|  |
| --- |
| Formation MAS-ICT CAS : JAVA et Service Web |
| AngularJS et Services REST |
| Sujet 1 : REST WEB SERVICES & JS Frameworks |
|  |
| **Serge PETER** |
| **1/20/2015** |

|  |
| --- |
| Rapport détaillant la réalisation d’écrans d’administration (type CRUD) illustrant l’utilisation du framework AngularJS couplé à des services WEB REST. |

Table des matières

[1. Introduction 3](#_Toc409785358)

[1.1 Description de l’existant 3](#_Toc409785359)

[1.2 But du mini-projet 3](#_Toc409785360)

[2. Composants 4](#_Toc409785361)

[2.1 MongoDB 4](#_Toc409785362)

[2.2 JPA 4](#_Toc409785363)

[2.3 EclipseLink 4](#_Toc409785364)

[2.4 Persistence.xml 4](#_Toc409785365)

[2.5 Entity Classes 4](#_Toc409785366)

[2.6 Service REST 5](#_Toc409785367)

[2.7 JAX-RS 5](#_Toc409785368)

[2.8 Maven 5](#_Toc409785369)

[2.9 Junit 5](#_Toc409785370)

[2.10 AngularJS 5](#_Toc409785371)

[3. Architecture technique 6](#_Toc409785372)

[4. Mise en route 6](#_Toc409785373)

[4.1 Installation de MongoDB 7](#_Toc409785374)

[4.2 Installation de Java 9](#_Toc409785375)

[4.3 Installation de Maven 9](#_Toc409785376)

[4.4 Installation de Eclipse 9](#_Toc409785377)

[4.5 Installation de Tomcat 10](#_Toc409785378)

[5. Réalisation 12](#_Toc409785379)

[5.1 JPA 12](#_Toc409785380)

[5.2 Couche DAO 16](#_Toc409785381)

[5.3 Services REST 19](#_Toc409785382)

[5.4 AngularJS 24](#_Toc409785383)

[5.5 Bootstrapping 24](#_Toc409785384)

[5.6 Template AngularJS 25](#_Toc409785385)

[5.7 Scope et two-way data binding. 27](#_Toc409785386)

[5.8 Injection du service $http 27](#_Toc409785387)

[5.9 Premier résultat 28](#_Toc409785388)

[5.10 Aide à la saisie, contrôles 28](#_Toc409785389)

[5.11 Listes de valeurs 29](#_Toc409785390)

[5.12 Champs requis 29](#_Toc409785391)

[5.13 Sous-mission sous contrôle 30](#_Toc409785392)

[5.1 Formulaire avec validation (v2) 31](#_Toc409785393)

[6. Alternatives 35](#_Toc409785394)

[6.1 Spring DATA MongoDB & REST 35](#_Toc409785395)

[6.2 Configuration : 35](#_Toc409785396)

[6.3 Entités : 36](#_Toc409785397)

[6.4 CrudRepository et PageAndSortingRespository 36](#_Toc409785398)

[6.5 REST 36](#_Toc409785399)

[7. Conclusion 37](#_Toc409785400)

[8. Références 38](#_Toc409785401)

# Introduction

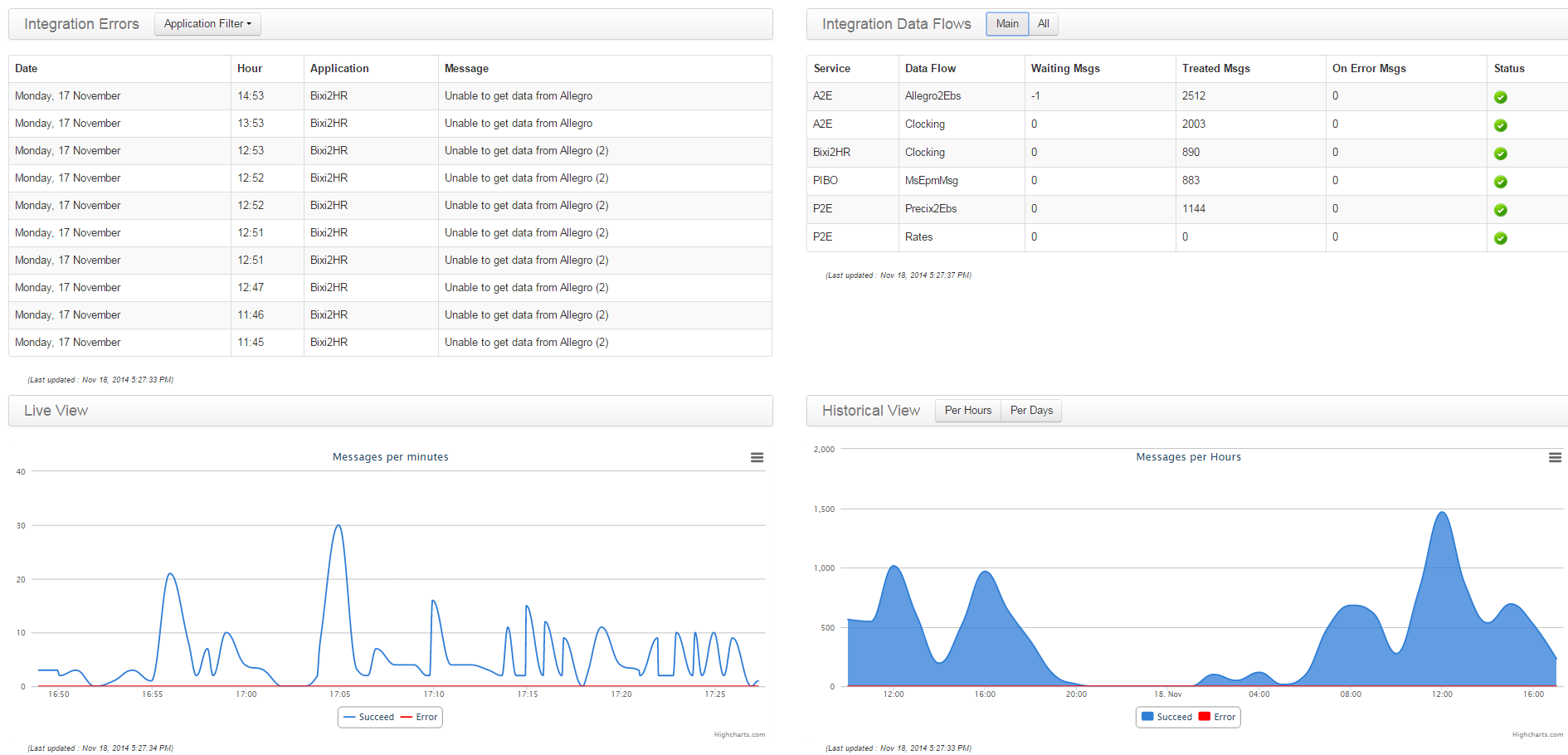
Ce document a pour but de présenter le travail que j’ai réalisé dans le cadre d’un mini projet concertant le cours [MAS-ICT CAS : JAVA et Service Web](http://www.mas-ict.ch/programme/cas-siw/module-wsj.aspx).

