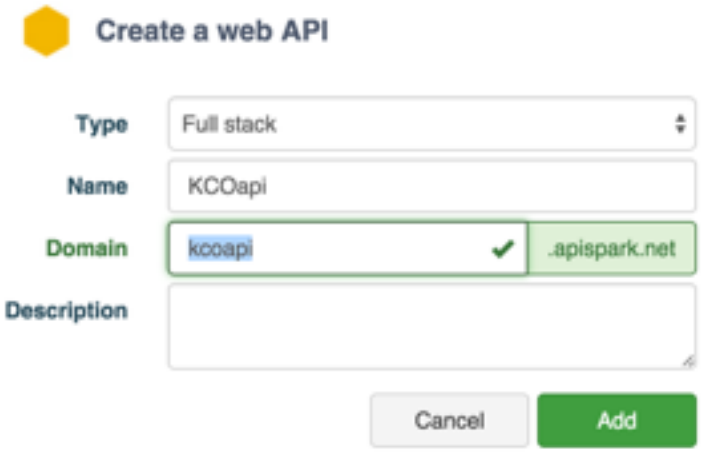
Quelles informations peut on récupérer ?
Quel est l'intérêt de stocker les services dans un annuaire ?

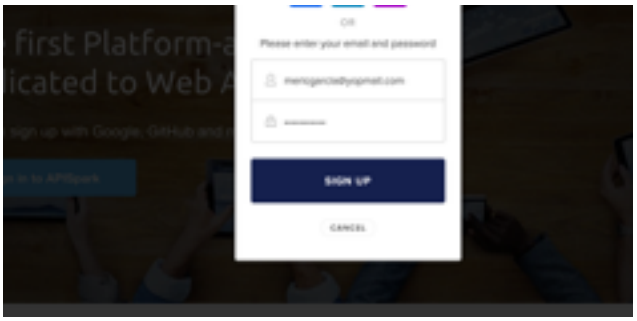
1. WOA : exposer une API Rest via un PAAS

1. Restlet et a

Aller sur le site : ht

Se créer un compte :ompte github).

