

Projet Mashup



Master 2 Informatique

Cours : Architecture distribuée

Enseignant : M. DIEGUEZ Lodeiro Martin

Étudiants

TOURE Boubacar

SYLLA Mohamed

Barro Julliette

I. Introduction

Ce projet consiste à mettre en place une application de type ‘Mashup’ en utilisant des technologies de services web(REST,SOAP) et des techniques de conception en couches (design patterns).

Une application Mashup est une application qui s'appuie sur des services à distance pour fournir des fonctionnalités aux différents utilisateurs. Il existe plusieurs applications web de type Mashup, les plus connues et les plus utilisées étant les sites de streaming et les sites de réservation de vol pour ne citer que ceux-là.

Dans notre cas, il s'agissait donc de mettre en place un CRM (Customer Relationship Management) permettant à des utilisateurs d'une entreprise de pouvoir récupérer des informations sur des potentiels clients en fonction de leur pouvoir d'achat et de l'état dans lequel ils vivent. Ces informations étaient disponibles à deux endroits différents, un api externe externe fourni par le service [Salesforce](#) et une API interne fournie par un service au sein de l'entreprise elle-même.

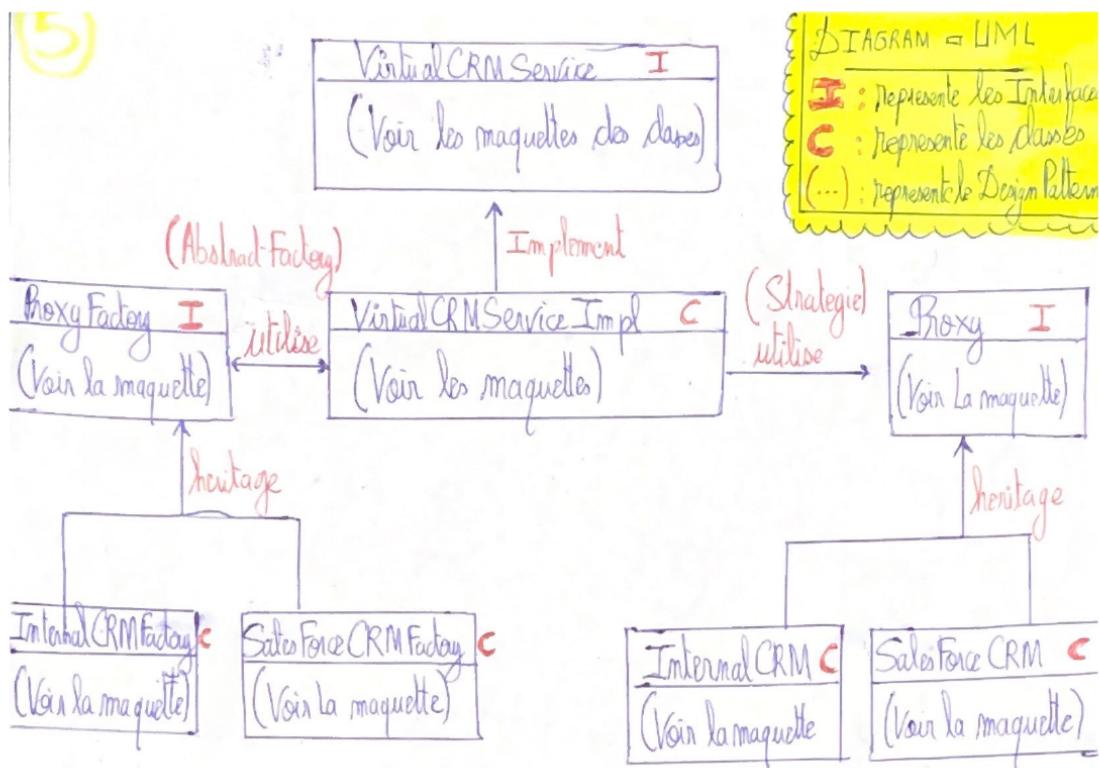
Le problème était donc de permettre aux différents utilisateurs de l'application d'obtenir leurs résultats sans se soucier de la provenance des informations.

Nous vous expliquerons dans les prochaines pages notre approche pour pallier ce problème afin de permettre aux utilisateurs d'utiliser une application simple et efficace.

