



# Projet : Mashup

19 septembre 2025

## 1 Introduction

Ce projet consistera à développer une application combinant différentes technologies de services web (principalement REST et RPC). Plus précisément, vous allez développer un « mashup » : une application web en suivant une approche basée sur la *conception par couches*. Le terme « mashup » désigne une application qui s'appuie sur un ensemble de services distants (pas nécessairement des services web) pour fournir ses fonctionnalités.

## 2 Première partie : VirtualCRMService et ses compagnons

Dans cette section, nous présentons la première partie du projet, dans laquelle plusieurs services web seront développés et d'autres seront simplement interrogés afin de développer une application *mashup*.

### 2.1 Contexte du projet

Pour définir le contexte du projet, nous supposerons que nous travaillons dans une entreprise qui, pour des raisons historiques, stocke les informations sur ses clients dans ses propres systèmes de gestion de la relation client (CRM<sup>(1)</sup>). Ce CRM est une application web installée sur l'un des ordinateurs de l'entreprise et utilise une base de données (probablement installée sur un autre ordinateur) pour stocker les informations sur les clients<sup>(2)</sup>.

Le modèle fournit le type de données (interne) ModelTO illustré à la figure 1. La couche de service est implémentée via Apache Thrift [1] et son interface est illustrée à la figure 1.

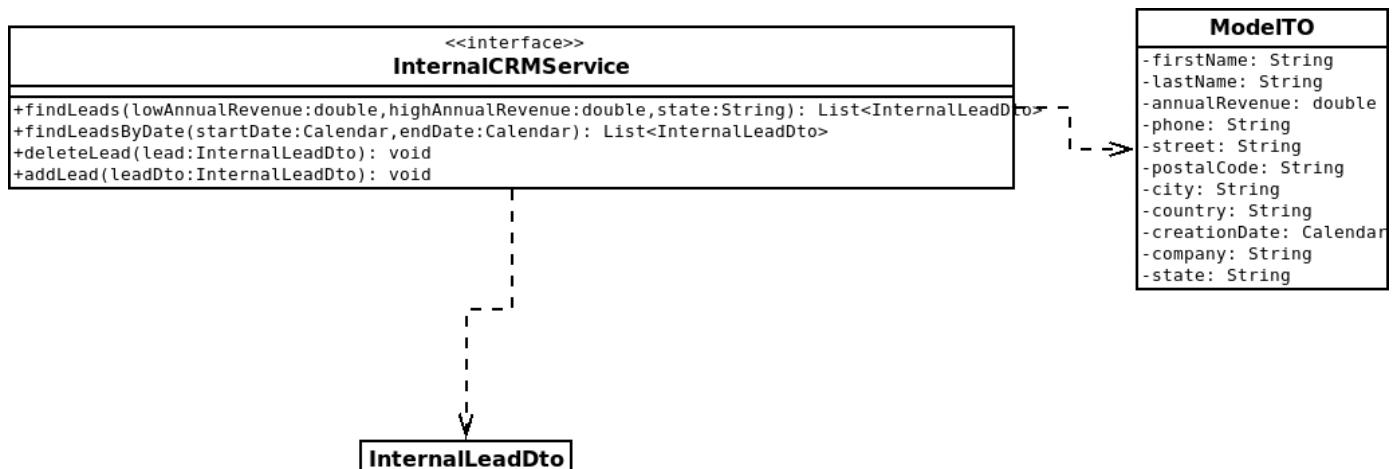


Figure 1 : Interface InternalCRM

<sup>1</sup> Voir [https://en.wikipedia.org/wiki/Customer\\_relationship\\_management](https://en.wikipedia.org/wiki/Customer_relationship_management).

<sup>2</sup>**Dans le cadre de ce projet, nous ne tiendrons pas compte de la modélisation de la base de données. Nous simulerons plutôt la base de données à l'aide d'une liste.**

Il y a un mois, notre entreprise a racheté une autre entreprise dans un domaine d'activité similaire. Cette entreprise avait souscrit à un service CRM auprès de Salesforce, de sorte qu'aujourd'hui, notre entreprise doit gérer deux services CRM différents.