Le but de tout ceci est purement didactique mais ceci correspond à un besoin réel dans mon entreprise qui est la construction d’une application de monitoring des flux de données (interfaces) entre application. Ces flux sont monitorés au niveau à la quantité de messages transmis et de la qualité ; message en erreurs.

## Description de l’existant

ITM est une application de monitoring permettant de faire remonter des alertes et de l’information en vue de la présenter sur un écran de monitoring (développée dans mon équipe). Le stack technologique est : MongoDB + Spring (Data, REST, …) et AngularJS avec surcouche graphique Twitter Bootstrap. Le but est de remonter des alertes et de suivre l’évolution de plusieurs interfaces et queues.

Voici à quoi cela ressemble :



## But du mini-projet

Le but est de créer une application d’administration des données liées à cette application.

Afin de rester en phase avec le cours, nous travaillerons avec du JEE pure du côté service de données REST et en AngularJS du côté présentation.

Les écrans d’administrations comprendront deux entités principales, dataflow et dataqueue, et éventuellement d’autres entités existantes.

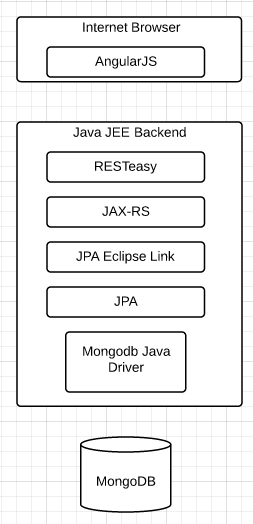
DataFlow : Ajout, Liste, Modification, Suppression.

DataQueue : Ajout, Liste, Modification, Suppression.

Un autre but sera d’expliquer les avantages / inconvénients de la solution JEE + AngularJS par rapport à Spring + AngularJS.

# Architecture technique

Le but de ce travail est de rester le plus proche possible de la norme JEE, sans utiliser de composant Spring, ceci afin de pouvoir ensuite comparer les 2 types de solutions. Dans notre cas, Eclipse Link, implémentation de la norme JPA, permet de travailler sur une base de données MongoDB de manière similaire à tout autre SGBD.



# Composants

Voici une brève présentation des composants utilisés dans le cadre de ce travail.

## MongoDB

MongoDB est une base de données relationnelle dite NoSQL. Les données sont stoquées sous forme de document JSON, facilement utilisable dans du code JavaScript. MongoDB est gratuite et open-source.

Nous utilisons MongoDB dans un cas précis de collecte de données de monitoring ; données qui n’ont pas pour but d’être conservée comme dans une application de gestion.

## JPA

Java Persistance API est une spécification Java permettant de faciliter la gestion de la persistance des données.

JPA 2.0 et 2.1 corresponde à des JSR (Java Specification Request) 317 et JSR 338.

Bien que normalement JPA s’adresse aux gestionnaires de base de données relationnelles, nous l’utiliserons de la même manière pour gérer nos collections NoSQL. JPA étant la spécification mais nous devons choisir une implémentation.

### EclipseLink

EclipseLink est l’implémentation référence de JPA, elle est fort heureusement compatible avec MongoDB (et Oracle NoSQL). Nous l’utiliserons donc comme librairie JPA dans notre cas pour cette principale raison.

Il en existe d’autres, les principales étant [DataNucleus](http://datanucleus.org/), [Hibernate](http://hibernate.org/) et [EclipseLink](http://eclipse.org/eclipselink/).

### Persistence.xml

Depuis quelques temps, les choses se sont grandement simplifiées au niveau des fichiers de configuration XML. Pour JPA (et EJB3), il nous reste un seul fichier dans lequel nous devons spécifier les informations de connexions à la base de données.

Ce fichier liste aussi les classes entité (ou Entity), qui sont l’image Java de la table (ou la collection pour le NoSQL).

### Entity Classes

Là aussi les choses ont évolués depuis l’apparition des annotations et des nouvelles versions de JEE. Le mapping entre les représentations SQL et Java est fait de manière grandement simplifiée par JPA.

Point important à noter, JPA définit pour des collections NoSQL des annotations spécifiques ; par exemple l’annotation NoSQL :



## Service REST

### JAX-RS

JAX-RS est une API Java faite pour les services REST. Un nombre important d’implémentations existe ; les plus connues étant Apache CXF, Jersey, RESTeasy de JBoss…

Dans notre cas nous utiliserons l’implémentation [RESTeasy](http://resteasy.jboss.org/). Choix arbitraire. Mais le but pour moi était de ne pas prendre quelque chose de connu (CXF).

## Maven

Je ne vais pas présenter Maven ici, mais juste dire que je l’utiliserais pour faciliter la configuration et l’obtention des librairies liée au projet.

## Junit

Des tests unitaires sont écrits afin de tester le fonctionnement des couches serveurs de l’application.

## AngularJS

AngularJS est un Framework JavaScript créé par Google. Une des choses les plus visible que nous allons faire ici avec AngularJS est le découplage entre l’application back-end écrite en Java et l’application user frontale, qui elle sera faite en pure JavaScript.

AngularJS facilite grandement le binding entre le model (via service REST) et la vue.

# Mise en route

Les outils nécessaires pour effectuer ces développements sont :

|  |  |  |
| --- | --- | --- |
| Composants | Version | Location |
| MongoDB | 2.6 et plus | <http://www.mongodb.org/downloads?_ga=1.174529378.1732284771.1420536613> |
| Java JDK | 1.7 | <http://www.oracle.com/technetwork/java/javase/downloads/index.html> |
| Maven | 3.0 et plus | <http://maven.apache.org/download.cgi> |
| Tomcat | 8 | <http://tomcat.apache.org/download-80.cgi> |
| Eclipse | Luna | <https://www.eclipse.org/downloads/> |

Du fait que je travaille sur Windows 7 version 64-bit, les informations d’installation seront faite sur cette plateforme. Nous devons pour ce faire installer le serveur de base de données MongoDB et créer une base de données de travail (itmee). Ensuite nous devrons avoir à disposition les outils de développement (Eclispe, Java, Maven, Tomcat 8).

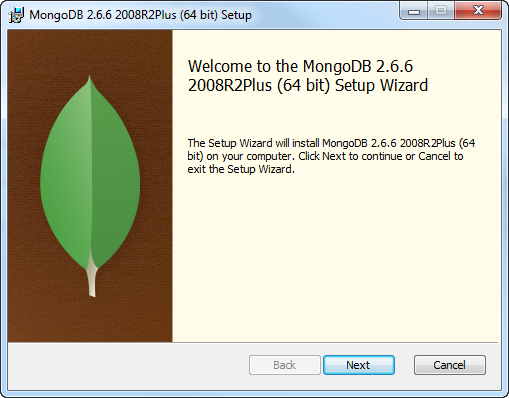
## Installation de MongoDB

MongoDB est disponible ici :

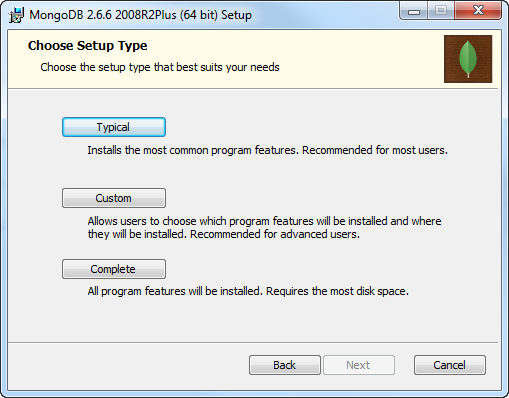
<http://www.mongodb.org/downloads?_ga=1.93256285.1732284771.1420536613>

Pour installer MongoDB sur Windows 7

Lancer l’installeur Windows mongodb\*.msi correspondant :



Choisissez une installation Typique.