II. Architecture globale

A. Diagramme de classe

Ce diagramme de classe est un diagramme UML de ‘haut niveau’ mettant en évidence les différentes classes et interfaces impliquées dans l’implémentation de **VirtualCRMService**.



B. Explications des différentes classes et interfaces

Pour visualiser la structure des différentes classes, voir Annexe.

1. Interfaces

- **VirtualCRMService** : Il s’agit d’une interface appelée par le contrôleur lorsqu’un utilisateur envoie une requête. Elle comprend deux méthodes `findLeads` et `findLeadsByDate` qui retournent toutes les deux des leads respectivement en fonction de la fourchette de salaire et de l’intervalle entre deux dates. Elle fait appel à l’interface **Proxy**.

- **Proxy** : Il s'agit d'une interface implémentée par nos différents proxys (*InternalCRM*, *Salesforce*), elle permet à l'interface *VirtualCRMService* d'appeler les méthodes de recherche des leads sans se soucier du type de CRM.
- **ProxyFactory** : Il s'agit d'une interface implémentée en suivant les règles du design pattern factory. Elle permet de créer un proxy.

2. Mappers

- **VirtualCRMMapper** : Il s'agit là d'une classe utilitaire créée pour nous permettre de mapper les données (transformation de String en data , de XMLGregorianCalendar en Calendar, etc).

3. CRM

- **InternalCRM** : c'est la classe qui permet de récupérer les informations des clients à partir du CRM interne, elle implémente l'interface proxy. Elle récupère grâce à une API SOAP créée par nos soins en vue de retourner les informations des clients présents dans le CRM interne.
- **SalesForceCRM** : c'est la classe qui permet de récupérer les informations des clients à partir de l'API Salesforce, elle implémente l'interface proxy.

Ces deux classes sont créées à travers le design pattern factory à l'aide des classes *InternalCRMFactory* et *SalesForceCRMFactory* et leur instance est unique grâce au design pattern singleton. Contrairement à *InternalCRM* qui utilise une API SOAP, nous avons utilisé une approche REST pour récupérer les données sur l'API fournie par Salesforce.

4.Factories

- **InternalCRMFactory** : permet de créer un *internalCRM*.
- **SalesForceCRMFactory** : permet de créer un *salesforceCRM*.

5. VirtualCRMServiceImpl

Cette classe est la plus importante de toutes les classes. Elle implémente *virtualCRMService* et permet donc de créer les différents proxy (*InternalCRM* et *SalesForceCRM*). Plusieurs patterns sont utilisés au sein de cette classe (singleton,factory et strategy).

6. Models

- **LeadTo et Lead** : ce sont les classes qui définissent les propriétés des clients (nom, prénom, adresse, société...). Lead a exactement les mêmes classes que LeadTo mais elle est une classe générée à travers un fichier XML schéma (xsd). A noter que la conversion d'un LeadTo vers un Lead se fait grâce à VirtualCRMMapper.
- **GeographicPointTo** : c'est une classe qui permet de trouver les coordonnées géographiques (la latitude et la longitude) d'un client à travers les informations de la classe Lead (state, street, postalCode...).
- **RSSFeed** : Il s'agit d'une classe permettant de créer le flux RSS. Ce flux RSS permet entre autres d'afficher les différents clients créées il y a moins de vingt quatre heures.
- **RSSFeedText et LeadInfo** : Ces deux classes nous ont plutôt servi pour l'affichage des différents résultats dans nos fichiers html.

C. Langage et framework utilisé

Nous avons utilisé le langage JAVA et plus précisément le framework SpringBoot qui nous a permis un temps de développement plus rapide grâce notamment aux annotations mais aussi à la génération des classes liées à InternalCRM et à la création des outils adéquats pour sa génération.

A noter aussi que toutes les dépendances liées au différents modules ont été ajoutées grâce à l'outil maven.