Des entreprises telles que **Salesforce<sup>3</sup>** proposent des services CRM via Internet. Le CRM est géré par un prestataire externe.

entreprise (Salesforce dans le cas présent), et une API REST est publiée afin de fournir des opérations telles que l'ajout/la suppression de prospects ou la récupération de prospects.

Pour l'instant, l'entreprise indique qu'elle dispose des données clients dans les deux CRM. Pour certaines fonctions...

(probablement nombreuses), il serait souhaitable de disposer d'une application offrant une vue unifiée des données clients, évitant ainsi aux employés d'avoir à utiliser explicitement les deux CRM, ce qui serait très fastidieux. Notre entreprise a notamment décidé de créer des services web intermédiaires pour récupérer les informations clients (*Lead*). L'interface et les principales dépendances de ce service web sont illustrées à la figure 2. Ce service web, appelé Virtual CRM service, propose deux opérations pour trouver des prospects (par date et/ou par chiffre d'affaires) et délègue ces opérations à la fois au CRM interne et à Salesforce. Pour chaque prospect récupéré, le virtualCRMService ajoutera les coordonnées spatiales (longitude et latitude) liées à l'adresse du client (voir figure 2). Pour récupérer ces informations, le service virtualCRMService interroge OpenStreetMap<sup>(4)</sup> afin d'obtenir ces données à partir de l'adresse du Lead.

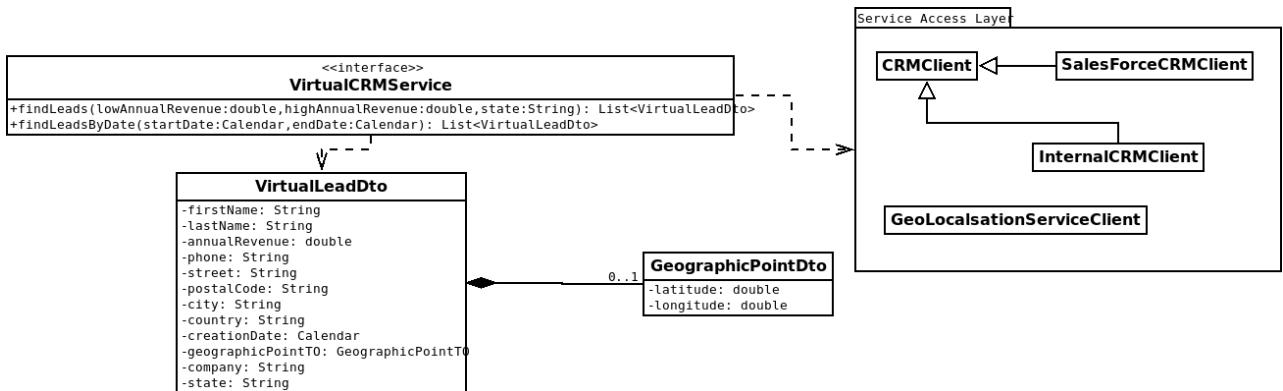


Figure 2 : Architecture générale du VirtualCRMService

La figure 3 montre l'architecture de l'application mashup à mettre en œuvre. Le code est structuré en un module : *virtualcrm*.

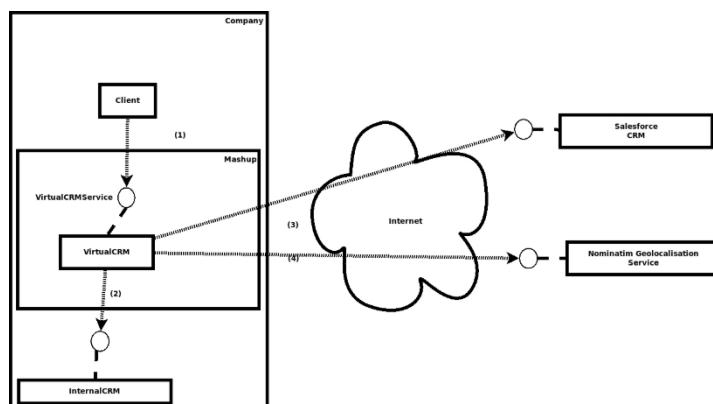


Figure 3 : Architecture de l'application mashup

<sup>3</sup> Voir <https://www.salesforce.com/fr/>

<sup>4</sup> <https://www.openstreetmap.org/>

Lorsque l'utilisateur exécute le client, nous appelons l'opération `findLeads(1)` de l'interface `VirtualCRMService` (Figure 2), fournie par le module « `virtualcrm` ». L'opération `findLeads` reçoit trois paramètres. Les deux premiers représentent la fourchette de revenus annuels que les clients doivent avoir (`lowAnnualRevenue < lowHighRevenue`), et le troisième le nom de la province dans laquelle ils sont situés. L'opération renvoie des informations sur les clients correspondant à ces critères de recherche.