Et procédez à l’installation.

* Créez un répertoire de stockage des données, par défaut c’est : c:\data\db,
* Copiez C:\Program Files\MongoDB 2.6 Standard\bin sur c:\mongodb,
* Créez un répertoire c:\mongodb\log,
* Créez un fichier de configuration :

echo logpath=C:\mongodb\log\mongo.log > "C:\mongodb\mongod.cfg"

* Créez un service Windows comme ceci :

"C:\mongodb\mongod.exe" --config "C:\mongodb\mongod.cfg" –install

* Vérifiez que le service fonctionne :

Net start MongoDB

* Faites un test de connexion :

C:\mongodb\mongo.exe

Dans notre cas il faut aussi créer une base de données et un utilisateur ; par soucis de simplicité nous donnons les droits « root » à cet utilisateur :

use itmee

Db.createUser(

{

user: "itmee",

pwd: "itmeeitmee",

roles: [ "root" ]

}

)

Un client graphique gratuit existe : <http://robomongo.org/> qui permet de faire les opérations d’administration et les requêtes plus facilement.

## Installation de Java

Java JDK est disponible ici :

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Une fois le logiciel téléchargé faite l’installation. Ceci ne pose pas de difficulté.

A noter que nous avons besoin du JDK et que le JRE ne suffit pas.

## Installation de Maven

Maven est disponible ici :

<http://maven.apache.org/download.cgi>

Une fois le package [apache-maven-3.\*.\*-bin.zip](http://mirror.switch.ch/mirror/apache/dist/maven/maven-3/3.2.5/binaries/apache-maven-3.2.5-bin.zip) downloadé, décompressez dans un répertoire local (par exemple c:\apps).

Vous devez ensuite :

1. Créer une variable d’environnement M2\_HOME correspondant au répertoire d’installation de Maven.
2. Créer une variable M2 correspondant a %M2\_HOME%\bin
3. Ajouter M2 au PATH.

Il faut aussi créer un fichier settings, contenant les chemins locaux de repositories, si existant ; des informations d’utilisation de proxy ou autre.

Ce fichier doit être placé ici. <HOME>/.m2/settings.xml.

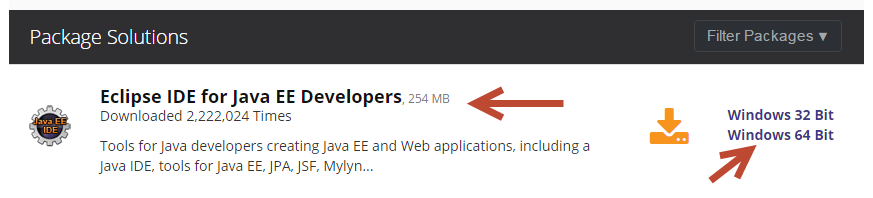
Plus d’info sont disponible ici : <http://maven.apache.org/settings.html>.

## Installation de Eclipse

Eclipse Luna est disponible ici :

<https://www.eclipse.org/downloads/>

Il faut ensuite faire attention de prendre la version EE :



Une fois le package téléchargé, décompresser et exécuter le fichier eclipse.exe.

## Installation de Tomcat

Tomcat 8 est disponible ici :

<http://tomcat.apache.org/download-80.cgi>

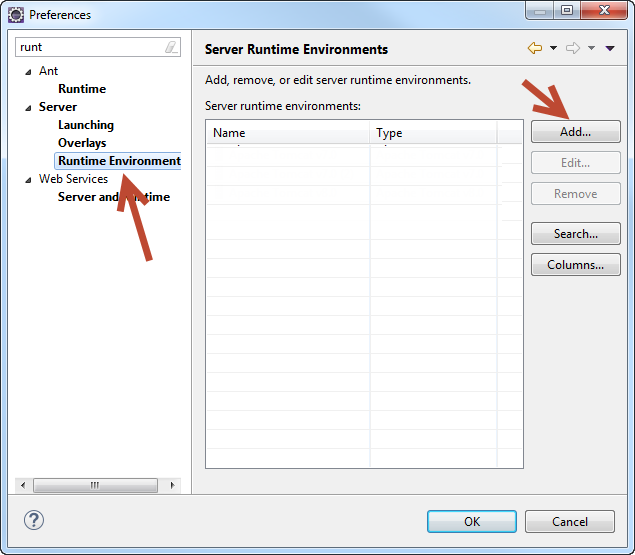
Une fois le package téléchargé, il faut le décompresser en local (par exemple sur c:\apps\tomcat8).

Nous utiliserons Eclipse pour les opérations de start/stop ainsi que le déploiement de l’application.

Pour ce faire il faut créer un serveur Tomcat8 dans Eclipse comme ceci :

**Ajouter un runtime Tomcat 8** :

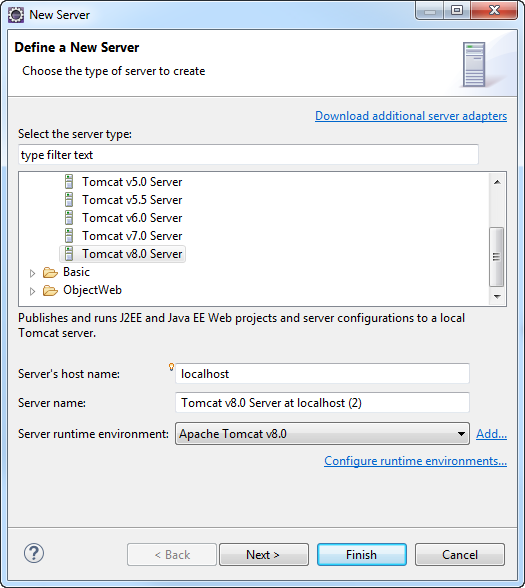
* Dans l’onglet Préférences de Eclipse, rechercher : runtime ; clicker sur Add…



* Définissez le répertoire Tomcat de travail, ainsi qu’un JRE, qui doit être préalablement définit dans Eclipse, ou le runtime par défaut.

**Ajouter un serveur Tomcat 8 :**

* Dans l’onglet « Servers », visible dans la perspective JEE, créez un nouveau serveur basé sur le runtime Tomcat 8 :



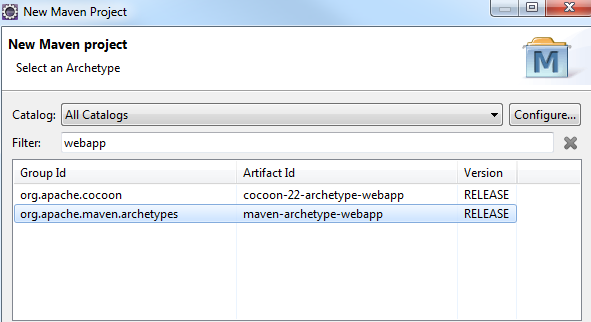
* Vous pourrez plus tard déployer l’application sur ce server.