III. Explication des différentes approches

A. Mise en place des différents APIs

Pour pouvoir récupérer les différentes APIs nous avons utilisé une approche SOAP et une approche REST

- **REST** : L'API REST est une API fournie par salesforce pour avoir accès aux différentes données des clients présents sur son site. La requête HTTP se fait donc en GET et les réponses sont récupérées au format JSON et pour chaque client un **lead** (voir partie modèle) est créé.
- **SOAP** : Il s'agit du service web mis en place par nos soins pour pouvoir fournir aux utilisateurs les informations des clients présents dans InternalCRM. Pour cela, nous avons créé un schéma XML (fichier XSD dans lequel nous avons créé toute la structure des données et nous avons aussi mis en place un **WSDL** (Web service description Language) qui a permis de bien définir la structure et la forme des requêtes HTTP à envoyer pour les différentes recherches.

Ces deux APIs sont donc disponibles via des points d'entrées qui sont appelées par les différents proxys(Salesforce et InternalCRM) et leurs réponses nous permettent de créer les différents clients (*LeadTo*) en vue de les afficher dans le site web destiné à cet effet.

B. Site web

Au cours de notre phase de développement, nous avons fait le choix de vraiment mettre en place un site web permettant d'afficher les informations des clients de manières plus lisibles qu'un JSON ou un XML. En effet les informations retournées par les différentes API(Salesforce ,Internal) sont récupérées puis traitées et affichées dans du HTML pour qu'elles puissent être lues et comprises par tout un chacun. Ce site web se compose de plusieurs pages :

- **Une page form :**

Elle permet de pouvoir saisir plus facilement les critères que l'on veut pour findLeads et findLeadsByDate.

image du formulaire de saisie de findLeads et findLeadsByDate

- **Une page leads et une page leadsByDate :**

Ces pages affichent les informations concernant les clients qui correspondent aux critères saisis.

Customer	Annual Salary	City	Street	Phone
Mariellen Goseling	50000.0 \$	Huntington	50 Hintze Drive	304-581-6622
Clevey Rawne	235600.0 \$	Boca Raton	24 Summerview Drive	561-380-0787
Jamison Darrington	35000.0 \$	Jacksonville	25 Continental Trail	904-453-9673

Image de la liste des clients ayant un revenu annuel compris entre 5000 et 1000000

Au clic sur la carte correspondant à un client, un pop-up s'affiche pour afficher toutes les informations concernant le client. (Voir l'image suivante)