Pour chaque client (`VirtualLeadDto`), nous renvoyons son nom, son prénom, l'entreprise pour laquelle il travaille, son revenu annuel prévu, son numéro de téléphone, son adresse, son code postal, sa ville, son département et son pays, la date à laquelle il a été enregistré dans le CRM et sa position géographique (`GeographicPointDto`). Si le service de géolocalisation n'est pas en mesure de renvoyer la position d'un client, l'attribut `geographicPointTO` doit être nul.

Le client doit conserver une seule instance (Singleton) de la classe implémentant l'interface `VirtualCRMService`. Ce comportement peut être obtenu grâce à l'utilisation de `Factory`. Dans ce cas, l'implémentation de cette interface doit initialiser l'état dans le constructeur et ne doit jamais le modifier pendant l'exécution des opérations de l'interface. de cette interface doit initialiser l'état dans le constructeur et ne doit jamais le modifier pendant l'exécution des opérations de l'interface.

La mise en œuvre de l'interface `VirtualCRMService` doit faire appel aux services de recherche du CRM interne (2) et de `SalesForce` (3) pour obtenir les données des clients potentiels, ainsi qu'au service de géolocalisation (4) pour obtenir leur position géographique.

Notez que l'accès au service web `Salesforce` nécessite une procédure d'authentification. Cette authentification n'a pas besoin d'être effectuée à chaque requête sur `Salesforce`. Au lieu de cela, la couche d'accès au service doit effectuer l'authentification lors de la création du `serviceClient` et, lorsque le jeton d'authentification expire, il doit être régénéré automatiquement.

L'interface doit renvoyer les informations de tous les clients (une liste d'objets de type `VirtualLeadDto`, voir figure 2), triées par profit potentiel (du plus élevé au plus faible). Indépendamment de l'origine du Lead (`salesforce` ou `internalCRM`), le lead sera converti en un type `VirtualLeadDto`.

Le client accédant à `virtualCRMService` consistera en une simple application en ligne de commande qui affichera les informations de chaque `Lead` récupéré sur la ligne de commande. Une approche similaire doit être mise en œuvre pour l'opération `findLeadsByDate`.

La section suivante explique les différentes parties de cet exercice qui doivent être mises en œuvre.

## 2.2 Service InternalCRM

Un service RPC sera implémenté avec Apache Thrift. Ce service fournit les opérations présentées dans la figure 1. En interne, il déléguera les opérations au modèle et exportera le type `InternalLeadDto` afin de masquer l'utilisation des types du modèle aux applications externes. De plus, il fournira des opérations pour créer et supprimer des prospects à partir du service. Ces opérations seront utilisées par une troisième application présentée dans la section 3.

Une caractéristique particulière mérite d'être mentionnée : le service `internalCRM` renverra (type `InternalLeadDto`) le prénom et le nom de chaque contact dans un seul champ en utilisant le format Nom, Prénom (par exemple Laporte, Aymeric). Au contraire, le type `VirtualLeadDto` de « `virtualCRMService` » a deux champs distincts pour le prénom et le nom respectivement.

## 2.3 Service Web Salesforce

`Salesforce` (<https://www.salesforce.com/>) propose à ses clients un service CRM et, depuis peu, notre entreprise y a accès. Notre `VirtualCRMService` peut facilement accéder et récupérer les informations contenues dans son CRM via une API REST [5].

Afin de faciliter le développement et le test d'applications qui accèdent à leurs données, `Salesforce` permet aux développeurs de créer des comptes développeurs identiques aux comptes réels, mais contenant des données fictives. En pratique, nous allons accéder aux données d'une entreprise fictive créée de cette manière. Pour ce faire, chaque groupe créera son propre compte développeur auprès de `Salesforce`.

