# **Conception de la partie technique**

Toute application a besoin d'accéder à des ressources techniques telles que :

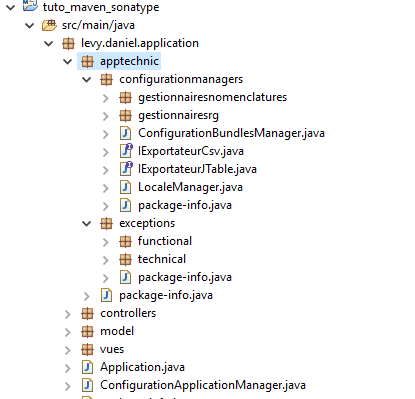
* des constantes,
* des properties,
* des nomenclatures,
* une Locale,
* des descriptions de fichiers,
* des exceptions typées pour les besoins de l'application ...

Ces ressources ne font pas authentiquement partie du métier, mais elles sont *indispensables au métier*.

Surtout, elles doivent être **accédées en tout point** de l'application.

Il n'est donc pas judicieux de positionner ces ressources dans les couches **Modèle**, **Vue** ou **Controller** (MVC). Elles peuvent être utilisées par n'importe quel élément de ces 3 couches.

Il est plus judicieux de créer une couche **appTechnic** à part *transverse* à toutes les couches applicatives (MVC).



## **Les ressources utilisées par l'application**

### **Les constantes applicatives**

Toute application peut avoir besoin d'utiliser des constantes applicatives.

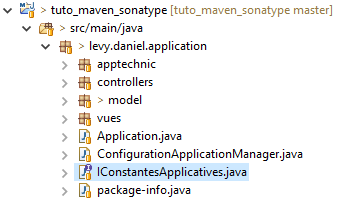
Par exemple, de nombreuses classes d'une application peuvent générer des fichiers csv utilisant le séparateur point virgule ';'.

Il peut être intéressant de mettre une fois pour toutes les constante telles que POINT-VIRGULE au niveau d'une Interface **IConstantesApplicatives**.

Cette interface est située sous la racine du package pricipal (levy.daniel.application) du classpath.



Diagramme N° 1 : Les constantes applicatives



### **Les properties**

Les application modernes utilisent des fichiers **.properties** pour permettre un *paramétrage de l'application postérieur au développement*.

Ces fichiers properties sont des fichiers textuels qui comportent des couples [clé = valeur].

Par exemple :

ressourcesexternes**=**D:/Donnees/eclipse/eclipseworkspace\_neon/tuto\_selenium\_complet/ressources\_externes

# application du contrôle sur le Pays. ------------------

interrogationmeteo.pays.actif=true

# application de la RG-Interro-Pays-01 : le pays doit être renseigné

interrogationmeteo.pays.rginterropays01.actif=true

# application de la RG-Interro-Pays-02 : le pays ne doit contenir que des lettres

interrogationmeteo.pays.rginterropays02.actif=true

* L'application référence en dur la clé.
* L'application va lire la valeur associée à la clé dans le properties au moment de l'exécution.
* *On peut changer librement les valeurs*. Par exemple, si une clé "label.bonjour" a pour valeur "bonjour" qui s'affiche dans un label d'une IHM de l'application, il suffit de changer la valeur "bonjour" en "bonjour tout le monde" pour modifier en conséquence le label dans l'IHM.
* *On ne doit pas changer les clés* car elles sont explicitement référencées par les classes Java.

L'utilisation de fichiers properties permet donc d'**externaliser** hors des classes Java des paramètres de l'application.

L'utilisation des properties permet également d'**internationaliser** l'application.

* dans un properties messages\_fr\_FR.properties, label.bonjour=bonjour permettra l'affichage de "bonjour" si on utilise la Locale fr\_FR
* dans un properties messages\_en\_US.properties, label.bonjour=hello permettra l'affichage de "hello" si on utilise la Locale en\_US

Certains properties sont **internes au classpath** et *ne peuvent être modifiés que par le centre serveur*. Il s'agit en général des properties de configuration de l'application.

D'autres properties doivent être **externes au classpath** et *peuvent être modifiés par la Maîtrise d'Ouvrage (MOA)*. il s'agit des properties contenant l'activation des contrôles, les messages librement adaptables par la MOA, ...

#### **Les properties INTERNES**

J'appelle properties *internes* les properties **devant figurer dans le classpath** de l'application.

Ces properties internes sont **incorporés dans le livrable** applicatif (jar ou war).

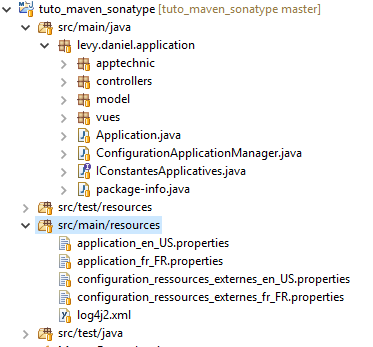
Ils figurent sous 'src' dans le cas d'un projet JSE 1.8 ou sous 'src/main/resources' dans la cas d'un projet MAVEN.