# Réalisation

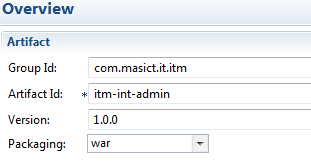
Je vais décrire ici la création du programme, depuis les couches basses technique Maven, JPA, DAO ; les services REST et la vue AngularJS.

## JPA

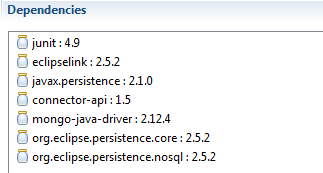
Nous créons pour commencer un projet en utilisant un archétype standard Maven :



Nous définirons les informations comme ceci :



Nous ajoutons les dépendances minimales pour pouvoir faire du JPA :



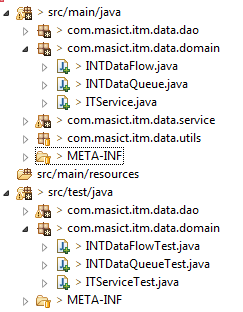
**Persistence.xml**

Une fois les librairies JPA à disposition, nous devons créer le fichier persistence.xml :

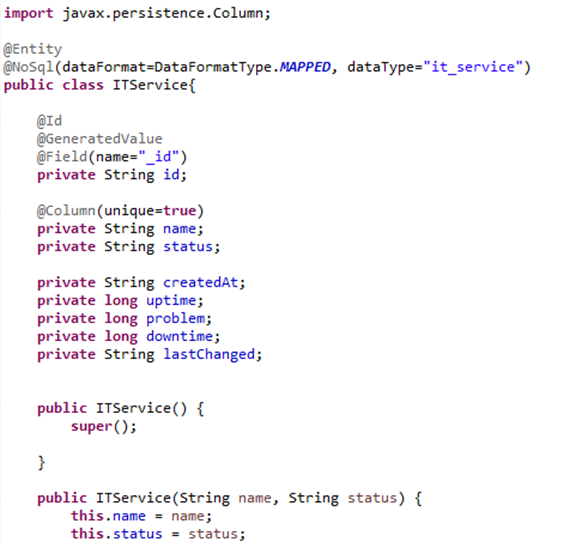


**Entités :**

Et finalement les premières classe entités :



Dont voici un exemple :





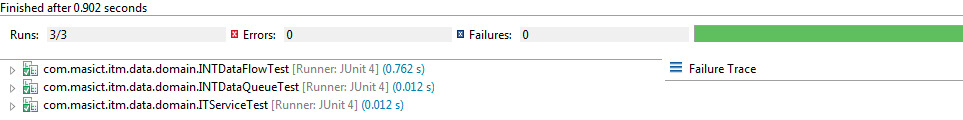
**A noter :**

* @NoSql pour des documents MongoDB.
* @Id pour la clé primaire (obligatoire)
* Le type temporel, ici en TIMESTAMP.
* Le nom de la collection MongoDB est en fait le champ : dataType.

**Classes de teste**



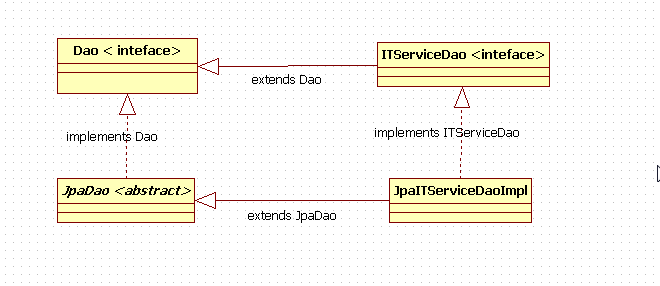
**Exécution du test :**



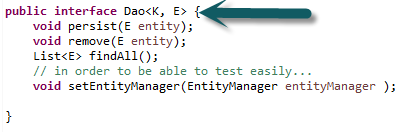
## Couche DAO

Je cherchais un exemple d’implémentation de la couche DAO en minimisant le code technique et mécanique ; qui n’apporte pas grand-chose du point de vue business mais qui parfois coute cher. Une pratique de design se fait en utilisant les types génériques. J’ai fait une version qui s’inspire de quelques idées trouvée sur internet ; <http://java.dzone.com/articles/jpa-implementation-patterns>.

On utilise la mécanique des interfaces et des types génériques pour spécialiser les Dao au niveau des entités :



1) Dao : Type générique :

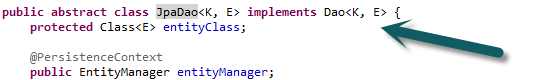


2) ITServiceDao : Type spécifié dans l'interface :

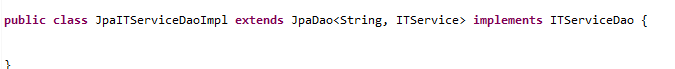
C:\Users\speter\AppData\Local\Temp\enhtmlclip\Image(7).png

3) Implémentation abstraite de Dao pour JPA :

Implémentation de Dao pour Jpa, on reste ici avec les types génériques (même dans l'implémentation).



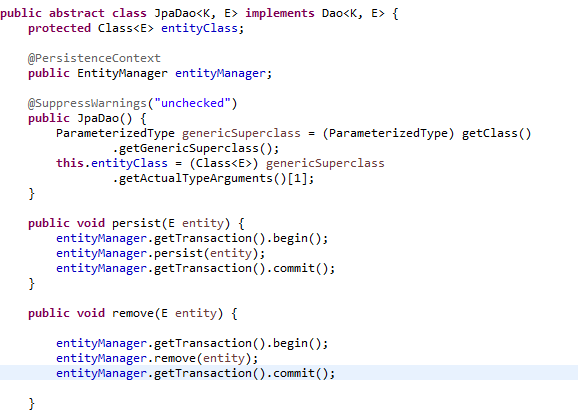
4) Implémentation finale :



Ce type d'implémentation permet de minimiser le code technique pour toutes les DAOs de l'application.

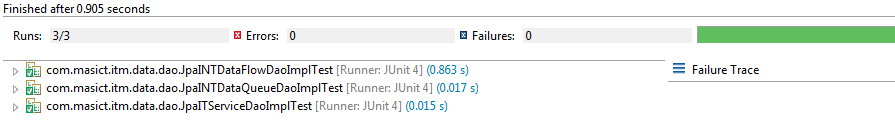
Un point à noter est l'utilisation de type générique dans l'implémentation :

* K pour la clé primaire
* E pour l'entity (NoSQL)



Au niveau de l'utilisation dans la classe de test :



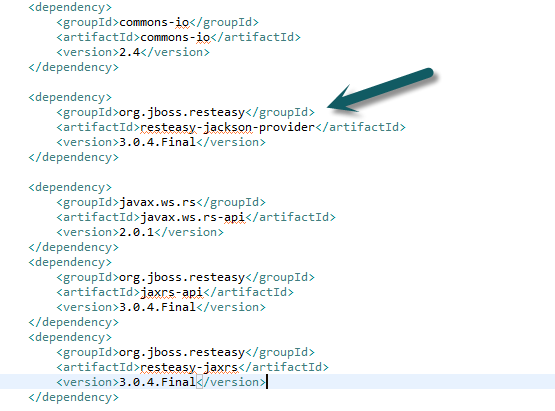


Tout est prêt pour la création des services REST.