Le modèle de données **Salesforce** se compose d'un certain nombre d'entités de données, chacune contenant différents types d'informations. Dans notre cas, nous utiliserons l'entité « Lead » qui contient diverses données sur les clients potentiels. Ces données comprennent divers éléments tels que l'adresse, la rue, l'État, le code postal ou le pays, ainsi que le chiffre d'affaires annuel de l'entreprise à laquelle appartient le contact (*AnnualRevenue*).

Pour interroger les données d'une entité, il est nécessaire d'utiliser la méthode de requête de l'API du service Web.

Cette méthode reçoit parmi ses paramètres une expression de requête écrite dans un langage appelé SOQL [6], qui est une version très simplifiée du SQL. Avant d'invoquer la requête, il est nécessaire de s'authentifier auprès du service en utilisant la méthode logique de l'API.

Vous trouverez divers exemples et ressources intéressants sur <https://developer.salesforce.com/>.

## 2.4 Service de géolocalisation

Plusieurs services Web de géolocalisation sont disponibles sur Internet [2, 3]. Certains d'entre eux, tels que **Google Maps**, nécessitent une clé API et fonctionnent selon un modèle de tarification *à l'utilisation*.

L'API Nominatim [3] d'**OpenStreetMaps** ne nécessite aucune inscription et fournit un service REST qui prend une adresse comme paramètre et renvoie la latitude et la longitude au format JSON ou XML, entre autres. Par exemple, si nous voulons obtenir l'emplacement de l'UFR (Figure refufr) des sciences de l'université d'Angers à l'aide de l'API *Nominating search* [4], il suffit d'appeler l'URL suivante (voir [4] pour plus de détails)

<https://nominatim.openstreetmap.org/search?&lt;params&gt;>

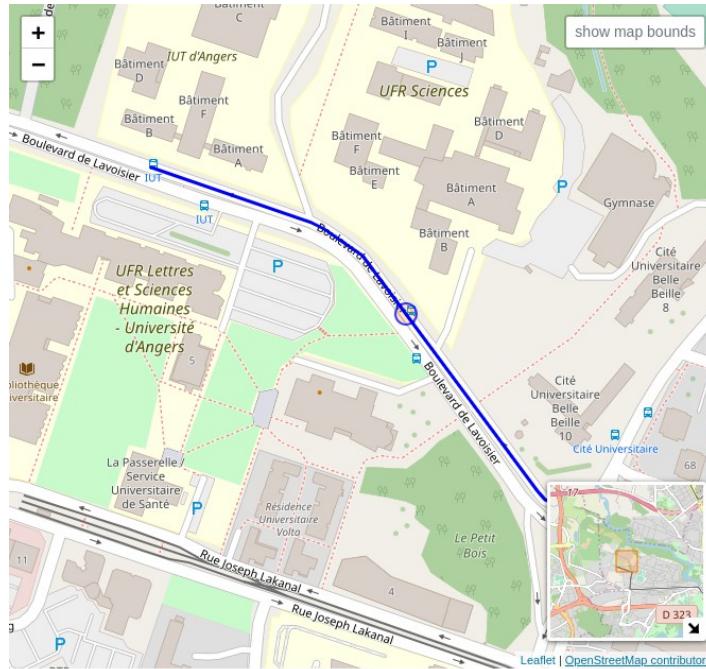


Figure 4 : Visualisation OpenStreetMap de l'UFR des sciences.

<https://nominatim.openstreetmap.org/search?city=angers&country=france&postalcode=49100&street=2+boulevard+de+lavoisier&format=json&limit=1>

et en analysant la réponse. Celle-ci peut être reçue au format JSON. Dans le cas de l'exemple, nous obtenons le JSON (simplifié) suivant (voir Figure 5) :

La réponse doit être analysée afin d'en extraire la **latitude** et la **longitude**.

```

▼ 0:
  place_id: 251271841
  licence: "Data © OpenStreetMap contributors, ODbL 1.0. http://osm.org/copyright"
  osm_type: "way"
  osm_id: 119321018
  lat: "47.4795093"
  lon: "-0.6003698"
  class: "highway"
  type: "residential"
  place_rank: 26
  importance: 0.10000999999999993
  address_type: "road"
  name: "Boulevard de Lavoisier"
  ▼ display_name: "Boulevard de Lavoisier, Belle-Beille, Angers, Maine-et-Loire, Pays de la Loire, Metropolitan France, 49045, France"
  ▼ boundingbox:
    0: "47.4782032"
    1: "47.4805084"
    2: "-0.6029448"
    3: "-0.5986745"

```

Figure 5 : Code JSON obtenu.