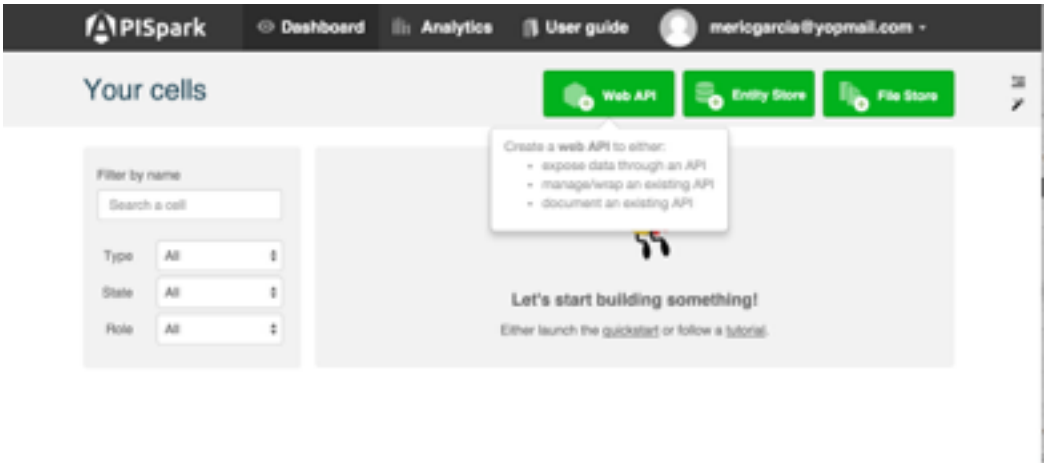




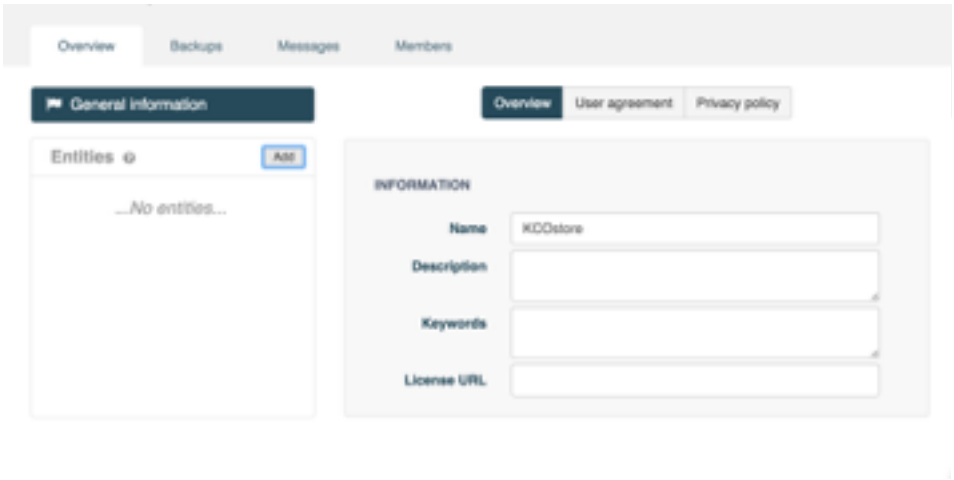
Vous aller recevoir un mail de validation. Cliquer sur le lien.

2. Creation d'une API et d'une base