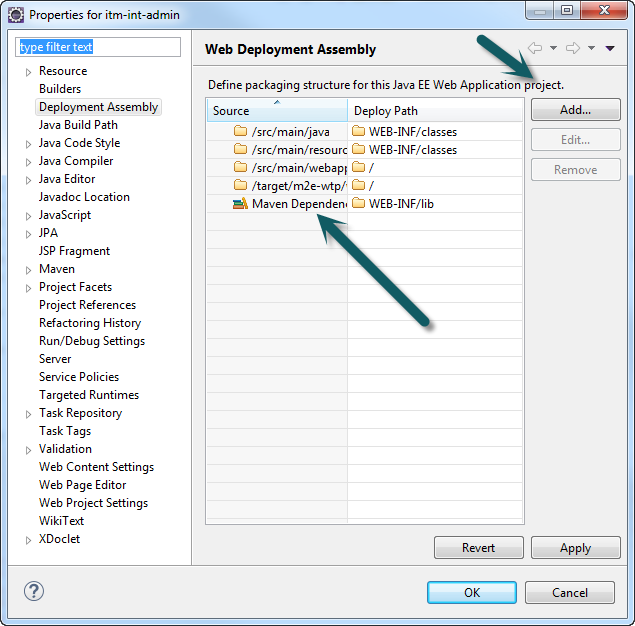
## Services REST

Dans la couche Service nous allons maintenant ajouter la fonctionnalité de services REST.

Dans un premier temps il faut ajouter les librairies manquantes dans le pom.xml :



Certaine librairies dépendantes doivent être déployée sur notre runtime Tomcat 8 pour que cela puisse fonctionner. Pour ce faire il faut ajouter les dépendances Maven dans le déploiement. Dans les propriétés du projet :



Note :

Pour une exécution directe depuis mvn.exe (en mode de commande ligne) , il faut faire attention d’utile :

mvn clean tomcat7:run-war

Afin d’utiliser toutes les dépendances Maven.

**Configuration du web.xml**

Pour la librairie RESTeasy il faut ajouter un peu de configuration dans le web.xml.



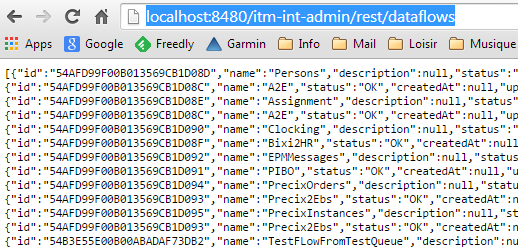
On peut noter que les fichiers Java seront scannés pour retrouver les annotations propres aux services REST et aussi placée sur l’url relative *REST*.

**Implémentation des services :**

|  |  |  |  |
| --- | --- | --- | --- |
| CRUD verb | http verb | Java annotations | Description |
| Read (list) | GET | @GET  @Path("/") | List des dataflows, à noter l’annotation parente :  @GET  @Path("/") |
| Read | GET | @GET  @Path("/{id}") | Obtenir un dataflow par son id |
| Create | PUT | @POST  @Path("/") | Créer un nouveau dataflow |
| Update | PUT | Idem | La méthode create et update est la même fonction |
| Delete | DELETE | @DELETE  @Path("/{id}") | Effacer un dataflow par son id |



Ce qui fait que si on test ces services :



A noter que : J’ai ajouté une fonction de création du jeu de données pour faciliter l’utilisation de l’application.

## AngularJS

Afin d’illustrer le Framework AngularJS et l’interaction de celui-ci avec la consommation de donnée sous forme de service reste je vais progresser par pallier. Vous trouverez en références des livres et tutoriaux sur le sujet qui illustre cette façon de faire.

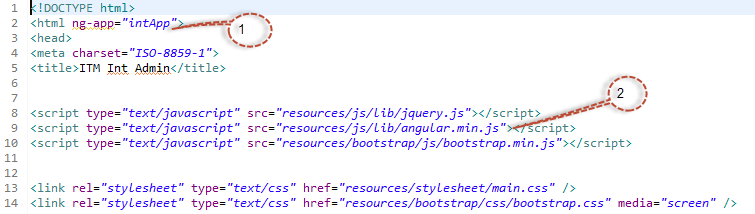
Nous allons procéder en deux étapes :

1. Mise en place basique d’AngularJS avec les éléments de base (bootstraping, templating, scope). Cette version sera capable de travailler avec les services REST mis en place précédemment.
2. Mise en place d’une version plus avancée des formulaires.

## Bootstrapping

Le bootrapping consiste à charger le Framework AngularJS en « même temps » que la page html. Ceci permettra d’afficher les éléments de la page quand elle est complète et de faire fonctionner le Framework correctement.

Première étape ; nous allons tout d’abord initialiser notre application :



1. Nous surchargeons la balise <html/> en ajoutant une directive AngularJS,
2. Nous importons les librairies angularjs.

Vous noterez aussi qu’on import jquery.js (bien qu’une version minimaliste soit disponible dans angularjs ainsi que les librairies et feuille de style twitter Bootstrap afin de profiter de la mise en forme issue de ces librairies.

Pour que cela fonctionne il faut bien sûr créer un script JavaScript qui sera source dans la page html.



## Template AngularJS

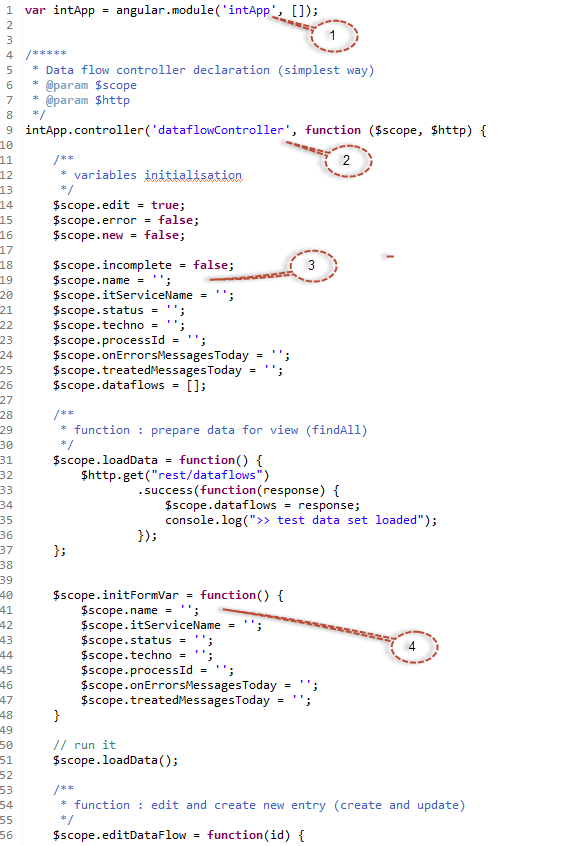
Le Template AngularJS n’est rien d’autre que la vue HTML augmentée avec les directives angularjs permettant d’afficher des données model et d’interagir avec l’utilisateur.