## 2.5 VirtualCRMService

Ce service récupère les informations provenant du service interne CRM, de Salesforce et du service de géolocalisation afin de fournir une interface commune pour la récupération des clients. Ce module est également un service web puisqu'il fournit une API implémentée avec Spring contenant deux opérateurs illustrés à la figure 2. L'architecture expliquée dans cette figure est très générale, une meilleure conception doit être élaborée à l'aide de modèles de conception (usines, injection de dépendances, façades, modèles, etc.).

## 2.6 Le client

Le client consiste en un simple outil en ligne de commande qui, à l'aide de l'API REST VirtualCRM, récupère les prospects des deux CRM et affiche leurs informations sous forme de texte dans la ligne de commande.

## 3 Deuxième partie : outil permettant de fusionner InternalCRM et Salesforce

À long terme, nous souhaitons utiliser un seul CRM : InternalCRM. C'est pourquoi nous allons développer un outil en ligne de commande qui récupérera tous les prospects de Salesforce et les ajoutera à InternalCRM via l'API Thrift. L'architecture générale de cet outil est illustrée à la figure 6.



Figure 6 : Architecture de l'outil permettant d'ajouter tous les prospects de Salesforce au CRM interne.

## 4 Évaluation

### 4.1 Groupes

Le projet sera mené en groupes de 3 personnes (de préférence).

### 4.2 Normes de codage

Afin d'écrire un code de haute qualité et facile à lire, un système de codage commun est utilisé, qui définit les règles de nommage des classes, des attributs et des méthodes, les règles d'identification, etc. Cela permet à une équipe de développement de maintenir l'apparence du code, quel que soit son auteur, ce qui facilite la maintenance.

Cela permet à une équipe de développement de conserver l'apparence du code, quel que soit son auteur, ce qui facilite la maintenance. La norme de codage [7] sera utilisée pour le projet. Les exemples présentés dans le cours suivent ce système de nommage.

L'utilisation d'un système de contrôle de version `git` sera obligatoire. `github` ou `gitlab`. Le lien vers le projet sera envoyé à l'enseignant par e-mail.

Enfin, il ne sera pas nécessaire de documenter les classes (par exemple, `JavaDoc`).

### 4.3 Soumission de la version finale

Une copie des sources du projet (contenant toutes les applications demandées) au format `.tar.gz` ou `.zip` et un rapport au format `.pdf` seront téléchargés sur `Moodle`.

En ce qui concerne les sources du projet, le fichier compressé doit contenir un projet pouvant être compilé via la ligne de commande `gradle`. Ce projet doit contenir les sources et les fichiers de configuration, mais il n'est pas nécessaire d'ajouter d'autres types de fichiers tels que `.class` ou `.war`.

### 4.4 Format du rapport

En règle générale, les diagrammes doivent utiliser la notation UML. Les diagrammes doivent être de haute qualité (et non issus d'une ingénierie inverse - la conception précède la mise en œuvre !) et doivent être expliqués de manière concise mais claire. La qualité du rapport sera essentielle pour l'évaluation du projet.

Le rapport comprendra les parties suivantes :

#### 1. Conception

- (a) **Architecture globale** : expliquez brièvement les modules qui ont été conçus et mis en œuvre. Deux diagrammes UML « de haut niveau » seront utilisés pour étayer l'explication.
- Un diagramme de classes illustrant les abstractions (interfaces, *fabriques*, classes d'implémentation d'interface, etc.) des différents modules impliqués dans l'implémentation de l'interface `VirtualCRMService`. Ce diagramme ne doit pas montrer les noms des opérations et des attributs dans les interfaces et les classes. Son objectif est d'illustrer les dépendances entre les principales abstractions.
  - Un diagramme de séquence illustrant le traitement d'un appel de l'opération `findLeads` de l'interface `VirtualCRMService`, en termes d'abstractions présentées dans le diagramme de classes ci-dessus.
  - Diagramme de séquence illustrant l'invocation de la méthode `mergeLeads` de la classe `LeadMerger`.
- (b) **i-ème service/module** : pour chaque service/module, il est nécessaire de rédiger une section expliquant sa conception. Chaque explication doit inclure :