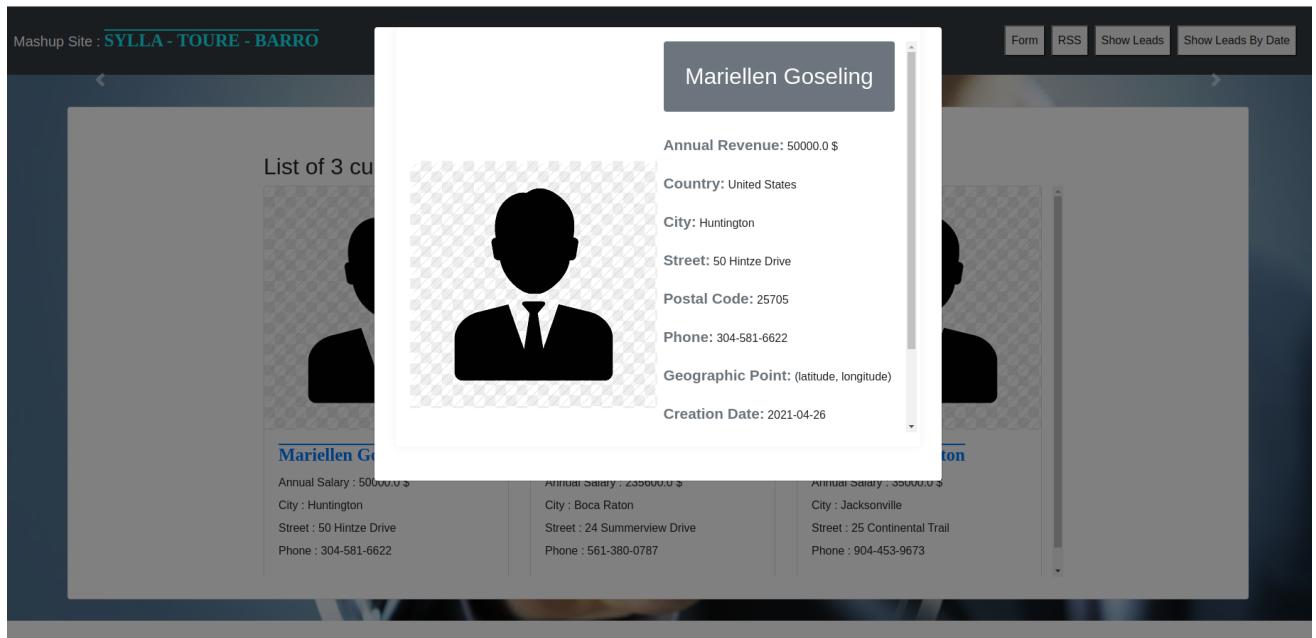


Image de la pop-up

• Service RSS (Last potential Clients)

Il s'agit d'un flux XML qui s'affiche au contact lorsque de nouveaux clients potentiels sont trouvés, il s'agit des clients créées il y a moins de 24 heures.

Ces informations sont affichées au format XML et au format Texte selon le souhait de l'utilisateur.

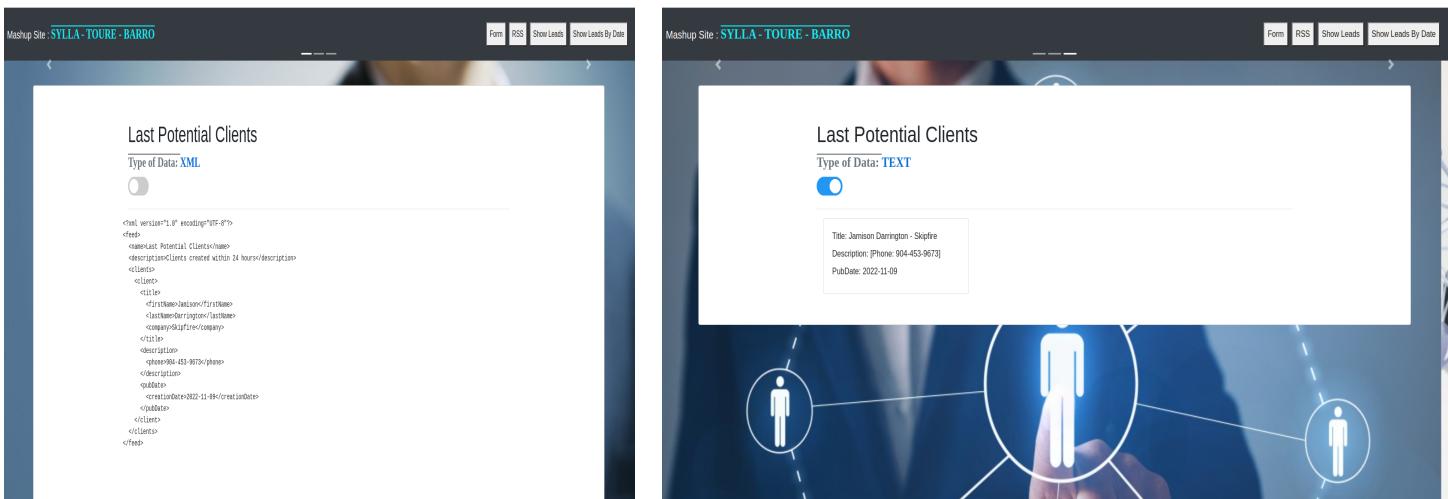


Image du flux RSS en XML et au format Texte

IV. Conclusion

En conclusion, nous devons avouer que rétrospectivement ce projet nous a permis d'atteindre pleinement les objectifs que nous nous étions fixées en mettant en avant nos connaissances dans de nombreux domaines tels que : la conception, la modélisation, la mise en place d'une architecture en couche qui est assez solide et en parfaite adéquation avec nos attentes.

En effet, ce mini projet nous a permis de comprendre et d'apprendre à maîtriser l'usage de API en s'inspirant de l'exemple des sites Mashup.

Nous sommes très heureux de vous annoncer notre fierté en vue du travail réalisé. Nous sommes par ailleurs convaincus que le travail élaboré n'est qu'une étape primaire aussi bien pour une carrière professionnelle que pour des études plus approfondies.