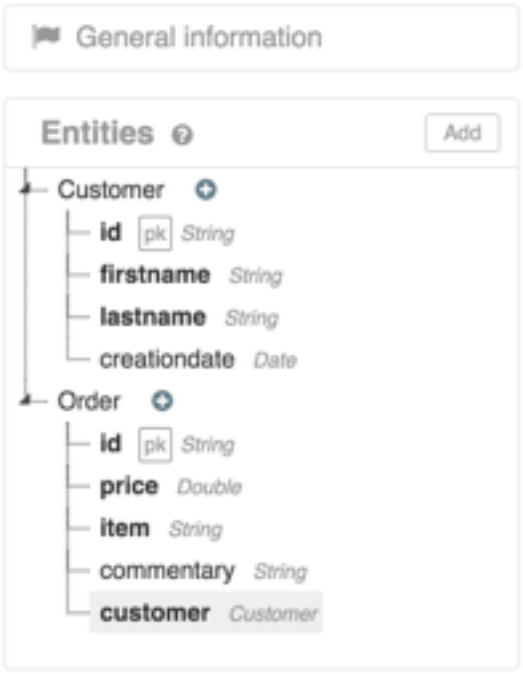
Nous allons créer une première API.



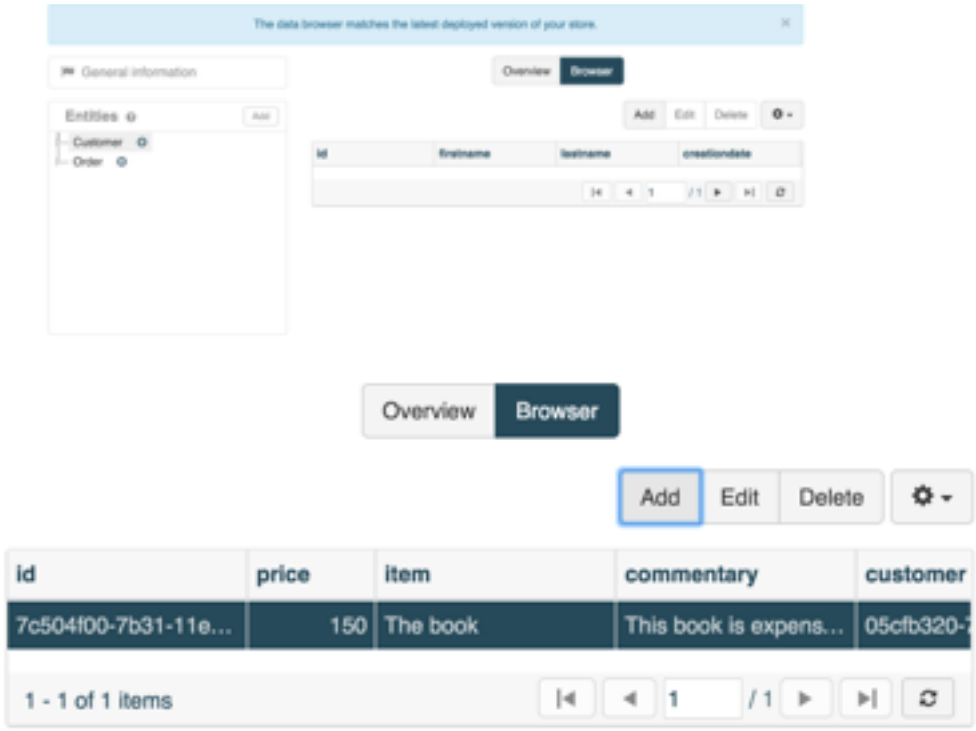
Créons également une « entity base ».