- La structure globale du paquet de modules. Il n'est pas nécessaire d'utiliser la notation UML. Il n'est pas non plus nécessaire de montrer tous les paquets, mais seulement les paquets de niveau supérieur qui reflètent l'architecture du module et ses principaux sous-paquets. Pour chaque paquet, une brève explication de son contenu doit être fournie.
  - Un ou plusieurs diagrammes illustrant l'architecture du module. En particulier, les diagrammes spécifient en détail les opérations des interfaces et des classes pour les objets de transfert (*objet de transfert*). Pour le reste des abstractions (classes concrètes), il suffira de spécifier les opérations et les attributs jugés pertinents pour comprendre l'explication du module (par exemple, un attribut important pour comprendre l'explication d'une interface). En d'autres termes, les diagrammes UML ne doivent pas regrouper toutes les abstractions implémentées, mais uniquement celles qui sont pertinentes et, pour chacune d'entre elles, ne montrer que ce qui est nécessaire à la compréhension de la conception du module.
2. **Compilation et installation de l'application** : il convient d'expliquer clairement comment compiler et installer l'application, en partant du principe que l'environnement est correctement configuré (par exemple, serveur d'applications installé et configuré, progiciels installés dans les mêmes répertoires qu'en laboratoire, etc.). Les instructions d'installation doivent être aussi simples que possible. Elles doivent notamment préciser :
- Comment décompresser la distribution source
  - Tout aspect particulier de la configuration qui doit être souligné
  - Le projet doit pouvoir être exécuté à l'aide de gradle.
  - **Lors de la correction de la version finale de chaque application, les instructions d'installation seront suivies.**
3. **Problèmes connus** : liste des bogues/erreurs connus dans le code.

#### 4.5 Itérations et livraisons

Pour chaque application, une approche itérative sera suivie. Chaque itération intègre plus de fonctionnalités que la précédente, jusqu'à ce que la dernière itération aboutisse à un logiciel qui implémente toutes les fonctionnalités. En particulier, chaque implémentation sera réalisée en deux itérations.

- **La correction de la première itération ne sera pas prise en compte dans la note finale et il ne sera pas nécessaire de soumettre un rapport.**
- Il suffira de montrer les diagrammes liés à cette itération à l'aide d'un outil de conception UML.
- L'objectif de cette première itération est d'essayer de détecter les erreurs majeures et, si nécessaire, de guider l'apprenant vers leur résolution.
- La deuxième itération correspondra à la correction (finale).

#### 4.6 Évaluation

L'évaluation du projet tiendra compte :

- Son bon fonctionnement
- La qualité de la conception
- La qualité de la mémoire

Un projet copié signifie un échec pour le groupe qui a autorisé la copie et pour le groupe qui a copié, aucune distinction ne sera faite.

## Références

- [1] Apache thrift, <https://nominatim.org/release-docs/develop/api/Search/>
- [2] API de géocodage Google, Présentation de <https://developers.google.com/maps/documentation/geocoding/>
- [3] Nominatim : géocodage Open Street Map, <https://nominatim.org/release-docs/develop/api/Overview/>
- [4] Nominatim : géocodage Open Street Map, <https://nominatim.org/release-docs/develop/api/ Search/>
- [5] Guide du développeur de l'API REST Salesforce, [https://developer.salesforce.com/docs/atlas.en-us.api\\_rest.meta/api\\_rest/intro\\_rest.htm](https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/intro_rest.htm)
- [6] Soql, [https://developer.salesforce.com/docs/atlas.en-us.soql\\_sosl.meta/soql\\_sosl/sforce\\_api\\_calls\\_soql.htm](https://developer.salesforce.com/docs/atlas.en-us.soql_sosl.meta/soql_sosl/sforce_api_calls_soql.htm)
- [7] Java code conventions (1997), <https://www.oracle.com/technetwork/java/codeconventions-150003.pdf>