Ces properties ne sont **pas paramétrables par la MOA** et ne peuvent être modifiés que par le centre-serveur ou les développeurs lors de nouvelles livraisons.

Ces properties contiennent essentiellement les paramètres applicatifs.



Diagramme N° 2 : Les properties INTERNES



#### **Les properties EXTERNES**

J'appelle properties *externes* les properties ne **devant PAS figurer dans le classpath** de l'application mais au contraire dans un **REPERTOIRE hors du serveur applicatif**.

Ces properties externes ne sont **PAS incorporés dans le livrable** applicatif (jar ou war). Ils sont au contraire livrés à part au centre-serveur sous forme de dossier Windows (répertoire).

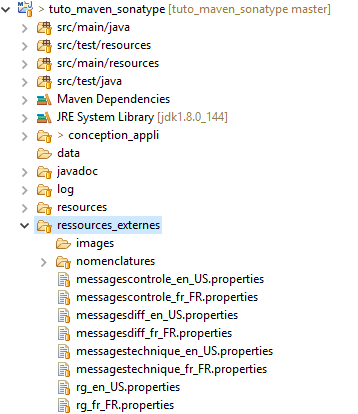
Le centre-serveur décide librement à quel endroit hors serveur applicatif il déploie ce répertoire. Il indique simplement la localisation de ce répertoire en dur au niveau de la clé "ressourcesexternes" du properties interne "configuration\_ressources\_externes.properties".

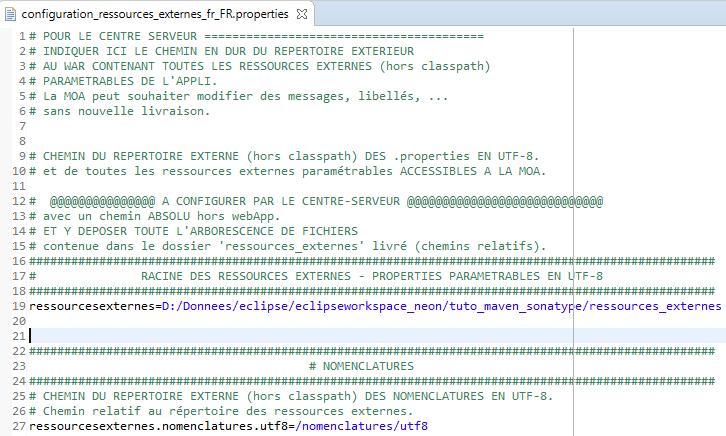
Ces properties sont **paramétrables par la MOA** et ne nécessitent aucune modification par le centre-serveur ou les développeurs (nouvelles livraisons).

Ces properties contiennent essentiellement les paramétrages des labels, activation des contrôles, messages, ...



Diagramme N° 3 : Les properties EXTERNES





##### **ressources\_externes**

## **Conception des ConfigurationManagers**

Toute application nécessite une phase de paramétrage et d'initialisation lorsqu'elle est déployée :

* Créer la base de données.
* Remplir les listes déroulantesdes Interfaces Homme-Machine (IHM) issues de la base de données (pays, régions, départements, villes, ...).
* Indiquer le path de la base de données dans les properties ou .xml de l'application.
* Créer le répertoire des ressources externes (hors classpath) qui doivent être accessibles à la Maîtrise d'Ouvrage (MOA). Ce répertoire peut contenir des fichiers paramétrables à tout moment par la MOA, des nomenclatures pour des valeurs finies (Autoroute, Route Nationale, Route Départementale, ...), des descriptions de fichiers, des logos, ...
* Créer les répertoires des logs (journaux écrits par l'application utiles pour la tierce maintenance applicative (TMA)).
* ...

Une fois l'application correctement déployée, tous ces concepts (logos, images, chemins, ...) doivent être accessibles en tout point de l'application.

La méthode consistant à créer une couche à part baptisée '**apptechnic**' offrant des SINGLETONS à toute l'application fournissant une unique instance des images, logos, chemins vers les ressources, ... est retenue.

La couche '**apptechnic**' contient des **'ConfigurationManagers'** chargés de lire une fois pour toutes les fichiers de configuration, puis de les rendre disponibles à toute l'application.

La couche '**apptechnic**' contient également les Exceptions typées de l'application. Ces Exceptions qui doivent remonter jusqu'à la couche SERVICE, puis à la couche CONTROLLER pour affichage à l'utilisateur dans des vues sont également transverses à l'application. Il ne semble donc pas judicieux de les inclure dans une des couches **Modele**, **Vues**, **Controllers**.



Diagramme N° 4 : Conception des ConfigurationManagers

### **contexte (racine)**

#### **levy.daniel.application**

##### **appTechnic**

###### ***configurationmanagers***