V. Annexe

Les Interfaces

Virtual CRM Service I	
+ findLeads(double, double, String);	: List<Lead>
+ findLeadsByDate(Calendar, Calendar);	: List<Lead>

Proxy I	
+ getLeads(double, double, String);	: List<Lead>
+ getLeadsByDate(Calendar, Calendar);	: List<Lead>

Proxy Factory I	
+ createProxy(): Proxy	(Factory)

Les Mappers:

Virtual CRM Mappers C	
+ mapGeographicPointToGeographic(...)	: Geographic
+ mapGeographicFromGeographicPointTo(...)	: GeographicPointTo
+ mapLeadFromLeadTo(Lead)	: LeadTo
+ mapLeadToFromLead(LeadTo)	: Lead
+ mapDateToXmlGregorianCalendar(...)	: XMLGregorianCalendar
+ mapStringToDate(String)	: Date
+ mapCalendarToDateString(Calendar)	: String

Zes CRM:

InternalCRM		C	(Singleton)
-	fakeDataOfLeads : List<Lead>		
-	apiRequest : String		
-	instance : InternalCRM		
+	getInternalCRM : InternalCRM		
+	getLeadsInFakeData(double, double, String) : List<Lead>		
+	getLeads(double, double, String) : List<Lead>		
+	recupInternalLead (HttpResponse<String>) : List<Lead>		
+	getLeadsByDate(Calendar, Calendar) : List<Lead>		
+	getLeadsByDateInFakeData(Calendar, Calendar) : List<Lead>		
+	getMillisBetween(Calendar, Calendar) : long		

SalesForceCRM		C	(Singleton)
-	... (Attributs de classes à voir code source)		
+	getSalesForceCRM() : SalesForce CRM		
+	getLeads(double, double, String) : List<Lead>		
+	getLeadsByDate(Calendar, Calendar) : List<Lead>		
+	recupLeads(JSONArray) : List<Lead>		
+	getSalesForceResponses (String) : JSONArray		
+	getUrlAndToken() : List<String>		

Les Factory (Implementation de l'interface "Proxy Factory")

<u>InternalCRMFactory</u>	<u>C</u>	(Factory)
+ createProxy() : Proxy		de type Internal CRM

<u>SalesForceCRMFactory</u>	<u>C</u>	(Factory)
+ createProxy() : Proxy		de type SalesForce CRM

Les Interfaces Impl :

<u>VirtualCRMServiceImpl</u>	<u>C</u>	3 types de design Pattern
+ proxyFactoryList : Hashtable<String, ProxyFactory>		(Singleton)
+ instance : VirtualCRMServiceImpl		(Stratégie)
+ proxyList : List<Proxy>		(Factory)
+ createProxy(String) : Proxy		
+ getVirtualCRMServiceImpl() : VirtualCRMServiceImpl		
+ findLeads(double, double, String) : List<Leads>		
+ findLeadsByDate(Calendar, Calendar) : List<Leads>		
:		

Le FakeData :

<u>FakeData</u>	<u>C</u>	(Singleton)
- (les attributs à voir dans le code source)		
+ getFakeData() : FakeData		
+ generateRandomDate() : XMLGregorianCalendar		
+ setDataToDate() : void		
+ generateData() : List<Leads>		

Les Models:

