Projet Cloud Integration

Idée initiale : former un mini carnet d'adresses.

- Répertoire « DataIn » : données des utilisateurs et des adresses au format JSON.
- Répertoire « DataOut » : aurait dû contenir les données stockées par l'appli, sous format JSON également.
- Répertoire *src.main.java.com.RichemeVyas.App* : le Main
- Répertoire *src.main.java.com.RichemeVyas.Domain* : les objets représentant les utilisateurs et les adresses (pour la désérialsiation) ainsi que les listes de ces objets.
- Répertoire *src.main.java.com.RichemeVyas.Services* : les services appelés.
- Répertoire *src.main.java.resources* : contient le fichier **config.xml** décrivant l'ensemble des channels utilisés.

config.xml

```
<!-- get data from dataIn -->
<int-file:inbound-channel-adapter
        directory="dataIn"
        filename-pattern="*.json"
        channel="JsonInputChannel">
    <int:poller fixed-delay="2000"/>
</int-file:inbound-channel-adapter>
```

Dans un premier temps, on cherche à importer l'ensemble des données JSON.

```
<!-- split the single json message into multiple messages -->
<int-file:splitter id="splitter" apply-sequence="false" first-line-as-header="true"
                   input-channel="JsonInputChannel" output-channel="SplittedJsonChannel"
                   auto-startup="true"/>

<int:channel id="SplittedJsonChannel"></int:channel>

<!-- route messages based on the value of their headers -->
<!-- I didn't find a way to make the value field work as a regex... e.g. newUser\d+.json and newAddress\d+.json :( -->
<int:header-value-router id="router" input-channel="SplittedJsonChannel" header-name="file_name" resolution-required="false">
    <int:mapping value="newUser1.json" channel="UsersInputChannel"/>
    <int:mapping value="newUser2.json" channel="UsersInputChannel"/>
    <int:mapping value="newAddress1.json" channel="AddressesInputChannel"/>
    <int:mapping value="newAddress2.json" channel="AddressesInputChannel"/>
    <int:mapping value="newAddress3.json" channel="AddressesInputChannel"/>
</int:header-value-router>
```

On split les données mises bouts-à-bouts en autant de messages qu'il y a de fichiers à l'origine. On route ensuite ces fichiers dans le bon channel : un pour les utilisateurs, un pour les adresses.

Nous aurions voulu que les valeurs indiquées dans les paramètres « value » de chaque *mapping* puissent fonctionner comme des regex – ce qui aurait donné seulement deux *mapping* avec pour value respectivement « newUser\d+.json » et « newAddress\d+.json » – mais malheureusement, nous n'avons pas obtenu le résultat escompté.

```
<!-- convert the json objects to Java objects that will be computed afterward -->
<int:json-to-object-transformer input-channel="UsersInputChannel" output-channel="usersChannel" type="com.RichemeVyas.Domain.User"></int:json-to-object-transformer>
<int:json-to-object-transformer input-channel="AddressesInputChannel" output-channel="addressesChannel" type="com.RichemeVyas.Domain.Address"></int:json-to-object-transformer>
```

On utilise des *transformers* pour changer les données brutes en objets Java manipulables.

Ici, nous avons rencontré un bug que nous ne sommes pas parvenus à résoudre (malgré le temps passé dessus…) : lors de l'exécution, il est indiqué qu'un constructeur est manquant pour pouvoir effectuer la désérialisation des donnés. Pourtant, lorsque nous avons ajouté un constructeur, l'erreur était toujours présente.
Cela nous a particulièrement étonné car que nous avons calqué notre utilisation du *json-to-object-transformer* sur celle que vous avez fourni dans le projet de base ; et nous avons également calqué nos classes du « domaine » sur leurs équivalents dans le projet de base.

Malheureusement donc, nous n'avons pas pu aller plus loin dans l'exécution de notre projet.

```
<!-- activation of both services to register users and addresses -->
<int:service-activator input-channel="usersChannel" output-channel="aggregatorChannel" ref="registrationServiceID" method="addNewUser"/>
<bean id="registrationServiceID" class="com.RichemeVyas.Services.RegistrationService"/>
<int:service-activator input-channel="addressesChannel" output-channel="aggregatorChannel" ref="addressesServiceID" method="addNewAddress"/>
<bean id="addressesServiceID" class="com.RichemeVyas.Services.AddressesService"/>
```

Néanmoins, nous avons cherché à obtenir un « fil » consistant, et avons donc ajouté deux *service-activator* qui auraient dû déclencher les services pour les utilisateurs et pour les adresses.

```
<!-- aggregation of data rteturn by both services -->
<int:aggregator id="myAggregator"
                input-channel="aggregatorChannel"
                output-channel="outputChannel"
                correlation-strategy-expression="payload.name"
                release-strategy-expression="size()==2">
</int:aggregator>

<int:channel id="outputChannel"></int:channel>

<int:logging-channel-adapter channel="outputChannel" level="INFO"/>
```

Enfin, nous avons conservé l'*aggregator* permettant de récupérer les messages de nos deux channels en parallèle pour les envoyer dans un seul et même channel, affichant ses messages en tant que logs de type « info ».