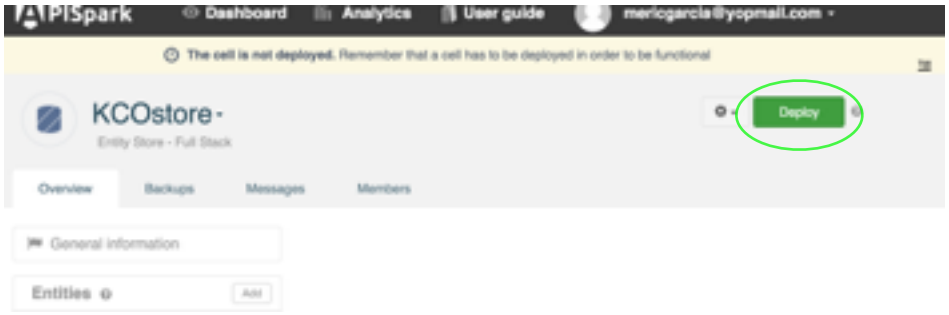
Créer le schéma suivant :



Ajoutons quelques données de test en utilisant la vue browser.

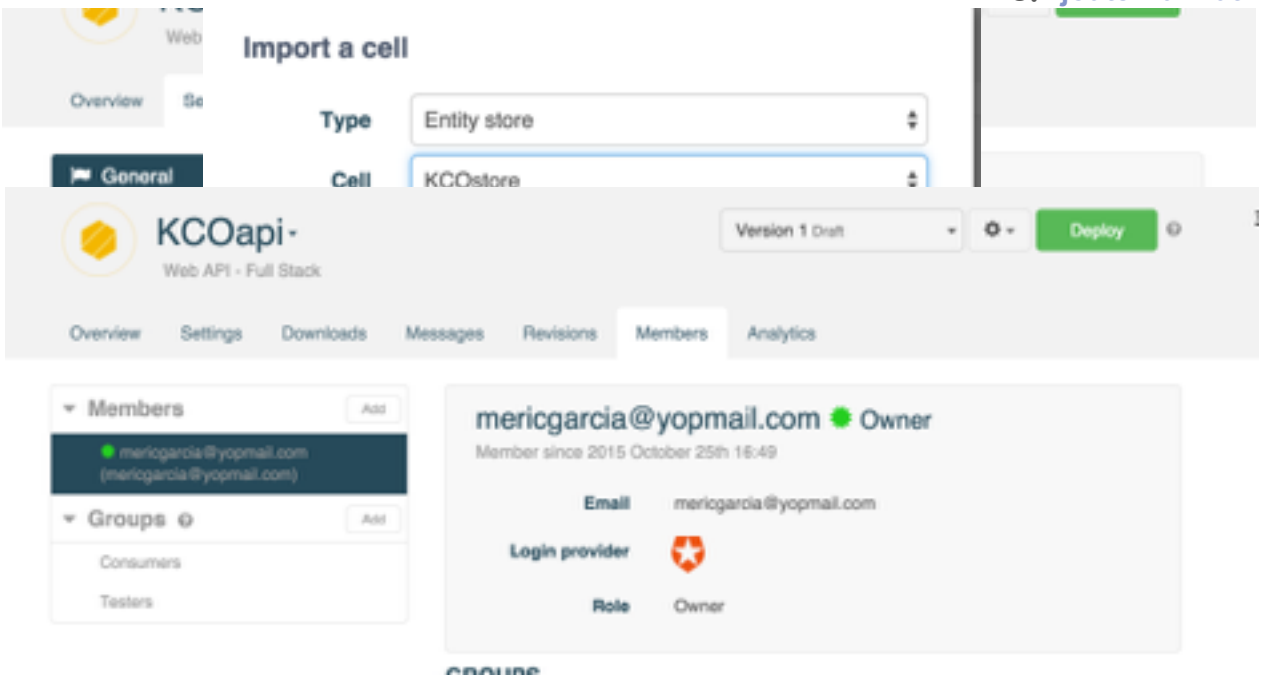


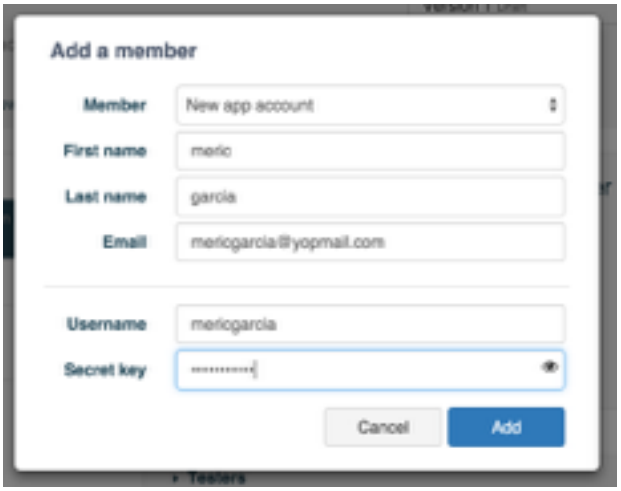
Et déployer la base :



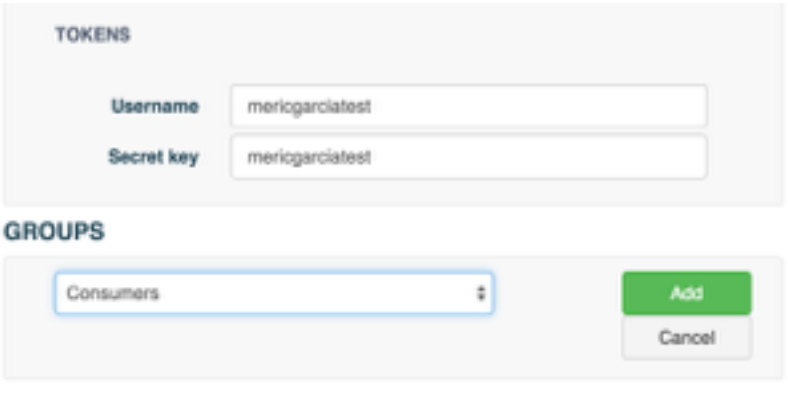
Retourner dans l'API et lier la base de données à celle ci.