<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th colspan="2" style="text-align: left; padding: 5px;">GeographicPoint To C</th> </tr> </thead> <tbody> <tr> <td>- latitude</td> <td>: double</td> </tr> <tr> <td>- longitude</td> <td>: double</td> </tr> <tr> <td colspan="2">+ getter And Setter (-)</td> </tr> <tr> <td colspan="2">+ toString () : String</td> </tr> </tbody> </table> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th colspan="2" style="text-align: left; padding: 5px;">LeadInfo C</th> </tr> </thead> <tbody> <tr> <td>- salaryMinimum</td> <td>: double</td> </tr> <tr> <td>- salaryMaximum</td> <td>: double</td> </tr> <tr> <td>- dateStart</td> <td>: String</td> </tr> <tr> <td>- dateEnd</td> <td>: String</td> </tr> <tr> <td>- etat</td> <td>: String</td> </tr> <tr> <td colspan="2">+ getter And Setter (...)</td> </tr> </tbody> </table> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th colspan="2" style="text-align: left; padding: 5px;">Lead To C</th> </tr> </thead> <tbody> <tr> <td>• firstName</td> <td>: String</td> </tr> <tr> <td>• lastName</td> <td>: String</td> </tr> <tr> <td>• annualRevenue</td> <td>: double</td> </tr> <tr> <td>• phone</td> <td>: String</td> </tr> <tr> <td>• street</td> <td>: String</td> </tr> <tr> <td>• postalCode</td> <td>: String</td> </tr> <tr> <td>• city</td> <td>: String</td> </tr> <tr> <td>• country</td> <td>: String</td> </tr> <tr> <td>• Date</td> <td>: creationDate</td> </tr> <tr> <td>• geographicPointTo</td> <td>: GeographicPoint To</td> </tr> <tr> <td>• company</td> <td>: String</td> </tr> <tr> <td>• state</td> <td>: String</td> </tr> <tr> <td>• URI</td> <td>: String</td> </tr> <tr> <td colspan="2">+ getter And setter (...)</td> </tr> </tbody> </table>	GeographicPoint To C		- latitude	: double	- longitude	: double	+ getter And Setter (-)		+ toString () : String		LeadInfo C		- salaryMinimum	: double	- salaryMaximum	: double	- dateStart	: String	- dateEnd	: String	- etat	: String	+ getter And Setter (...)		Lead To C		• firstName	: String	• lastName	: String	• annualRevenue	: double	• phone	: String	• street	: String	• postalCode	: String	• city	: String	• country	: String	• Date	: creationDate	• geographicPointTo	: GeographicPoint To	• company	: String	• state	: String	• URI	: String	+ getter And setter (...)		<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th colspan="2" style="text-align: left; padding: 5px;">RSSFeed C</th> </tr> </thead> <tbody> <tr> <td>- leads</td> <td>: List<Lead></td> </tr> <tr> <td colspan="2">+ createFeedForClients () : Document</td> </tr> <tr> <td colspan="2">+ toString () : String</td> </tr> <tr> <td colspan="2">+ getPotentialClients (Document) : List< RSSFeedText ></td> </tr> </tbody> </table> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th colspan="2" style="text-align: left; padding: 5px;">RSSFeed Text C</th> </tr> </thead> <tbody> <tr> <td>- title</td> <td>: String</td> </tr> <tr> <td>- description</td> <td>: String</td> </tr> <tr> <td>- pubDate</td> <td>: String</td> </tr> <tr> <td colspan="2">+ getter And Setter (-)</td> </tr> </tbody> </table>	RSSFeed C		- leads	: List<Lead>	+ createFeedForClients () : Document		+ toString () : String		+ getPotentialClients (Document) : List< RSSFeedText >		RSSFeed Text C		- title	: String	- description	: String	- pubDate	: String	+ getter And Setter (-)	
GeographicPoint To C																																																																											
- latitude	: double																																																																										
- longitude	: double																																																																										
+ getter And Setter (-)																																																																											
+ toString () : String																																																																											
LeadInfo C																																																																											
- salaryMinimum	: double																																																																										
- salaryMaximum	: double																																																																										
- dateStart	: String																																																																										
- dateEnd	: String																																																																										
- etat	: String																																																																										
+ getter And Setter (...)																																																																											
Lead To C																																																																											
• firstName	: String																																																																										
• lastName	: String																																																																										
• annualRevenue	: double																																																																										
• phone	: String																																																																										
• street	: String																																																																										
• postalCode	: String																																																																										
• city	: String																																																																										
• country	: String																																																																										
• Date	: creationDate																																																																										
• geographicPointTo	: GeographicPoint To																																																																										
• company	: String																																																																										
• state	: String																																																																										
• URI	: String																																																																										
+ getter And setter (...)																																																																											
RSSFeed C																																																																											
- leads	: List<Lead>																																																																										
+ createFeedForClients () : Document																																																																											
+ toString () : String																																																																											
+ getPotentialClients (Document) : List< RSSFeedText >																																																																											
RSSFeed Text C																																																																											
- title	: String																																																																										
- description	: String																																																																										
- pubDate	: String																																																																										
+ getter And Setter (-)																																																																											

DIAGRAM - UML

- I** : représente les Interfaces
- C** : représente les classes
- (...) : représente le Design Pattern