Dans notre vue simplifiée nous avons plusieurs composants notables :



1. L’assignation du controller AngularJS au body avec la notation gn-controller,
2. Une boucle permettant d’afficher les éléments de la liste datalows avec la notation ng-repeat,
3. L’affichage des élément du tableau avec les expressions {{ }} ,
4. Des interactions avec des fonctions via les nq-click.

Du côté script angularjs nous retrouvons les éléments mis en relation :



1. Initialisation de l’application intApp,
2. Création du contrôleur utilisé dans notre page html, et injection du service $http,
3. Initialisation de variable « scope », permettant de transmettre les données du model à la vue,
4. Déclaration des fonctions, par exemple ici une fonction d’initialisation du formulaire.

## Scope et two-way data binding.

Le scope est un élément très important pour permettre la transmission des données entre les composants du Framework.

AngularJS permet de transmettre les informations du model vers la couche de présentation dans les deux sens sans faire de rafraichissement de page.

## Injection du service $http

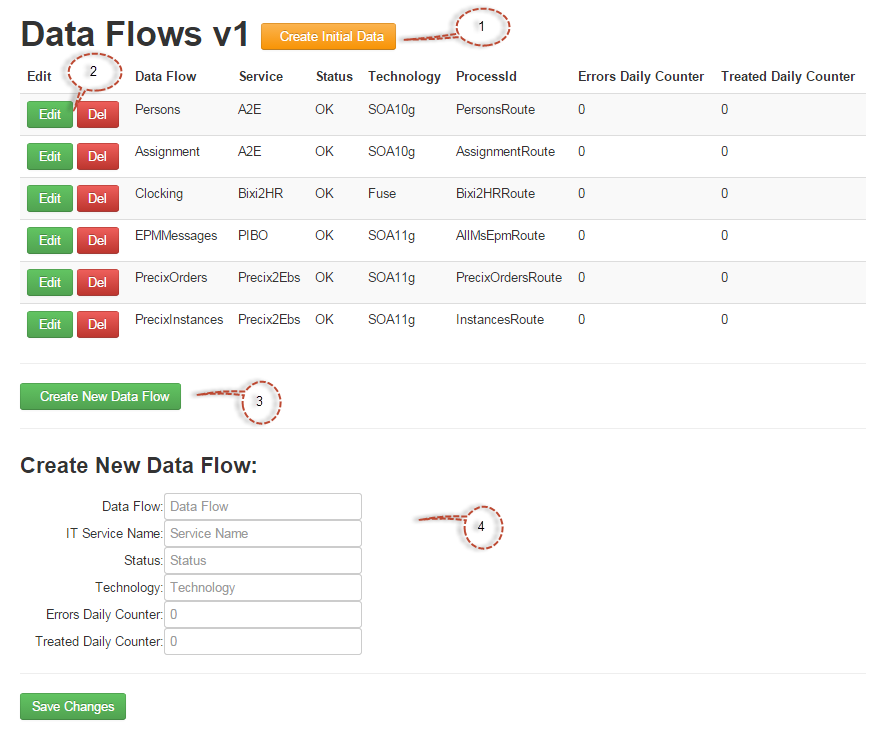
Nous utilisons le service $http pour faire les différents appels aux services REST. Ce service encapsule les appels AJAX (XMLHttpRequest) au backend.

Nous pouvons mapper les services REST disponible comme ceci :

|  |  |  |  |
| --- | --- | --- | --- |
| CRUD verb | http verb | AngularJS function | Description |
| Read (list) | GET | $http.get("rest/dataflows") | List des dataflows |
| Read | GET | $http.get("rest/dataflows/" + id) | Ubtenir un dataflow par son id |
| Create | PUT | $http.put("rest/dataflows/", $scope.dataflow ) | Créer un nouveau dataflow |
| Update | PUT | $http.put("rest/dataflows/", $scope.dataflow ) | Update un dataflow existant |
| Delete | DELETE | $http.delete("rest/dataflows/" + id) | Effacer un dataflow par son id |

## Premier résultat (v1)

Voici donc une capture d’écran de la vue concernant cette première itération ; veuillez ouvrir le lien Data Flow v1 : http://<localhost:port>/itm-int-admin/dataflows-v1.html



1. Création du set de donnée initial,
2. Edition ou Suppression,
3. Création,
4. Formulaire d’édition.

## Aide à la saisie, contrôles

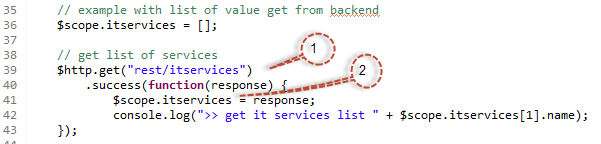
Nous allons maintenant sécuriser la vue en y ajouter des contrôles et liste de valeurs. Ceci correspondra à la version v2 de notre vue d’administration.

## Listes de valeurs

Il est possible de créer des listes de valeurs statiques ou dynamiques et de les affichées dans des éléments <select/> dans la vue :

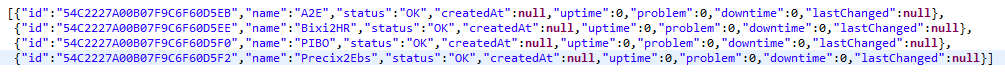
Voici un exemple de liste de valeurs dynamiques :

**JavaScript :**

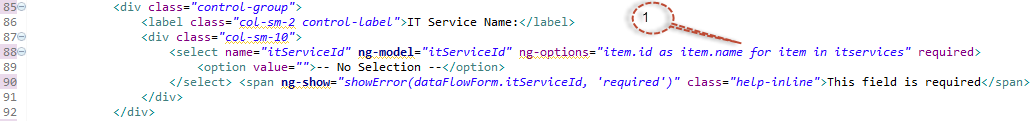


1. Appel du service REST
2. Assignation dans une variable $scope

**Données fournies en JSON :**



**Page de présentation HTML :**



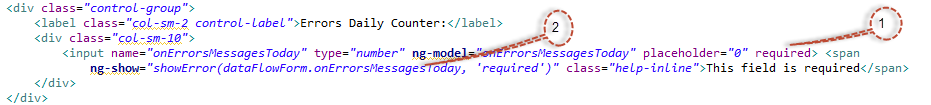
1. Utilisation dans la vue avec la directive ng-options

## Champs requis

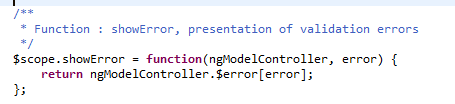
Il est facile de rendre un champ obligatoire (required).