3. Ajouter un uti-

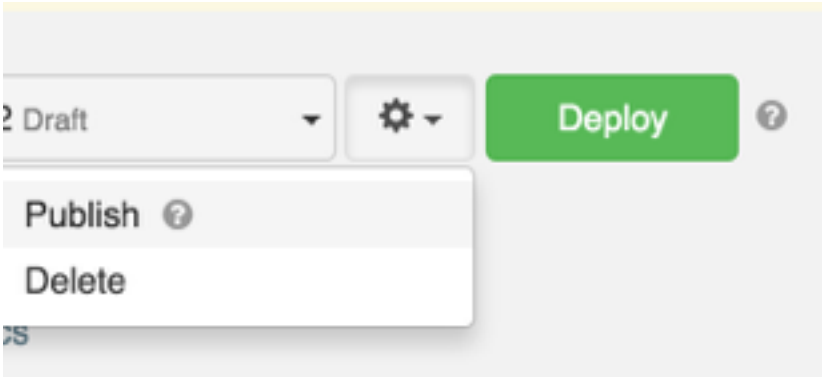




Lui affecter le rôle consommer.



Nous allons ensuite publier et déployer l'API.

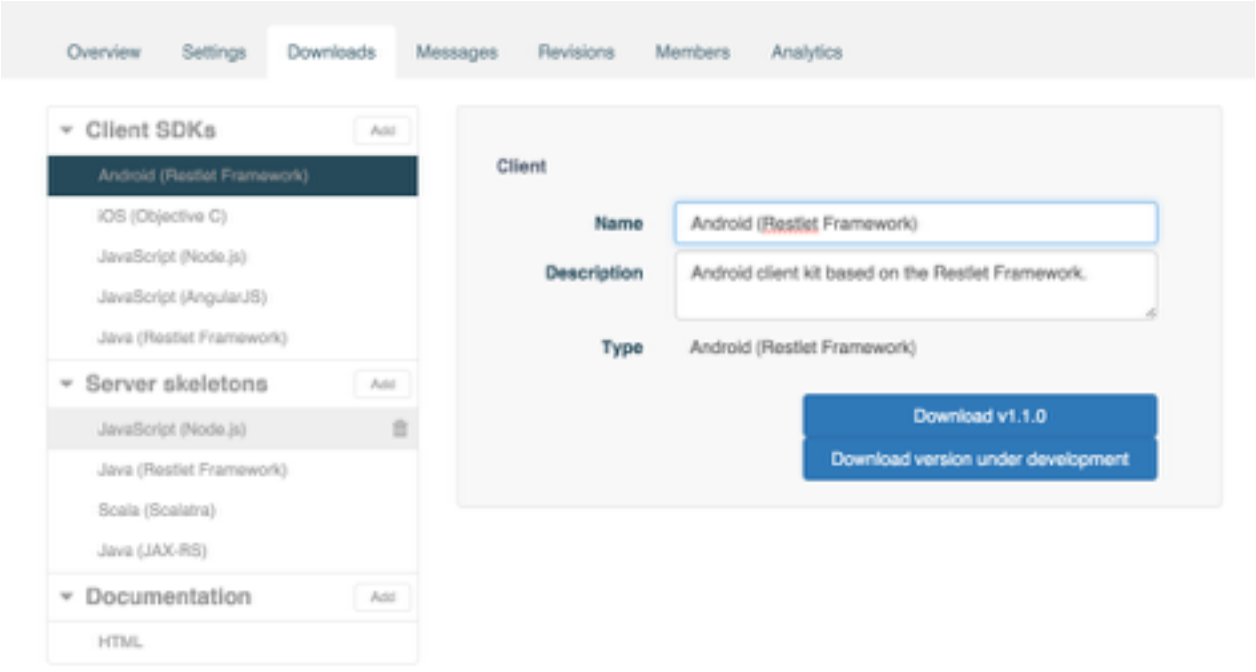


4. Test

Utiliser ce compte pour effectuer une requête via un navigateur.

5. Pour aller plus loin.

Reslet permet également de récupérer le projet générer afin de l'héberger en local.



Télécharger le serveur (JAVA ou javascript si vous avez un serveur nodeJS sur votre poste).
Pour builder et démarrer le projet récupérer :

JAVA :
mvn clean install
puis : java -jar target/….jar

NodeJS :
npm install
puis : node index

L'application est déployée en local. En regardant de plus prêt les sources générées, on s'aperçoit que seules l'API et créées et que les différentes méthodes sont à implémenter.