Voici un exemple :

**HTML :**



**JavaScript :**



La directive ng-show permet de contrôler l’affichage du message d’erreur en utilisant la fonction showError().

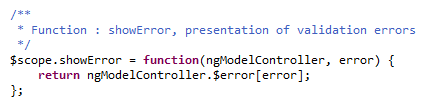
## Sous-mission sous contrôle

Il est possible d’empêcher l’exécution de la soumission de la page en utilisant une directive ng-disabled comme ceci :

**HTML :**

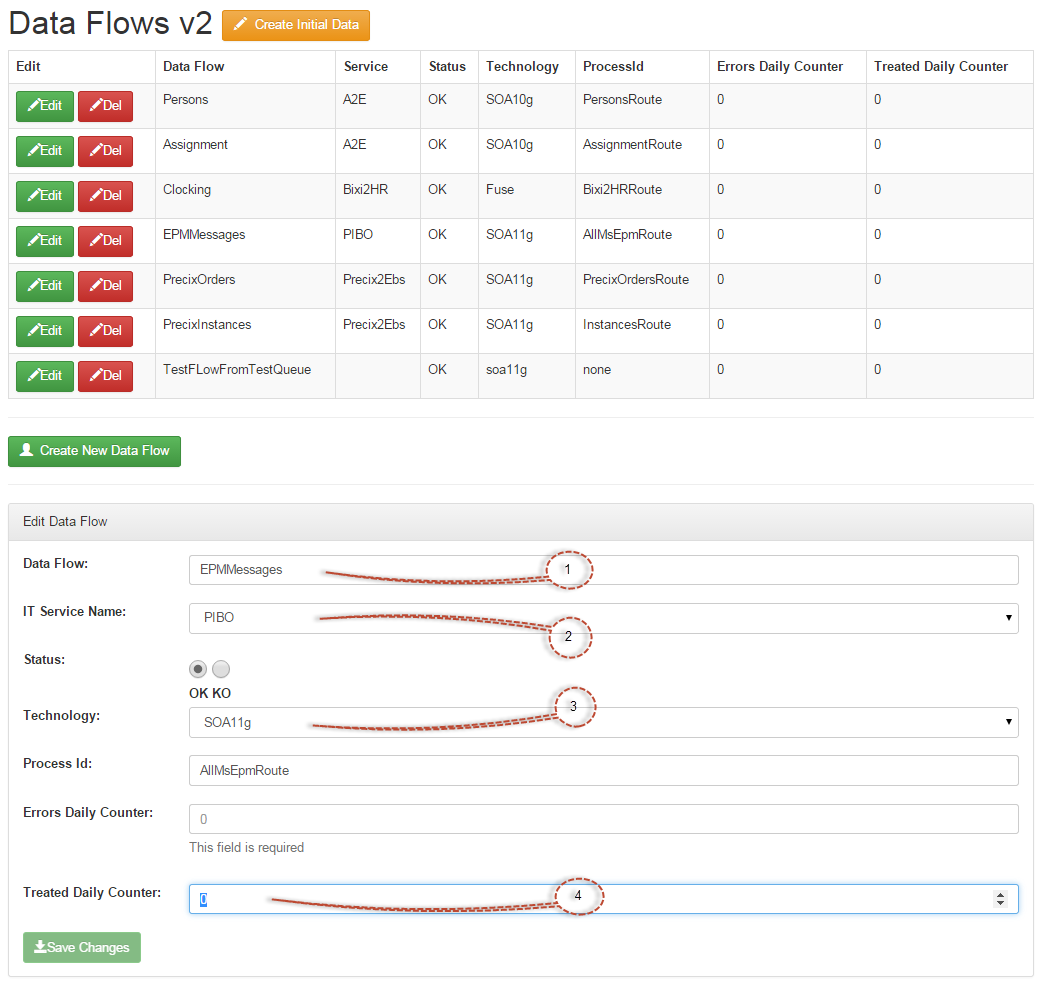


**JavaScript :**



## Formulaire avec validation (v2)

Voici donc une capture d’écran du formulaire d’édition amélioré avec quelques contrôles et aide à la saisie : http://<localhost:port>/itm-int-admin/dataflows-v2.html



1. Champs textuel non null,
2. Liste de valeurs dynamique,
3. Liste de valeurs statique,
4. Champs de type numérique non null,
5. Bouton Save de submit contrôlé (ici il est « disabled » car un champ est null.

D’autres possibilités existent pour valider et ainsi sécuriser un formulaire sous AngularJS ; comme par exemple la validation d’un email, les types de données attendues, et autres limites sur les champs textes, que l’on peut retrouver dans les diverses documentations.

## Utilisation de ngResource

AngularJS fournit un mécanisme permettant d’injecter des services dans l’application. Il est possible d’utiliser l’extension ngResource pour rendre plus direct le mapping des services REST entre le backend et le frontend.

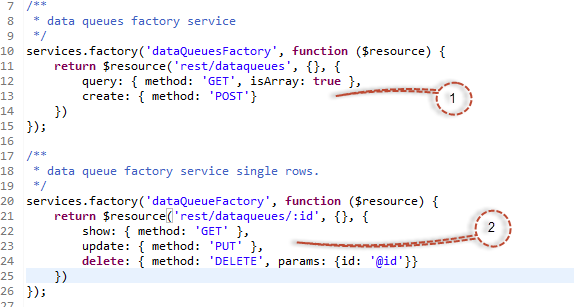
J’ai fait comme exemple final un troisième écran, cette fois sur l’entité DataQueue afin d’illustrer l’utilisation de ce service.

Voici comment j’ai procédé :

Importation de et chargement de la librairies ngResource :

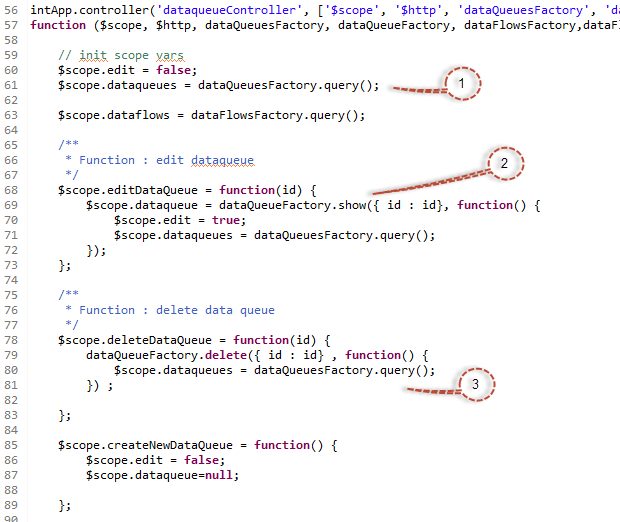


Ensuite il faut créer deux factories pour mapper les methodes query, create, show, update, delete avec leur correspondant REST



1. Méthodes sans paramètres get dans l’URL,
2. Méthodes avec paramètres.

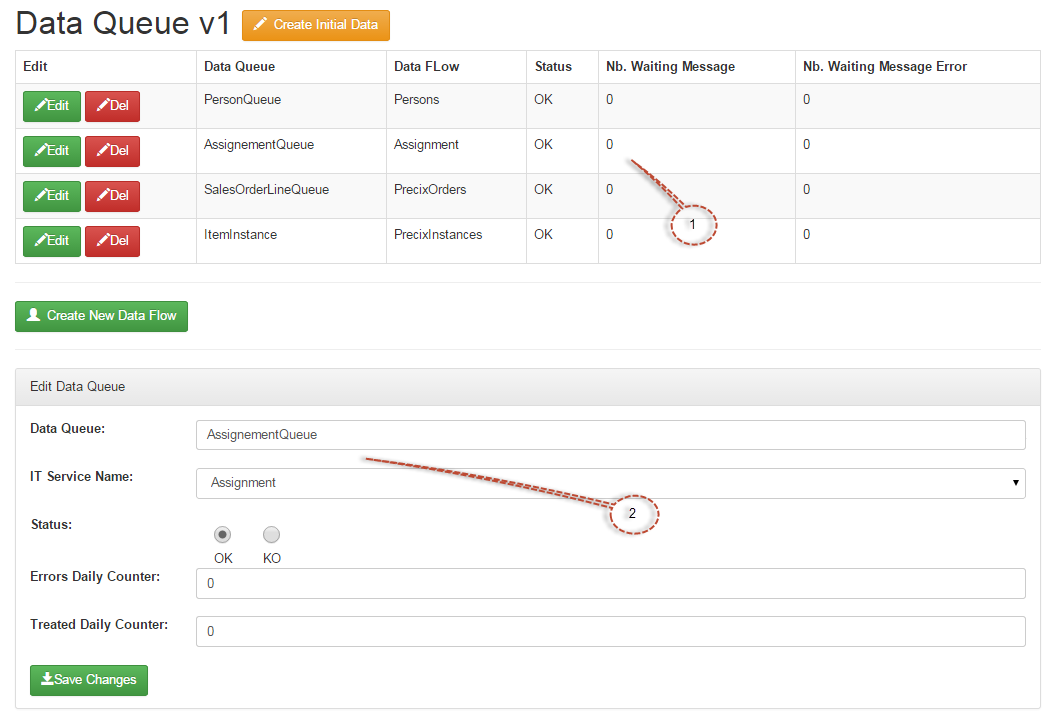
Utilisation dans le code du contrôler :



1. Appel des méthodes de la fabrique : query (pour list),
2. Appel de la méthode show avec le paramètre id,
3. Appel de la méthode delete avec le paramètre id, on remarque la mécanique JavaScript permettant de faire le rafraîchissement de l’écran de manière synchone.

Tous les éléments du CRUD peuvent donc être simplement mappés avec les services REST.

## Formulaire utilisation de ngResource (v1)



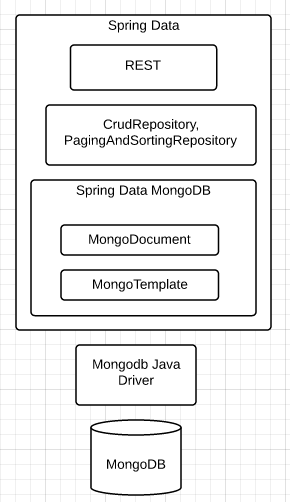
1. La liste des enregistrements avec les outils de Edit / Delete,
2. Le formulaire.

# Alternatives

Dans le cahier des charges de ce projet je me proposais de comparer ce que proposait JEE dans les standards concernant JPA et les services reste et le Framework Spring, notamment Spring DATA (MongoDB), Spring DATA REST. A noter aussi l’existence de JPA-RS dans le monde JEE qui permet de faire des choses similaires.

## Spring DATA MongoDB & REST

Spring Data et un Framework bien connu qui permet de travailler rapidement sur les sujets qui nous intéresse ici, à savoir la constitution de service REST monté sur une base de données MongoDB.



Dans ce stack technologique certains aspects sont très similaires :

### Configuration

La configuration se fait par fichiers de configurations qui sont utilisés par le Framework. Ces informations sont injectées au runtime, Spring étant un grand utilisateur de l’injection de dépendances.

En JEE nous avions le fichier persistence.xml. Pour Spring DATA MongoDB, un fichier de configuration Spring existe aussi est comporte des informations similaires comme les informations de connexion.

### Entités

Les entités ou documents (pour du NoSQL) sont déclinés en plusieurs package : DATA JPA, DATA MongoDB, …

Pour Spring DATA MongoDB la création de ces entités est donc similaire bien que différentes ; les annotations étant différente (Pas de normalisation).

**Par exemple :**

**Spring :**

@Document(collection="int\_data\_flow")

**public** **class** INTDataFlow {}

**Eclipse Link :**

@Entity

@NoSql(dataFormat = DataFormatType.***MAPPED***, dataType = "int\_data\_flow")

**public** **class** INTDataFlow {}

### CrudRepository et PageAndSortingRespository

Un apport très appréciable de Spring est le CrudRepository permettant de générer très rapidement des classes de gestion de données similaire à DAO.

Dans ce travail j’ai fait pour l’exemple l’implémentation de la couche DAO en utilisant un mécanisme de types génériques inspiré sur ce qui se fait maintenant dans Spring avec ces interfaces.

Le principe est de créer une interface étendant CrudRepository. L’implémentation sera ensuite générée par le Framework :

public interface INTBusinessFaultRepository extends PagingAndSortingRepository <INTBusinessFault, Serializable> { }

### REST

Spring met à disposition tous les outils nécessaires à la réalisation de service REST. Si l’on suit la méthode « traditionnelle » il faut créer une application Web Spring.

Il est désormais possible de le faire plus simplement en utilisant Spring Boot. Ceci permettra d’exposer les Repositories directement en service REST.

Il est à noter qu’Eclispe Link propose aussi une fonctionnalité similaire: jps-rs ; qui permet d’exposer très facilement les services REST.

# Conclusion

# Références

JPA

* <http://theopentutorials.com/post/uncategorized/how-to-create-a-ejb-3-x-project-using-maven-in-eclipse-part-1/>

Entity NoSQL

* <https://wiki.eclipse.org/EclipseLink/Examples/JPA/NoSQL>

DAO

* http://java.dzone.com/articles/jpa-implementation-patterns
* <http://stackoverflow.com/questions/3888575/single-dao-generic-crud-methods-jpa-hibernate-spring>
* <http://blog.xebia.com/2009/03/09/jpa-implementation-patterns-data-access-objects/>

REST

* <http://examples.javacodegeeks.com/enterprise-java/rest/resteasy/json-example-with-resteasy-jackson/>
* http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api

AngularJS

* https://docs.angularjs.org/guide
* <http://www.w3schools.com/angular/>

Livre sur AngularJS

* AngularJS de Brad Green, Shyam Seshadri, publié par O'Reilly en avril 2013; ISBN:978-1-4493-4485-6
* Mastering Web Application Development with AngularJS de Pawel Kozlowski et Peter Bacon Darwin, publié par Packt Publishing en Août 2013; ISBN 978-1-78216-182-0

Spring

* http://projects.spring.io/spring-data/
* http://www.programming-free.com/2014/07/spring-data-rest-with-angularjs-crud.html